Complementos de Base de Dados

Resumos
2016/2017

João Alegria | 68661

Capítulo 1

Hadoop / HDFS

Hadoop

O que é o Hadoop?

O Hadoop é um software *open-source* desenvolvido pela a Apache, com alta escalabilidade, confiável, projectado para computação distribuída sobre o paradigma de programação map-reduce.

• ***É útil em:***

- Processamento distribuído de grandes conjuntos de dados em clusters
  (um cluster - ou agregado de computadores - é formado por um conjunto de computadores, que utiliza um tipo especial de sistema operativo classificado como sistema distribuído. Muitas vezes é construído a partir de computadores convencionais que estão ligados em rede e comunicam através do sistema, trabalhando como se fossem uma máquina única de grande parte)
- "scale-up" de um para milhares de servidores

• ***Quem utiliza o Hadoop atualmente e seus exemplos de utilização:***

- Adobe
- Facebook
- Yahoo
- Amazon
- eBay
- Spotify

Os exemplos de uso do Hadoop são analisar padrões dos utilizadores em sites de e-commerce (vendas online) e sugerir novos produtos para que eles possam comprar.
O Hadoop simplificou o processo paralelo uma vez que o desenvolvedor não precisa de se preocupar com problemas relativos ao processamento paralelo.
Escreve apenas as funções de como quer que os dados sejam processados.

• ***Quando se deve usar o Hadoop:***

- Em ficheiros muito grandes - ficheiros "muito grandes" neste contexto são arquivos com centenas de megas, gigas ou terabytes de tamanho.
- Transmissão de acesso de dados - o HDFS foi construído em torno da ideia de que o padrão de processamento de dados mais eficiente é a "*write-once*" - escrita única - e "*read-many-times pattern*" - padrão de várias leituras.
- Cada análise envolve uma grande proporção, se não toda, dos conjuntos de dados, portanto o tempo de ler todo o conjunto de dados é mais importante que a latência na leitura do primeiro registo (*record*).

• ***Componentes do Hadoop***

- O framework do Hadoop é formado por dois componentes principais:
  armazenamento e processamento

O primeiro é o HDFS (Hadoop Distributed File System), que manipula o armazenamento de dados entre todas as máquinas na qual o cluster do Hadoop está a ser executado.
O segundo, o Map-Reduce, manipula a parte do processamento da framework.

HDFS - Hadoop Distributed File System

O HDFS é um sistema de ficheiros escalável e distribuído, cuja arquitetura é fortemente baseada na GFS (Google File System), que também é um sistema de ficheiros distribuídos. Os sistemas de ficheiros distribuídos são assim necessários, uma vez que os dados tornam-se demasiados grandes para serem armazenados numa máquina só.
O HDFS armazena todos os ficheiros em blocos. O tamanho do bloco padrão é 64MB. Todos os ficheiros no HDFS possuem múltiplas réplicas, o que auxilia o processamento paralelo.

- ***Os clusters HDFS possuem dois tipos de nós:***

- **Namenode** - Administra o namespace do sistema de ficheiros. Ele gere todos os ficheiros e diretórios. Namenodes possuem o mapeamento de ficheiros e os blocos nos quais estão armazenados. Todos os arquivos são acedidos usando esses namenodes e datanodes.
    - **Namenodes secundários** - Este node é responsável por verificar a informação do namenode. Em caso de falha, podemos usar esse nó para reiniciar o sistema.

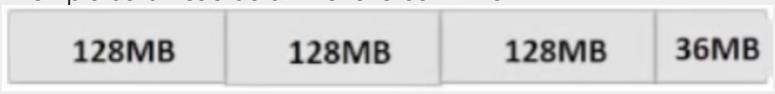
- **Datanode** - Armazena os dados em forma de blocos. Os datanodes comunicam aos namenodes sobre os ficheiros que possuem armazenados para que o namenode esteja ciente e os dados possam ser processados.

➡ Namenode é talvez o principal ponto crucial de falha no sistema.

Características do HDFS

- Sistema de ficheiros distribuídos
- Alta performance
- Alta disponibilidade
- Confiança
- HDFS corre em cima do sistema de ficheiros existente
- Projectado para tratar arquivos muito grandes, como padrões de acesso de dados de streaming
- HDFS utiliza blocos para armazenar ficheiros ou parte deles

Blocos

Ficheiros Blocos:	- Tamanho: 64MB (padrão), 128MB (recomendado) - 1 bloco HDFS é suportado por múltiplos "*OS blocks*" 
Vantagens dos Blocos:	- Tamanho fixo: Fácil de calcular quantos são armazenados num disco - Um ficheiro pode ser maior do que um disco - Se um ficheiro, ou parte dele, é menor que o tamanho do bloco, apenas o espaço necessário é utilizado. Exemplo da divisão de um ficheiro com 420MB: 

Vantagens do HDFS

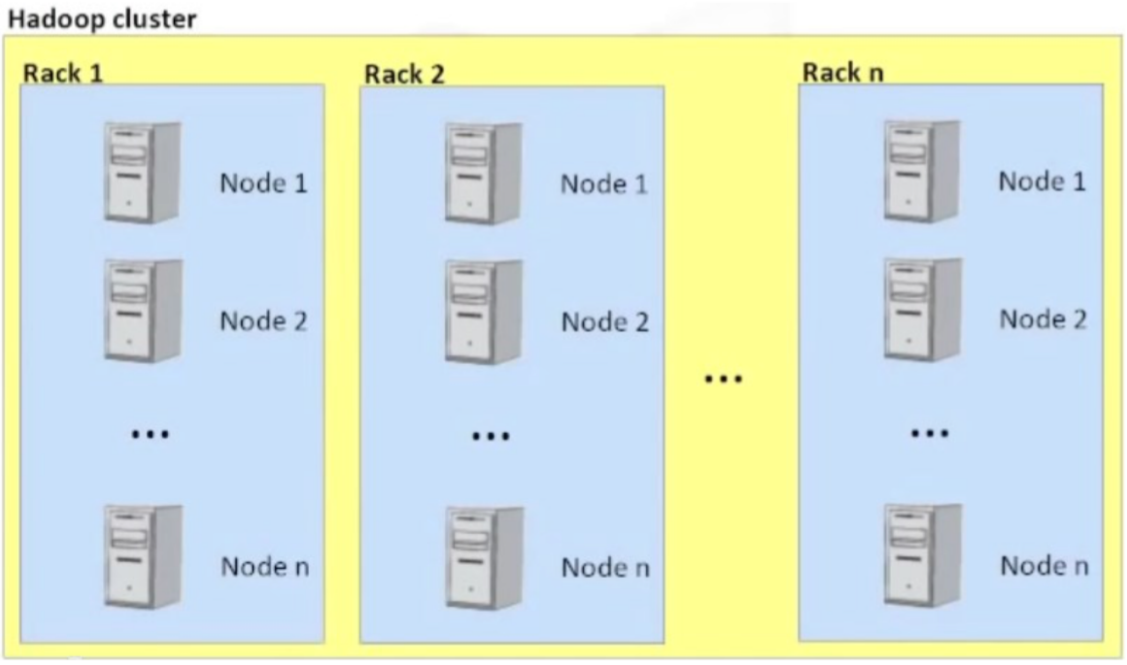
- ***Escalável***
  - apenas é necessário adicionar nós
  - evita “network bottleneck”

- ***Flexível***
  - todos os tipos de dados (documentos, registos, etc)
  - em todos os formatos (estruturado, semi-estruturado, não estruturado)
  - armazena qualquer informação

- ***Eficiente***
  - eficiência de custo (< $1000 por Terabyte)

Principais Entidades

Node	Rack	Hadoop Cluster
- Namenode (NN) - nó mestre (único) - Datanode (DN) - nó escravo (vários)	- Coleção de nodes conetados ao mesmo switch de rede - Pode existir mais que um rack - largura de banda entre racks > largura de manda entre nodes	- Coleção de racks



Map-Reduce

Map-Reduce é um paradigma de programação introduzido pelo Google para processar e analisar grandes conjuntos de dados

Neste paradigma, cada tarefa é especificada em termos de funções de mapeamento e redução. Ambas as tarefas rodam paralelamente no cluster. O armazenamento necessário para essa funcionalidade é fornecido pelo HDFS.

A razão para a escalabilidade desse paradigma é a sua natureza distribuída do funcionamento da solução. Uma grande tarefa é dividida em várias tarefas pequenas que são então executadas em paralelo em máquinas diferentes e então combinadas para solução que deu início.

Principais Componentes do Map-Reduce

• ***Job Tracker Node***

- Todos as tarefas de Map-Reduce são submetidas ao Job Tracker.
- O Job Tracker precisa de comunicar com o Namenode para conseguir os dados.
- O Job Tracker submete a tarefa para os nós Task Trackers.
- Esses Task Trackers precisam reportar-se ao Job Tracker em intervalos regulares, dizendo que estão "vivos" e a efetuar as suas tarefas.
- Caso o Task Tracker não se reporte, então o nó é considerado "morto" e o seu trabalho é direcionado para outro Task Tracker.
- O Job Tracker é um ponto crucial de falhas. Se o Job Tracker falhar, não é possível rastrear as tarefas.

• ***Task Tracker Node***

- O Task Tracker aceita as tarefas do Job Tracker. Essas tarefas são tanto de map, reduce ou ambas (*suffle*).
- O Task Tracker cria um processo JVM separado para cada tarefa a fim de se verificar que uma falha no processo não resulta numa falha de Task Tracker.
- Task Trackers também reportam-se ao Job Tracker continuamente para que este possa manter os registos de tarefa bem ou mal sucedidos.

Vantagens	Limitações
• **Barato** - Dimensiona até Petabytes, ou mais… • **Rápido** - Processamento paralelo de dados • **Melhor** - Adequado para problemas de BigData	• ***Escalável*** - Tamanho máximo do cluster: 4000 nós - Tamanho máximo de tarefas em simultâneo: 40 000 nós - Sincronização bruta no Job Tracker • ***Namenode é o único ponto de falha*** - A falha acaba com todos os trabalhos em execução na fila - Os trabalhos devem ser reenviados por utilizadores • ***Baixa utilização de recursos*** - Partição de recursos nos slots de map-reduce • ***Não suporta outros paradigmas*** - Suporta apenas map-reduce - Aplicações interativas que implementam o map-reduce são 10x mais lentas • ***Falta de protocolos compatíveis*** - O cliente e o cluster devem ser da mesma versão

Yarn - Yet Another Resource Negociator

O Yarn foi um grande avanço na versão 2 do Hadoop. Trouxe melhorias significativas de desempenho para algumas aplicações, oferece suporte a modelos de processamento adicional e implementa um mecanismo de execução flexível.
O Yarn é então um gestor de recursos que foi criado separando os mecanismos de processamento e recursos de gestão de map-reduce (que foi criado na versão 1 do Hadoop).
O Yarn é frequentemente chamado de sistema operativo do Hadoop porque é responsável pela gestão e monitoramento de cargas de trabalho, mantendo um ambiente de vários vizinhos, implementar controlos de segurança e gestão de recursos de alta disponibilidade do Hadoop.
Assim como um sistema operativo num servidor, o Yarn é projectado para permitir múltiplas e diferentes aplicações a correr numa plataforma de vários vizinhos.
No Hadoop 1, os utilizadores tinham a opção de escrever programas map-reduce em *Java*, *Python*, *Ruby* ou outras linguagens de script usando streaming ou usando o *Pig*, uma linguagem de transformação de dados. Independentemente do método que foi usado, esta fundamentalmente se baseou no modelo de processamento map-reduce para executar.
O Yarn suporta vários modelos de processamento para além do map-reduce. Um dos benefícios mais significativos é não estarmos limitados a nível de I/O, grande fonte de latência do map-reduce. Este avanço significa que os utilizadores de Hadoop devem estar familiarizados com os prós e contras dos novos modelos de processamento e entender quando aplicá-los aos casos de estudo particulares.

O Yarn executa vários tipos de trabalhos

- Processamento em tempo real
- ad-hoc OLAP
- map-reduce

O Yarn não é apenas um Hadoop 2.0 mas também:

- Framework para desenvolver e/ou executar aplicações de processamento distribuído

Requisitos atuais de Big Data

- Disponibilidade
- Confiança
- Utilização
- Compatibilidade “wire”
- Agilidade e evolução
  - Capacidade para os clientes controlarem as atualizações
- Escalabilidade
  - Clusters de 6000 a 10 000 máquinas

Capítulo 2

NoSQL

NoSQL

O que é o NoSQL?

O NoSQL é um movimento que promove soluções de armazenamento de dados não relacionais.
É composto por diversas ferramentas que, de forma particular e específica, resolvem problemas como:

- Tratamento de grandes volumes de dados
- Execução de consultas com baixa latência
- Modelos flexíveis de armazenamento de dados, como documentos XML ou JSON.

As tecnologias NoSQL não têm como objetivo substituir as bases de dados relacionais, mas apenas propor algumas soluções que em determinados cenários são mais adequadas. Desta forma, é possível trabalhar com tecnologias NoSQL e base de dados relacionais dentro de uma mesma aplicação.

- ***É útil em:***

  Cenários onde sistemas de base de dados tradicionais não são suficientes ou adequados às necessidades específicas, tais como:
  - baixa latência
  - grandes volumes de dados,
  - escalabilidade
  - ou estruturas em que as conexões entre os dados são tão importantes quanto o próprio dado.

- ***Para que serve:***

  As ferramentas NoSQL fornecem meios mais eficientes de armazenamento de grandes volumes de dados e/ou mecanismos de pesquisa de baixa latência, factores importantes que precisam ser considerados durante a escolha de uma solução de armazenamento de dados.

Porquê o NoSQL?

- ***Vantagens de base de dados NoSQL sobre base de dados relacionais:***

  As razões para as empresas a adoptarem num ambiente de base de dados NoSQL invés de uma base de dados relacional tem quase tudo a ver com os seguintes drivers do mercado e os requisitos técnicos:

Crescimento de Big Data

Big Data é um dos principais motivos do crescimento e popularidade do NoSQL para os negócios. A diversidade de tecnologias que recolhem dados e distribuem para vários dispositivos, multiplicam o crescimento dos dados.

Na verdade, uma das primeiras razões para usar NoSQL é porque teremos um projeto de Big Data, normalmente caracterizado por:

- Alta velocidade (de dados) - Dados que vêm rapidamente, possivelmente de diferentes locais.
- Variedade de dados - armazenamento de dados estruturados, semi-estruturados e não estruturados.
- Volume de dados - dados que envolvem muitos terabytes ou petabytes em tamanho.
- Complexidade de dados - dados que são armazenados e geridos em diferentes locais ou em data-centers.

Localização real independente

O termo “localização independente" significa a capacidade de ler e gravar numa base de dados, independentemente de onde a operação I / O ocorre fisicamente e ter qualquer funcionalidade de escrita propagada para fora a partir desse local, para que ele esteja disponível para os utilizadores e máquinas em outros sites. Tal funcionalidade é muito difícil de projetar em base de dados relacionais. Algumas técnicas podem ser implementadas, como arquiteturas de mestre / escravo, podem satisfazer a necessidade da localização de operações de leitura independentes, mas a escrita de dados em todos os lugares é uma questão diferente, especialmente quando esses volumes de dados são elevados.
Outros cenários onde a independência do local é uma vantagem são muitas e incluem atendimento aos clientes em muitas regiões geográficas diferentes e que precisam para manter os dados locais a esses sites para acesso rápido.

Disponibilidade de dados contínua

No mercado de hoje, onde a concorrência é apenas um clique de distância, o tempo de inatividade (downtime) pode ser mortal para a reputação de uma empresa. As falhas de hardware podem e irão ocorrer, felizmente ambientes de base de dados NoSQL são construídas com uma arquitetura distribuída para que não haja pontos únicos de falha e implementação de métodos redundantes dos dados. Se um ou mais servidores da base de dados, ou "nós" for abaixo, os outros nós do sistema são capazes de continuar com as operações sem perda de dados, mostrando, assim, tolerância a falhas verdade.
Desta forma, ambientes de banco de dados NoSQL são capazes de fornecer disponibilidade contínua seja em locais simples, através de data-centers ou na nuvem. Quando implementado de forma adequada, base de dados NoSQL pode fornecer alto desempenho em escala maciça, que nunca vão abaixo.
Isto é extremamente benéfico como atualizações do sistema ou modificações sem ter que colocar a base de dados offline. Este facto atrai a atenção de empresas que prestam serviços a clientes que esperam que as suas aplicações estejam sempre disponíveis e onde o tempo de inatividade equivale perdas de imenso dinheiro.

Capacidade transacionais modernas

O conceito de transações parece estar a mudar na era da Internet, e tem sido demonstrado que as transações ACID já não são uma exigência em sistemas de bases de dados orientadas.
À primeira vista, essa afirmação parece exagerada, como integridade transacional é uma característica da maioria cada sistema de dados - especialmente aqueles com requisitos de informação que exijam precisão e segurança.
No entanto, o que isto se refere não é ao comprometimento dos dados, mas sim às aplicações modernas que garantem a consistência transacional entre sistemas amplamente distribuídos.
O "C" no ACID refere-se a consistência de dados em sistemas de gestão de base de dados relacional que é aplicada através de chaves estrangeiras / restrições de integridade referencial. Este tipo de consistência não é utilizada em sistemas de gestão de dados progressivas, tais como base de dados NoSQL, porque não há operações JOIN, pois isso exigiria uma aplicação mais rígida da consistência. Em vez disso, a "consistência" que diz respeito à base de dados NoSQL é encontrado no teorema de CAP, o que significa a consistência imediata ou eventual de dados em todos os nós que participam numa de dados distribuída.

Modelos de dados flexíveis

Uma das principais razões que as empresas optem por um sistema de base de dados NoSQL de um Sistema de Gestão de Base de Dados Relacional (SGBDR) é devido ao modelo de dados mais flexível que é encontrado na maioria das base de dados NoSQL.
O modelo de dados relacional é baseado em relacionamentos definidos entre tabelas, que são definidas por uma determinada estrutura de coluna, todos estes guardados num schema - tudo muito rigoroso e uniforme.
Os problemas começam a surgir com o modelo relacional em torno de escalabilidade e desempenho ao tentar gerir os grandes volumes de dados que estão a tornar-se um facto da vida num moderno ambiente de IT e de negócios.
Um modelo de dados NoSQL - muitas vezes referida como sem schema - pode suportar muitos desses casos de uso e outros que não se encaixam bem num SGBDR.
Uma base de dados NoSQL é capaz de aceitar todos os tipos de dados - estruturados, semi-estruturados, não estruturados e - muito mais facilmente do que uma base de dados relacional que contam com um esquema predefinido. Esta característica de uma base de dados relacional pode ser um obstáculo à flexibilidade porque um schema predefinido rigidamente determina como são organizados os dados de uma base de dados. Muitas das aplicações de negócios de hoje, na verdade, têm a capacidade de fazer cumprir as regras sobre o uso de dados tornando-se uma plataforma de base de dados sem schema uma opção viável.
Finalmente, os factores de performance entram em jogo com um modelo de dados SGBDR, especialmente quando estão envolvidas imensas entradas e se tentam atualiza-las, o que pode ter implicações reais sobre o desempenho. No entanto, um modelo de dados NoSQL lida facilmente com tais situações e proporciona um desempenho muito rápido para operações de leitura e escrita.

Melhor Arquitetura

Outro motivo para usar uma base de dados NoSQL é porque é necessário uma arquitetura mais adequado para uma aplicação particular. É fundamental que as organizações adotem uma plataforma NoSQL que lhes permite manter os seus dados num contexto das suas aplicações. Algumas, mas nem todas as soluções NoSQL fornecem arquiteturas modernas que podem resolver o tipo de aplicações que requerem altos níveis de escalabilidade, a distribuição de dados e disponibilidade contínua.

Inteligência de negócios e análises

Um driver estratégico da implementação de um ambiente de base de dados NoSQL é a capacidade de extrair os dados que estão sendo captados a fim de derivar conhecimentos que coloca o seu negócio em uma vantagem competitiva. Extraindo inteligência de negócios de volumes de dados é uma tarefa muito difícil de alcançar com os sistemas de base de dados relacionais tradicionais. Bases de dados modernas NoSQL não só fornecem armazenamento e gestão de dados de aplicações de negócios, mas também entregar análises de dados integrados que fornecem compreensão instantânea de conjuntos de dados complexos e facilitar a tomada de decisão flexível.

Tipos de Base de Dados NoSQL

Existem quatro tipos genéricos de base de dados NoSQL, cada um com seus próprios atributos específicos:

Base de Dados orientada a Grafos	Base de Dados Key-Value
Baseada na teoria dos grafos, estas bases de dados são projetadas para dados cujas relações estão bem representados como um grafo e tem elementos que estão interligados, com um número indeterminado de relações entre eles. Exemplo: Neo4j e Titan.	Base de dados projetada para armazenar dados de uma forma sem esquema (*schema-less*) e uma das menos complexas opções de NoSQL. Num armazenamento chave-valor, a totalidade dos dados consiste num par chave-valor indexado, daí o nome. Exemplos: Cassandra, Dyanmo DB, Azure Table Storage (ATS), Riak, Berkeley DB.
Base de Dados Colunares	**Base de Dados orientada a Documentos**
Base de dados orientada a colunas, onde a informação deixa ser armazenada em linhas e são concebidas para armazenar tabelas de dados como secções de colunas de dados. Embora esta simples descrição soe como o inverso de uma base de dados padrão, as base de dados colunares oferecem um desempenho muito elevado e uma arquitetura altamente escalável. Exemplos: HBase, BigTable e HyperTable.	Expande a ideia básica de armazenamento “key-value” onde "documentos" são mais complexos, na medida em que contêm dados e a cada documento é atribuída uma chave única, que é usado para recuperar o documento. Estes são projetados para armazenamento, recuperação e getão de informações orientado a documentos, também conhecido como dados semi-estruturados. Exemplos: MongoDB e CouchDB.

A tabela a seguir apresenta alguns dos principais atributos que devem ser considerados ao avaliar os bancos de dados NoSQL.

Tipo de BD NoSQL	Performance	Escalabilidade	Complexidade	Funcionalidade
Key-Value	Alta	Alta	Nenhuma	Variável (Alta)
Colunares	Alta	Alta	Baixa	Mínima
Documentos	Alta	Variável (Alta)	Baixa	Variável (Baixa)
Grafos	Variável	Variável	Alta	Teoria de Grafos

Capítulo 3

Arquiteturas de Sistemas de BD

Sistema Centralizado

Um sistema centralizado é aquele que corre numa só máquina e não interage com outros computadores.

Composto por 1 host e vários terminais que possuem pouco poder de processamento que ficam conectados a um grande PC (*mainframe*) que possui a capacidade de processar e armazenar dados. Esta arquitetura tem como característica a necessidade de grandes investimentos tanto na aquisição como manutenção.

Vantagens	Desvantagens
- Os hosts fornecem alto grau de segurança, concorrência e controlo de cópias de segurança e recuperação. - Não há necessidade de um diretório distribuído, já que todos os dados são localizados num único host. - Não existe a necessidade de junções distribuídas, já que todos os dados são localizados num único host.	- Todos os acessos aos dados realizados por outro que não seja o host onde a BD está, gera alto custo de comunicação. - O host em que a BD está localizada pode criar um "engarrafamento" dependendo da quantidade de acessos simultâneos - Pode acontecer problemas de disponibilidade dos dados se o host onde os dados estão armazenados ficar offline

Existem dois modelos de sistemas centralizados:

1) Single-user System (Utilizador Único) **-** sistema operativo associado a um único utilizador

2) Multi-user System (Multi Utilizador) **-** sistema operativo de múltiplos utilizadores, que estão conectados ao sistema através de terminais.

Sistema Cliente-Servidor

O funcionamento deste sistema pode ser dividido em:

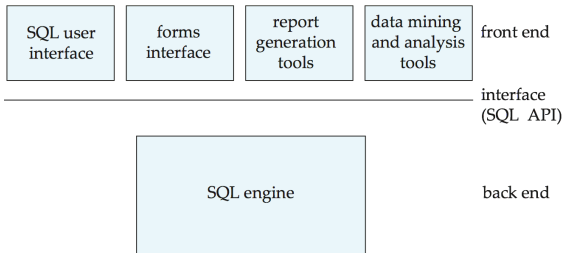
1) Back-end (Servidor) **-** Gere estruturas de acesso, avaliação de consultas e otimização, controlo de concorrência e recuperação.

2) Front-End (Cliente) **-** Consiste em ferramentas tais como formulários, relatórios e interface gráficas de utilizador.

A interface entre o front-end e o back-end é através de SQL ou através de uma interface de um programa.

Vantagens de substituição de *mainframes* com redes de estações de trabalho ou computadores pessoais ligados a máquinas de servidores back-end:

- Melhor funcionalidade para o custo
- Flexibilidade na localização de recursos e expansão de instalações
- Melhores interfaces de utilizador
- Manutenção mais fácil

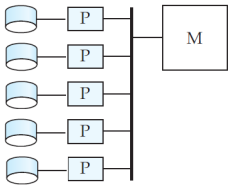
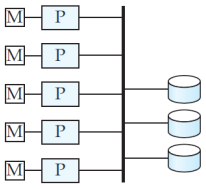
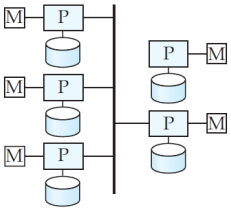
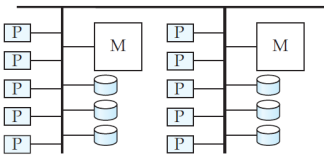


Sistema Paralelo

Num sistema paralelo é utilizado o paralelismo de vários servidores que trabalham simultaneamente sobre as suas respetivas base de dados para a obtenção de um rápido resultado. No paralelismo, os servidores são alinhados e cada um é responsável pelas sub-tarefas de uma base de dados.

• Duas principais medidas de desempenho:
- rendimento - número de tarefas que podem ser concluídas num determinado intervalo de tempo
- tempo de resposta - quantidade de tempo que leva para completar uma única tarefa a partir do momento em que são fornecidas

Arquiteturas das Base de Dados Paralelas

Memória Partilhada	Disco Partilhado
- Processadores e discos acedem a uma memória comum, através de uma rede de interconexão - Comunicação extremamente eficiente entre processadores - dados na memória compartilhada podem ser acedidos por qualquer processador sem precisar movê-lo, utilizando o software. - *Desvantagem*: Arquitetura não-escalável 	- Todos os processadores podem aceder diretamente a todos os discos através de uma rede de interconexão, mas os processadores têm memórias privadas - Arquitetura tolerante a falhas - se um processador falhar, os outros processadores podem assumir as suas funções, desde que a base de dados seja residente num dos discos que estejam acessíveis aos processadores - *Desvantagem* - Ocorre engarrafamento na interligação ao subsistema de disco - Podem ser dimensionado para um número maior de processadores, mas a comunicação entre processadores é mais lenta. 
Nenhuma Partilha	**Hierárquica**
- Nó é composto por um processador, memória e um ou mais discos. Os processadores num nó comunicam-se com outro processador em outro nó, usando uma rede de interconexão. Um nó funciona como um servidor para os dados no disco ou discos que possui o nó. - Escalável até milhares de processadores sem interferência. - *Desvantagem* - custo de comunicação e acesso ao disco não-local; o envio de informação envolve a interação de software em ambos os pontos. 	- Combina as características das arquiteturas de memória partilhada e de disco partilhado - Nível superior é atingido numa arquitetura sem partilha, onde os nós estão conectados numa rede de interconexão, e não partilham disco e memória com o outro. - Cada nó do sistema podia ser um sistema de memória partilhada com alguns processadores. - Alternativamente, cada nó pode ser um sistema de disco partilhado, e cada um dos sistemas de partilha de um conjunto de discos pode ser um sistema de memória partilhada. 

Base de Dados Distribuídas

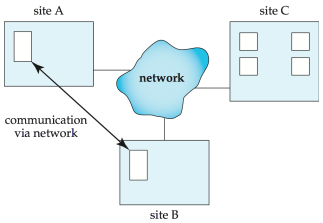
Consiste em múltiplos servidores, cada um responsável pelos seus dados. Estes dados podem ser acedidos utilizando uma rede, e apesar de estarem fisicamente distribuídos, devem apresentar-se ao utilizador logicamente integrados, como se fossem um único servidor.

> Um sistema de bases de dados distribuídas consiste em múltiplos servidores, cada um responsável pelos seus dados. Estes dados podem ser acedidos utilizando uma rede e, apesar de estarem fisicamente distribuídos, devem apresentar-se ao utilizador logicamente integrados, como se estivessem num único servidor.

A **principal razão** para utilizar base de dados distribuídos é o facto do problema em análise ter natureza distribuída. Por exemplo: Uma organização tem sede em Coimbra, uma fábrica A em Braga, um armazém B em Aveiro e uma fábrica C em Évora, pode fazer sentido utilizar uma BDD, pois espelha a estrutura da organização.
Não se devem utilizar quando não faz sentido ter um servidor em vários locais. Por exemplo: Uma empresa em sede na zona alta de Coimbra e três lojas na baixa, não faz sentido ter um servidor em cada local. Portanto se o problema não for de natureza distribuída, não se deve utilizar uma BDD. A solução mais adequada aqui era cliente/servidor.

Cada servidor que participa na BDD tem uma autonomia local, ou seja, é administrado separado e independentemente dos outros servidores. Esta autonomia local tem várias vantagens:

- Os nós dos sistemas podem espelhar mais facilmente a estrutura lógica das organizações
- Os dados locais são controlados por um administrador local, pelo que o domínio de administração é menor e mais manejável
- A recuperação de falhar pode, em muitos casos, ser efetuada numa base estritamente local
- Os nós do sistema podem ter a dimensão mais adequada a cada problema local e crescer independentemente.



BDD Homogéneas	BDD Heterogéneas
- Todos os sites têm um software igual - Estão conscientes uns dos outros e acordam em cooperar no processamento dos pedidos dos utilizadores - Cada site aceita perder parte da sua autonomia em termos de direitos de mudança de esquema e software - Objetivo: Dar a noção ao utilizador que o sistema é único	- Sites diferentes podem usar esquemas e softwares diferentes - diferenças nos esquemas constituem problemas complicados no processamento de consultas - diferenças no software constituem problemas complicados no processamento de transações - Os sites podem não estar conscientes uns dos outros e poderão fornecer apenas capacidades limitadas para a cooperação no processamento de transações - Objetivo: Integrar base de dados existentes para fornecer funcionalidade útil

- Diferença entre transações locais e globais:
  - Transações Local - Uma transação local acede a dados no local onde a transação foi iniciada
  - Transações Global - Uma transação global pode aceder a dados em localizações diferentes daquelas onde a operação foi iniciada

Vantagens BDD	Desvantagens BDD
- Partilha de dados - os utilizadores podem aceder a dados residentes num site diferente - Autonomia - cada site pode manter um nível de controlo sobre os dados guardados localmente - Maior disponibilidade do sistema através da redundância - como há replicação dos dados, se um site falha o sistema continua ativo	- A complexidade aumenta pois é necessário assegurar uma coordenação correta entre os sites, o que resulta em: - maior custo no desenvolvimento do software - maior possibilidade de bugs - aumenta a carga do processamento
Vantagens da autonomia de cada servidor da BDD As BDDs estão divididas por diversas sub-redes, cada sub-rede contém um servidor que funciona de modo autónomo, isto é, cada servidor pode executar tarefas independentemente dos outros servidores, mas o resultado final vai ser obtido juntando a informação que cada servidor contém.	

Soluções implementadas

- Atomicidade dos dados em dois ou mais nós
- Protocolo de duas fases:
  - Cada transação é executada e só aceite por um coordenador
  - Cada localização segue a decisão de um coordenador

Tipos de Rede

• Local-area networks (LANs)

- Constituída por processadores que estão distribuídos em pequenas áreas geográficas, como um único edifício ou alguns edifícios adjacentes.
- Maior largura de banda e Baixa Latência

• Wide-area networks (WANs)

- Constituída por processadores distribuídos numa grande área geográfica.
- Baixa largura de banda e Maior latência

Indicadores de Desempenho

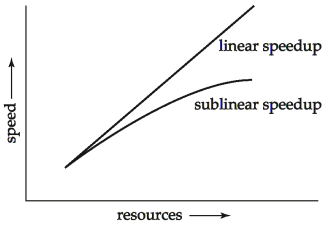
- Capacidade: O número de tarefas que são concluídas num determinado intervalo de tempo
- Tempo de resposta: A quantidade de tempo que leva para completar uma única tarefa a partir do momento em que são fornecidas

Acelerar e Aumentar (Speed-Up e Scale-Up)

• Acelerar - Speed-Up

- Um problema é executado num sistema pequeno e é recolhido por um sistema N vezes maior.

$$\text{Medido por: speedup} = \frac{\text{tempo decorrido no sistema menor}}{\text{tempo decorrido no sistema maior}}$$

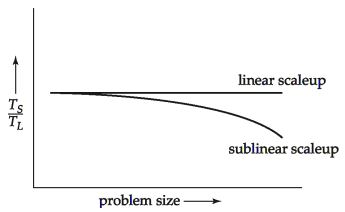


- Speed-Up é linear se a equação for igual a N.

• Aumentar - Scale-Up

- Aumenta tanto o tamanho do problema como o de sistema
- Sistema N vezes maior para realizar operações N vezes maiores

$$\text{Medido por: scaleup} = \frac{\text{tempo decorrido no problema menor}}{\text{tempo decorrido no sistema maior}}$$



- Scale-Up é linear se a equação for igual a 1.

- Batch Scale-Up
- Tarefa grande, típico na maioria das consultas de apoio à decisão e simulação científica.
- Necessário usar um computador N vezes maior em problemas N vezes maiores

- Transaction Scale-Up
- Consultas submetidas por utilizadores independentes a uma base de dados partilhada
- Adequado para execução paralela

• Fatores Limitantes

Speed-Up e Scale-Up são frequentemente sublinear devido a:
- Custos de iniciação: custo de iniciar vários processos pode dominar o tempo de computação, se o grau de paralelismo for alto.
- Interferência: Processos que acedem a recursos compartilhados
- Inclinação: Aumentar o grau de paralelismo aumenta a variância em tempos de serviço de tarefas em execução paralelamente. Tempo total de execução é determinado pela mais demorada das tarefas paralelas em execução.

Estrutura de Base de Dados Distribuídas

Problema: Tomar decisões sobre a localização de dados e programas nos nós de uma rede de computadores, assim como possivelmente projetando a rede em si.

Sistema Gestão Base Dados Distribuídas (SGBDD)

Nos SGBDD, a localização das aplicações engloba:
- Localização do software SGBDD
- Localização das aplicações que executam sobre a base de dados

Distribuição dos Dados

• Nível de partilha:
- sem partilha: Cada aplicação e os seus dados são executados em um site e não há comunicação com qualquer outro programa ou acesso aos dados dos outros sites
- partilha de dados: Todos os programas são replicados em todos os sites, mas os dados não. Deste modo as solicitações do utilizador são manipuladas no site de onde são originárias e os dados são movidos pela rede
- partilha de dados + programa: Ambos os dados e o programa podem ser partilhados independentemente do site.

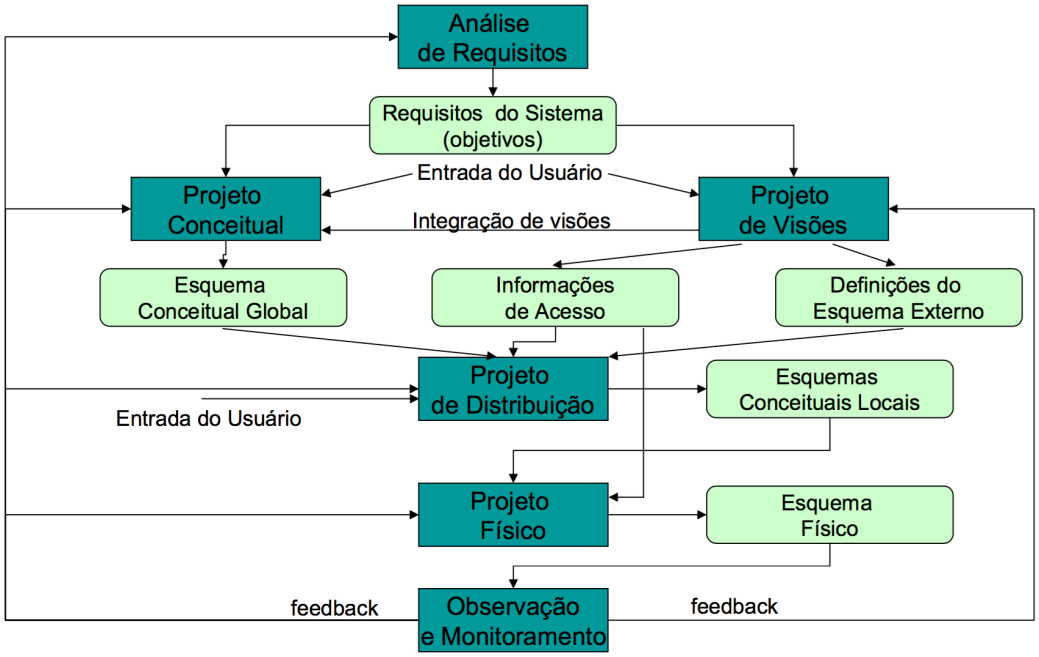
• Padrões de Acesso:
- estático: os padrões de acesso de pedidos de utilizador não mudam ao longo do tempo
- dinâmico: os padrões de acesso de pedidos de utilizador podem mudar ao longo do tempo

• Nível de Conhecimento dos Padrões de Acesso:
- nenhuma informação: os designers não têm nenhuma informação de como os utilizadores irão aceder à base de dados (muito improvável de acontecer)
- informação completa: os designers têm a informação completa, ou seja, os padrões de acesso podem ser razoavelmente previstos e não se desviam significativamente destas previsões
- informação parcial: onde existem desvios em relações às previsões

Estratégias para a Estrutura da BDD

Top-Down

- É uma abordagem mais adequada para SGBDs integrados e distribuídos homogeneamente
- A atividade começa com uma análise de requisitos que define o ambiente do sistema e deduz as necessidades de dados e processamento de todos os utilizadores potenciais da BD
- O estudo de requisitos também especifica onde o sistema final deve ficar com respeito aos objetivos de um SGBDD. Estes objetivos são definidos tendo em conta a performance, fiabilidade, disponibilidade, economia e flexibilidade
- O documento de requisitos é a entrada para duas atividades paralelas:
  - vista do design - definir a interface para os utilizadores finais
  - design conceptual - processo pelo qual a empresa é examinada para determinar os tipos e as relações das entidades. Este processo por ser dividido em dois grupos de atividades relacionadas:
    - análise de entidades - determinação, atributos e relação das entidades
    - análise funcional - funções fundamentais
- O esquema concetual global e a informação do padrão de acesso recolhida como resultado da visão do design são entradas para a etapa design da distribuição
- O objetivo nesta fase, que é a fase deste capítulo, é desenhar o esquema concetual local distribuindo as entidades pelos sites do sistema distribuído
- As entidades correspondem a relações
- Em vez de distribuir relações, dividimos em sub-relações, a que se chama de fragmentos, que estes sim irão ser distribuídos
- Assim a atividade do design de distribuição consiste em dois passos:
  - Fragmentação
  - Alocação e Replicação
- O último passo é o design físico, que mapeia o esquema concetual local para os dispositivos de armazenamento fisico disponíveis nos sites correspondentes
  - A entrada para este processo são:
    - Esquema concetual local e o padrão de acesso
- O resultado é feedback que pode resultar em fazer backup numa das etapas anteriores no design



Fragmentação

Processo de divisão de tuplos das tabelas em duas ou mais tabelas (fragmentos)

- **Informação necessária:**
  - Informações da BD
  - Informações da Aplicação
  - Informações da Comunicação da Rede
  - Informações do Sistema do Computador

- **Informação usada para decidir fragmentação:**

Quantitativa	Qualitativas
- Localização do Site - Frequência de Consultas - Onde a consulta (query) é executada - Queries seletivas	- Tipo de acesso a dados - Ler, escrever

- **Porquê fragmentar?**

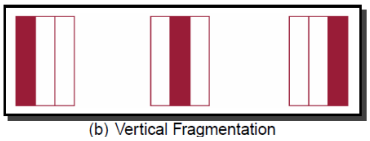
Vantagens	Desvantagens
- Fiabilidade - Performance - Custos de Comunicação - Capacidade de Armazenamento e Custo - Segurança	- Se os fragmentos não são mutuamente exclusivos - Degradação da performance nos *joins*

- **Fragmentos de relações como unidades de distribuição**
  - Aplicações precisam de um sub-conjunto de relações
  - Transações podem ocorrer ao mesmo tempo
  - Execução paralela de uma única query
  - Fragmentos pequenos - melhor desempenho
  - Degradação do desempenho dos *joins*
  - Controlo da semântica dos dados é mais difícil

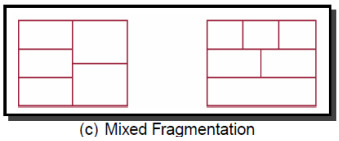
- **Tipos de Fragmentação**

Horizontal	Vertical
Os tuplos de uma relação são divididos em sub-conjuntos e cada sub-conjunto é alocado a uma BD diferente (mantém as colunas)	Produz um conjunto de relações distintas a partir da mesma relação

(a) Horizontal Fragmentation



(b) Vertical Fragmentation



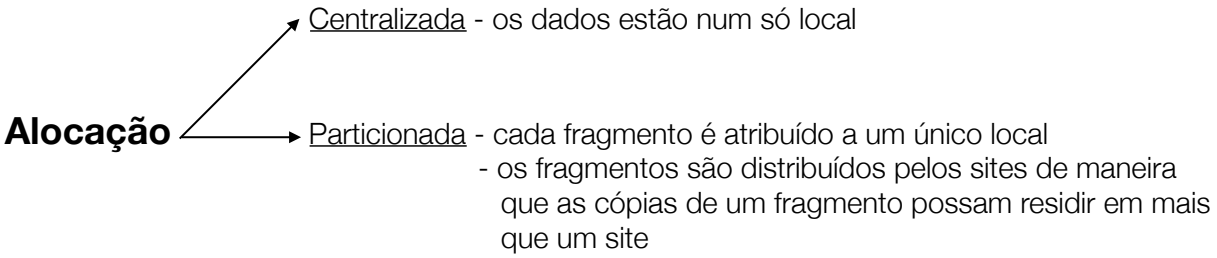
(c) Mixed Fragmentation

- **Regras de Correção da Fragmentação / Regras de Semântica**

Completude	Decomposição de uma relação R em fragmentos R_1, R_2, ..., R_n é completa sse cada item dos dados em R pode ser encontrado em pelo menos um fragmento R_i
Reconstrução	Verifica se é possível reconstruir a tabela original a partir da fragmentação resultante
Disjunção	Se uma relação R é decomposta em fragmentos R_1, R_2, ..., R_n , e o item de dados d_i está em R_j , então d_i não está em qualquer outro fragmento. Ou seja, não há dados repetidos (excepto nas PK da Fragmentação Vertical

Alocação

- Assumindo que a base de dados está fragmentada corretamente, é necessário decidir sobre a alocação dos fragmentos nos vários sites da rede.
- Quando são alocados, podem ser replicados ou mantidos como cópia única.
- As razões para a replicação são:
  - fiabilidade
  - eficiência de queries “read-only”
- No entanto, a execução de “update queries” causam problemas, uma vez que o sistema tem de assegurar que todas as cópias dos dados são atualizadas corretamente.
- Uma base de dados não replicada contém fragmentos que são alocados nos sites e só existe uma cópia de cada fragmento na rede.
- Na execução da alocação deve-se ter em conta dois aspetos:
  - custo mínimo
  - performance - minimizar o tempo de resposta e maximizar o rendimento do sistema em cada site



Replicação

- O sistema mantém múltiplas cópias de dados, guardando em diferentes sites para recolha mais fácil e tolerância a falhas. O fragmento pode ser replicado ou único.
- Uma relação ou fragmento está replicado se está guardado redundantemente em dois ou mais sites

A replicação pode ser caracterizada em:

- replicação total: Os fragmentos estão disponíveis em todos os sites
- replicação parcial: Alguns fragmentos podem ser replicados enquanto outros não, ou seja, um fragmento disponível em alguns sites
- sem replicação: Um fragmento alocado em apenas um site

Vantagens	Desvantagens
- Disponibilidade (no caso de falhas) - Paralelismo - Redução de dados a transferir	- Aumento do custo de atualizações - Aumento do controlo de concorrência (atualizações concorrentes em réplicas podem levar a inconsistência de dados)

Botton-Up

- Processo de integração de base de dados já existentes e independentes
- As BDs devem estar ligadas entre si para realizar tarefas entre si
- Razões:
  - Migração entre plataformas
  - Mudança organizacionais
  - Evolução da tecnologias

Capítulo 4

Processamento de Consultas

Tem como função:

- Transformar uma query de alto nível (cálculo relacional) numa query equivalente de baixo nível (álgebra relacional)
- Escolher uma estratégia de processamento de queries com menor custo de recursos computacionais
- Buscar entre várias transformações equivalentes a mais correta

No caso da BD distribuída, a consulta deve ser estendida com operações de comunicação e otimização.
Existem 3 passos para atingir:

- Analisar e Traduzir: Verificar a sintaxe e as relações
  Traduzir (decompor) numa expressão de álgebra relacional equivalente
- Otimização: Plano de avaliação ideal (localização dos dados)
- Execução: Execução do plano e avaliação

O sucesso de SGBDR (Sistemas de Gestão de Base de Dados Relacionais) é devido a:

- Linguagem de queries declarativas e fácil
- Tecnologias de processamento de queries avançada

A transformação deve atingir:

- Correção: Fácil de alcançar através do mapeamento bem definido do cálculo relacional para álgebra relacional
- Eficiência: É difícil de alcançar pois é difícil selecionar a estratégia de execução que minimiza o consumo de recursos

Processos de Consultas Distribuídas

- **Objetivos:**
  - Dar ao utilizador a impressão de que a query é realizada numa única BD
  - Transformar uma query de alto nível definida para um BD distribuída (que parece ser uma única BD) numa estratégia de execução eficiente (expressa em linguagem de baixo nível) a ser executada em BD locais

- **Otimização de Consultas:**
  - Para uma mesma consulta de alto nível podem existir muitas estratégias de execução diferentes - aquela que minimiza o consumo de recursos deve ser a escolhida

- **Medidas de Consumo:**
  - Minimizar o custo de: I/O + CPU + custo de comunicação
    - redes de comunicação mais rápidas (LAN e WAN)

- **Operações da álgebra relacional:**
  - A álgebra relacional é a base para expressar o processamento de uma consulta
    - A complexidade dos operadores da álgebra relacional afetam diretamente o tempo de execução de uma query e ditam alguns princípios úteis ao processamento
    - A complexidade é definida pela cardinalidade
    - Os operadores mais seletivos devem ser executados primeiro (reduzem a cardinalidade)
    - Operadores devem ser ordenados pela complexidade de forma crescente (produto cartesiano deve ficar para último

Problemas no processo de Otimização das Queries

Problemas a considerar na otimização das queries:

- Técnica: procura exaustiva, heurística, híbrida
- Tempo: quando a query é otimizada (estática, dinâmica)
- Estatísticas: número de tuplos, o seu tamanho e o número de colunas
- Decisão dos sites: centralizado, distribuído, misturado (distribuição de como os elementos estão dispostos)
- Topologia da Rede: LAN ou WAN
- Fragmentos replicados
- Uso de *semijoins*: reduz o tamanho de *joins* e o custo de comunicação

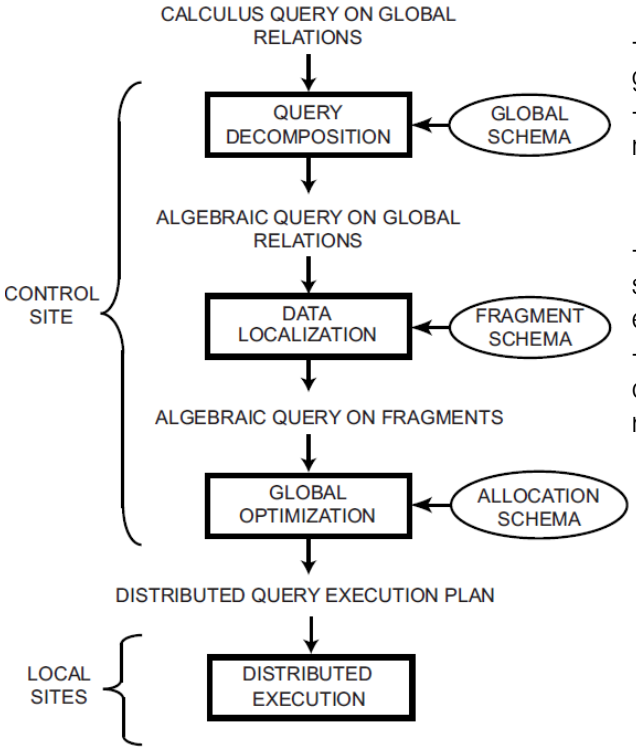
Técnica	- Procura exaustiva - elevado custo para manter mas ótimo - Heurística - baixo custo mas não ótimo - Híbrida - é construída uma query estática e a adaptação é abaliado em tempo de execução

Tempo	- Estático
	  - A otimização da query ocorre em tempo de compilação
	  - O custo pode ser amortizado por conta das várias execuções da query
	  - É apropriado para usar com o método de pesquisa exaustiva
	- Dinâmico
	  - A otimização da query ocorre em tempo de execução
	  - É possível a qualquer ponto da execução escolher a melhor estratégia de execução considerando o conhecimento preciso dos resultados
	  - Desvantagem: A otimização fica cara
	  - É apropriado para usar com o método de pesquisa híbrido

Decisão dos Sites	- Centralizado
	  - Um único site toma a decisão
	  - Requer o conhecimento de toda a BD distribuída
	- Distribuído
	  - Vários sites tomam a decisão
	  - Requer apenas informações locais
	- Híbrida
	  - Um único site toma a decisão mais importante e os restantes tomam decisões locais

Estatística	- As estatísticas para a otimização de consultas são definidas com base nos fragmentos, incluindo a cardinalidade e tamanho dos fragmentos, bem como o tamanho e o número de valores distintos de cada atributo - Atualizações periódicas das estatísticas levam a uma “re-otimização” no caso da otimização estática
Fragmentos Replicados	Para fins de fiabilidade é útil ter fragmentos replicados - Alguns algoritmos de otimização exploram a existência de fragmentos replicados em tempo de execução visando diminuir o custo de comunicação - O algoritmo torna-se mais complexo porque aumenta o número de possibilidades de estratégia em execução
Uso de *semijoins*	Quando o principal componente de custo é a comunicação, um *semijoin* é útil para melhorar o processamento, pois reduz o tamanho dos dados a serem trocados pela rede. Porém, pode resultar num aumento de troca de mensagens e do tempo de processamento local

Camadas de Processamento de Consulta



- A entrada é uma consulta definida sobre relações globais (a distribuição fica transparente).
- Os 3 primeiros níveis mapeiam a query de entrada num plano de execução de query distribuída.
    - A decomposição da consulta e a localização dos dados correspondem à reescrita da query.
- As 3 primeiras atividades são executadas por um site de controlo central e usam informações sobre o esquema armazenado no diretório global.
- O 4º nível é responsável pela execução da query distribuída (o plano é executado e a resposta é retornada ao utilizador)
    - isto é feito pelos sites locais e pelo site de controlo

Decomposição de consultas

- Decomposição da query de cálculo algébrico numa consulta algébrica sobre relações globais
- As técnicas utilizadas por essa camada são as de um SGBD centralizado

Nomalização	A query de cálculo é reescrita de forma normalizada
Análise	A query normalizada é analisada semanticamente, de forma a que as queries incorretas sejam rejeitadas
Simplificação	As queries corretas são simplificadas, eliminando predicados redundantes
Restruturação	A query de cálculo é reestruturada para uma query algébrica

Localização dos Dados

- Localiza os dados e determina os fragmentos que estão envolvidos na query e transforma a query distribuída numa query sobre fragmentos

Entrada	Query algébrica sobre relações distribuídas
Objetivo	Localizar dados com o uso de informações de distribuição de dados
Resultado	Determina que fragmentos estão envolvido na query e transforma a query distribuída numa query sobre fragmentos

Otimização de Queries Globais

- Encontrar a melhor estratégia de execução possível. A estratégia consiste em ordenar as operações incluindo as de comunicação para minimizar o custo computacional

Entrada	Query algébrica sobre os fragmentos
Objetivo	- Encontrar a melhor estratégia de execução possível - A estratégia de execução pode ser descrita como operações de álgebra relacional e primitivas de comunicação para transferência de dados entre sites - A estratégia consiste em ordenar as operações incluindo as de comunicação, para minimizar o custo computacional
Resultado	Plano de execução de consulta distribuída

Execução Distribuída

- Otimização da query usando um repositório local, usando algoritmos de sistemas centralizados

Entrada	Operações de álgebra relacional
Objetivo	Otimização da query usando um repositório local
Resultado	A otimização local usa algoritmos de sistemas centralizados

Capítulo 5

Decomposição de consultas e Localização de dados

Decomposição de uma Query

A decomposição de uma query é realizada em 4 fases:

Nomalização	A query de cálculo é reescrita de forma normalizada
Análise	A query normalizada é analisada semanticamente, de forma a que as queries incorretas sejam rejeitadas
Simplificação	As queries corretas são simplificadas, eliminando predicados redundantes
Reestruturação	A query de cálculo é reestruturada para uma query algébrica

Normalização

Transformar a query numa forma normalizada
A parte mais importante é a transformação dos qualificadores (clausula WHERE)

Lógica clássica:

- Forma Normal Conjuntiva

$$(p_{11} \vee p_{12} \vee \cdots \vee p_{1n}) \wedge \cdots \wedge (p_{m1} \vee p_{m2} \vee \cdots \vee p_{mn})$$

- Forma Normal Disjuntiva

$$(p_{11} \wedge p_{12} \wedge \cdots \wedge p_{1n}) \vee \cdots \vee (p_{m1} \wedge p_{m2} \wedge \cdots \wedge p_{mn})$$

EXEMPLO

Encontre os nomes dos funcionários que trabalham no projeto P1 por 12 ou 24 meses

EMP(ENO, ENAME, TITLE)
PAY(TITLE, SAL)
PROJ(PNO, PNAME, BUDGET)
ASG(ENO, PNO, RESP, DUR)

```
SELECT ENAME
FROM   EMP, ASG
WHERE  EMP.ENO = ASG.ENO
AND    ASG.PNO = "P1"
AND    DUR = 12 OR DUR = 24
```

Forma Normal Conjuntiva

$$\text{EMP.ENO} = \text{ASG.ENO} \wedge \text{ASG.PNO} = \text{“P1”} \wedge (\text{DUR} = 12 \vee \text{DUR} = 24)$$

Forma Normal Disjuntiva (Pode ser redundante)

$$(\text{EMP.ENO} = \text{ASG.ENO} \wedge \text{ASG.PNO} = \text{“P1”} \wedge \text{DUR} = 12) \vee$$
$$(\text{EMP.ENO} = \text{ASG.ENO} \wedge \text{ASG.PNO} = \text{“P1”} \wedge \text{DUR} = 24)$$

Análise

Espécie de compilador que serve para:
- Detetar erros de sintaxe
- Detetar se uma consulta está correta semanticamente
- Detetar tipos incorretos
  - nomes de atributos e relações
  - tipos de valores/atributos em operações

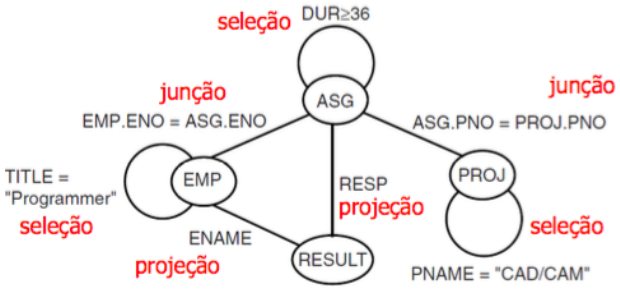
onde a consulta pode ser representada por meio de um grafo

EXEMPLO - SEMÂNTICA CORRETA

Encontre os nomes e as responsabilidades dos programadores que trabalham no projeto CAD/CAM por mais de três anos

EMP(ENO, ENAME, TITLE)
PAY(TITLE, SAL)
PROJ(PNO, PNAME, BUDGET)
ASG(ENO, PNO, RESP, DUR)

```
SELECT ENAME, RESP
FROM   EMP, ASG, PROJ
WHERE  EMP.ENO = ASG.ENO
AND    ASG.PNO = PROJ.PNO
AND    PNAME = "CAD/CAM"
AND    DUR ≥ 36
AND    TITLE = "Programmer"
```

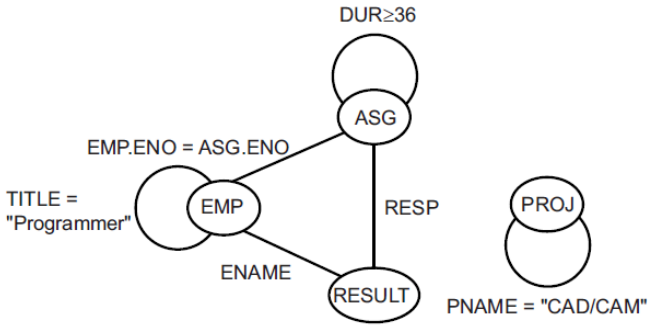


EXEMPLO - SEMÂNTICA INCORRETA

Uma query é semanticamente incorreta se o seu grafo da query não for conetado

EMP(ENO, ENAME, TITLE)
PAY(TITLE, SAL)
PROJ(PNO, PNAME, BUDGET)
ASG(ENO, PNO, RESP, DUR)

```
SELECT ENAME, RESP
FROM   EMP, ASG, PROJ
WHERE  EMP.ENO = ASG.ENO
AND    PNAME = "CAD/CAM"
AND    DUR ≥ 36
AND    TITLE = "Programmer"
```



Soluções:
- Rejeitar a consulta
- Supor um produto cartesiano implícito
  (ASG x PROJ)
- Deduzir o predicado ASG.PNO = PROJ.PNO

Simplificação / Eliminação de Redundância

Simplificação das consultas eliminando as redundâncias.

As redundâncias são muitas vezes devido a restrições de integridade semântica (forma normal conjuntiva ou disjuntiva)

Regras de Idempotência	Operadores Lógicos
$p \wedge p \Leftrightarrow p$	$p_1 \wedge p_2 \Leftrightarrow p_2 \wedge p_1$
$p \vee p \Leftrightarrow p$	$p_1 \vee p_2 \Leftrightarrow p_2 \vee p_1$
$p \wedge true \Leftrightarrow p$	$p_1 \wedge (p_2 \wedge p_3) \Leftrightarrow (p_1 \wedge p_2) \wedge p_3$
$p \vee false \Leftrightarrow p$	$p_1 \vee (p_2 \vee p_3) \Leftrightarrow (p_1 \vee p_2) \vee p_3$
$p \wedge false \Leftrightarrow false$	$p_1 \wedge (p_2 \vee p_3) \Leftrightarrow (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$
$p \vee true \Leftrightarrow true$	$p_1 \vee (p_2 \wedge p_3) \Leftrightarrow (p_1 \vee p_2) \wedge (p_1 \vee p_3)$
$p \wedge \neg p \Leftrightarrow false$	$\neg(p_1 \wedge p_2) \Leftrightarrow \neg p_1 \vee \neg p_2$
$p \vee \neg p \Leftrightarrow true$	$\neg(p_1 \vee p_2) \Leftrightarrow \neg p_1 \wedge \neg p_2$
$p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$	$\neg(\neg p) \Leftrightarrow p$
$p_1 \vee (p_1 \wedge p_2) \Leftrightarrow p_1$	

EXEMPLO

p1: title="Programmer",
p2: title="Elect. Eng."
p3: ename="J. Doe"

```
SELECT TITLE
FROM   EMP
WHERE  (NOT (TITLE = "Programmer")
AND    (TITLE = "Programmer"
OR     TITLE = "Elect. Eng.")
AND    NOT (TITLE = "Elect. Eng."))
OR     ENAME = "J. Doe"
```

$$(\neg p_1 \wedge (p_1 \vee p_2) \wedge \neg p_2) \vee p_3$$

$$(\neg p_1 \wedge \underline{(p_1 \vee p_2) \wedge \neg p_2}) \vee p_3$$

$$(\underline{\neg p_1 \wedge p_1} \wedge \neg p_2) \vee (\neg p_1 \wedge \underline{p_2 \wedge \neg p_2}) \vee p_3$$

$$(\underline{\neg p_1 \wedge ((p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_2))}) \vee p_3$$

$$(false \wedge \neg p_2) \vee (\neg p_1 \wedge false) \vee p_3$$

$$false \vee false \vee p_3$$

p3: ename="J. Doe"

```
SELECT TITLE
FROM   EMP
WHERE  ENAME = "J. Doe"
```

Reestrutução / Reescrita

- **Etapas:**
  - Transformação direta da consulta de cálculo relacional para álgebra relacional
  - Reestruturação da consulta algébrica para melhorar o desempenho

A consulta em álgebra relacional é representada por uma árvore de operadores

- **Árvore de Operadores:**

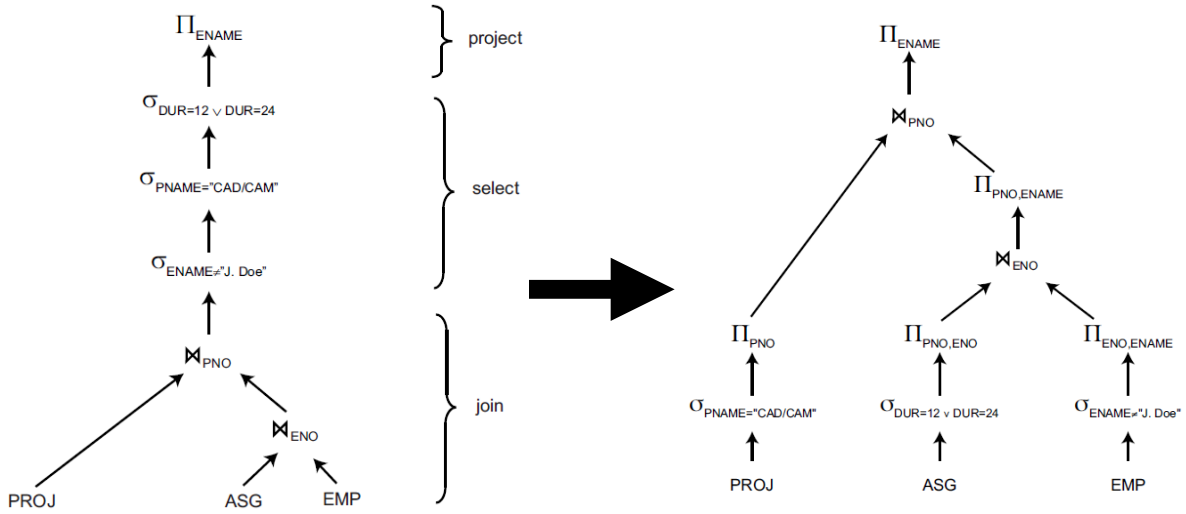
- Folhas = Relações
- Não folhas = Relações intermédias
- Sentido: folhas até à raíz

1. Cada relação é uma folha
   - FROM
2. Raiz é a operação de projeção
   - SELECT
3. Qualificações são sequências das folhas até a raiz (operações relacionais)
   - WHERE

EXEMPLO

Selecionar todos empregados que não são "J.Doe" que trabalham no projeto CAD/CAM por 1 ou 2 anos

```
SELECT ENAME
FROM   PROJ, ASG, EMP
WHERE  ASG.ENO = EMP.ENO
AND    ASG.PNO = PROJ.PNO
AND    ENAME != "J. Doe"
AND    PROJ.PNAME = "CAD/CAM"
AND    (DUR = 12 OR DUR = 24)
```



Árvore Equivalentes:

- Fase de otimização: comparação de todas as árvores possíveis
- # árvores possíveis —> ∞
- A ideia consiste em aplicar as regras de transformação para que as “más árvores” sejam eliminadas
- Vantagens
    1. Separação de operações unárias (simplifica a consulta)
    2. Agrupamento de operações unárias (acesso à relação de uma vez)
    3. Comutação (permutação) de operações unárias e binárias (operações de seleção podem ser executadas primeiro)
    4. Ordenação das operações binárias (usada na otimização de consultas)

Estrutura de Base de Dados Distribuídas

- Localização de dados fragmentados entre as unidades de armazenamento do SGBDD
- Converter uma consulta algébrica sobre relações globais numa consulta algébrica expressa sobre fragmentos físicos

- Programa de localização
    - É um programa em álgebra relacional cujos operandos são os fragmentos
    - Obtido a partir das regras de reconstrução das consultas globais

- Cada relação global é substituída pelo seu programa de localização
- A consulta resultante é chamada de “consulta localizada”
- A “consulta localizada” ainda pode ser simplificada e reestruturada por meio da aplicação de **regras de redução.**

- **Regras de Redução:**
    - **Redução da Fragmentação Horizontal**
        - A função de fragmentação horizontal faz a distribuição duma relação com base nos predicados de seleção (o operador de reconstrução é a união)
        - As técnicas de redução para a fragmentação horizontal procuram determinar quais sub-árvores produzirão relações vazias para que sejam removidas

    - **Redução com seleção (select)**
        Dada uma relação R que foi fragmentada horizontalmente como R_1, R_2, ..., R_w, onde R_j = $sel_{pj}(R)$, a regra pode ser enunciada formalmente como:

        $$\textbf{Rule 1: } \sigma_{p_i}(R_j) = \phi \text{ if } \forall x \text{ in } R : \neg(p_i(x) \wedge p_j(x))$$

        onde p_i e p_j são predicados de seleção, x denota um tuplo e p(x) denota “predicado p(x) válido para x”

        ➡ Um predicado da consulta é aplicado a um fragmento e retorna um conjunto vazio!

EXEMPLO

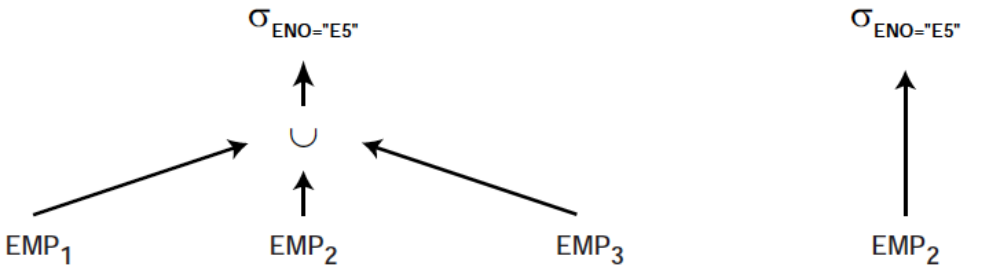
EMP(ENO, ENAME, TITLE)

$EMP_1 = \sigma_{ENO \leq "E3"}(EMP)$
$EMP_2 = \sigma_{"E3" < ENO \leq "E6"}(EMP)$
$EMP_3 = \sigma_{ENO > "E6"}(EMP)$

→ Fragmentação Horizontal

$EMP = EMP_1 \cup EMP_2 \cup EMP_3$ → Programa de Localização (a evitar)

Considere a consulta:



(a) Localized query (b) Reduced query

➡Os predicados de EMP1 e EMP3 são contraditórios com o predicado da seleção da consulta, logo os fragmentados podem ser eliminados

- **Redução com junção (join)**
  - Distribuir junções sobre uniões e eliminar junções inúteis
  - Primeiro, as uniões devem ser movidas para cima da árvore para que as junções apareçam
  - A distribuição da junção sobre a união pode ser expressa como:

$$(R_1 \cup R_2) \bowtie S = (R_1 \bowtie S) \cup (R_2 \bowtie S)$$

  - onde Ri são fragmentos de R, e S é uma relação
  - Segundo, as junções inúteis devem ser removidas
  - Supondo que os R_i e R_j sejam definidos respetivamente de acordo com os predicados p_i e p_j sobre o mesmo atributo, a regra de simplificação pode ser expressa da seguinte maneira:

Rule 2: $R_i \bowtie R_j = \phi$ if $\forall x$ in $R_i, \forall y$ in $R_j : \neg(p_i(x) \wedge p_j(y))$

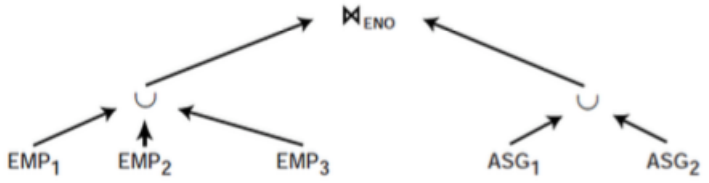
EXEMPLO

```
SELECT *
FROM   EMP, ASG
WHERE  EMP.ENO = ASG.ENO
```

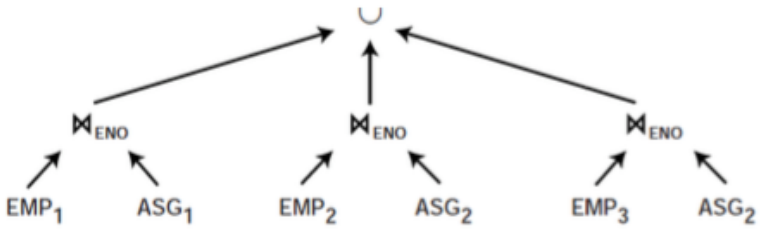
$ASG_1 = \sigma_{ENO \leq "E3"}(ASG)$
$ASG_2 = \sigma_{ENO > "E3"}(ASG)$

→ ASG: Funcionários e os projetos onde eles trabalham

$EMP_1 = \sigma_{ENO \leq "E3"}(EMP)$
$EMP_2 = \sigma_{"E3" < ENO \leq "E6"}(EMP)$
$EMP_3 = \sigma_{ENO > "E6"}(EMP)$



(A) LOCALIZED QUERY



(B) REDUCED QUERY

➡ As junções inúteis são eliminadas e as que ficam podem ser implementadas em paralelo

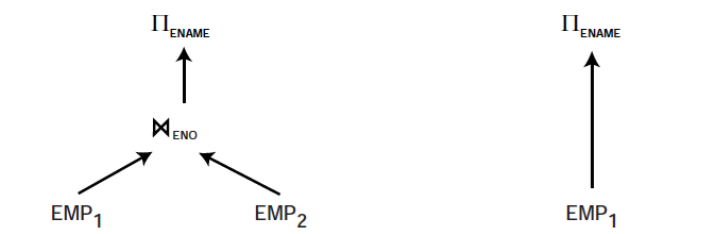
- **Redução da Fragmentação Vertical**
  - O programa de localização para uma relação fragmentada verticalmente consiste na junção dos fragmentos sobre o atributo comum
  - Consultas podem ser reduzidas pela determinação das relações intermédias inúteis e pela remoção das sub-árvores que as produzem
  - Dada uma relação R definida sobre atributos A = {A1, ..., An}, fragmentada verticalmente como Ri = πA'(R), onde A' está contido em A, a regra pode ser expressa como:

Rule 3: $\Pi_{D,K}(R_i)$ is useless if the set of projection attributes D is not in A'.

EXEMPLO

```
SELECT ENAME
FROM   EMP
```

$$EMP_1 = \Pi_{ENO,ENAME}(EMP)$$
$$EMP_2 = \Pi_{ENO,TITLE}(EMP)$$



(A) LOCALIZED QUERY (B) REDUCED QUERY

➡ O fragmento EMP2 não tem ENAME!

- **Redução para Fragmentação Derivada**
  - A operação de junção pode ser otimizada quando as relações foram fragmentadas usando o mesmo predicado de seleção
  - A junção de duas relações pode ser implementada como a união das junções parciais (as junções parciais podem ser executadas em paralelo)
  - Só vale quando os predicados usados na fragmentação são os mesmos!

Fragmentação derivada

- Uma maneira de distribuir duas relações para que o processamento conjunto da seleção e junção seja otimizado
- Os predicados não são os mesmos!
- Consultas definidas sobre fragmentos derivados podem ser reduzidos
- Este tipo de fragmentação é feita para otimizar junções
- Uma regra útil consiste em:
  1. Distribuir as junções sobre as uniões (Substituir junções por uniões de junções parciais)
  2. Aplicar a regra 2 para redução de fragmentação horizontal

EXEMPLO

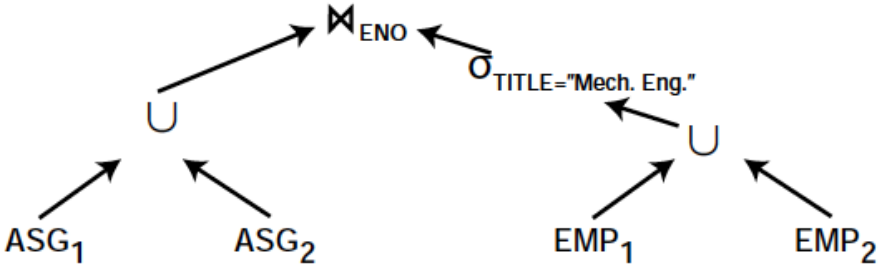
$ASG(ENO, PNO, RESP, DUR)$ → Informações sobre os projetos onde os empregados estão alocados

$EMP_1 = \sigma_{TITLE="Programmer"}(EMP)$
$EMP_2 = \sigma_{TITLE\neq"Programmer"}(EMP)$ → Fragmentação derivada

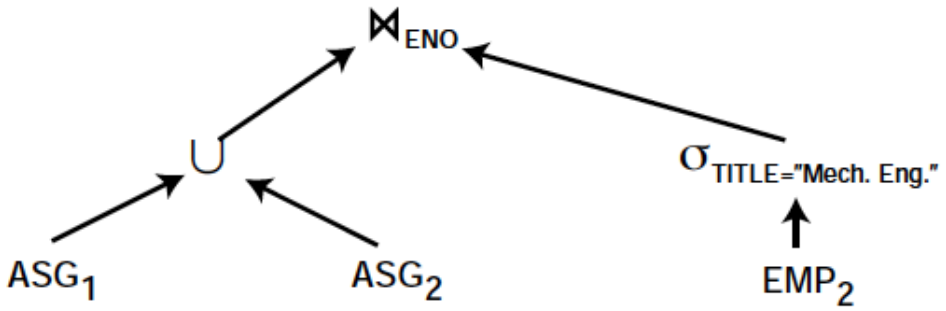
$$ASG_1 = ASG \ltimes_{ENO} EMP_1$$
$$ASG_2 = ASG \ltimes_{ENO} EMP_2$$

$ASG = ASG_1 \cup ASG_2$ → Programa de localização

```
SELECT *
FROM   EMP, ASG
WHERE  ASG.ENO = EMP.ENO
AND    TITLE = "Mech. Eng."
```

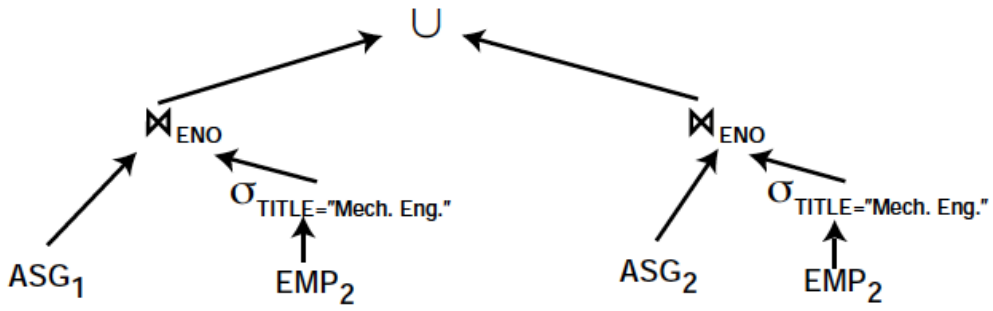


(A) LOCALIZED QUERY



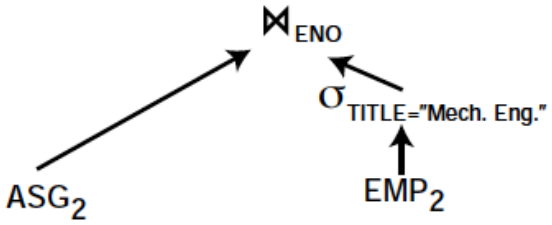
(B) QUERY AFTER PUSHING SELECTION DOWN

➡ EMP1 é removido porque só tem programadores



(c) QUERY AFTER MOVING UNIONS UP

➡A sub-árvore da esquerda é eliminada porque os predicados dos fragmentos são contraditórios



(D) REDUCED QUERY AFTER ELIMINATING THE LEFT SUBTREE

-**Redução para Fragmentação Híbrida**

$EMP_1 = \sigma_{ENO \leq "E4"}(\Pi_{ENO,ENAME}(EMP))$
$EMP_2 = \sigma_{ENO > "E4"}(\Pi_{ENO,ENAME}(EMP))$
$EMP_3 = \Pi_{ENO,TITLE}(EMP)$

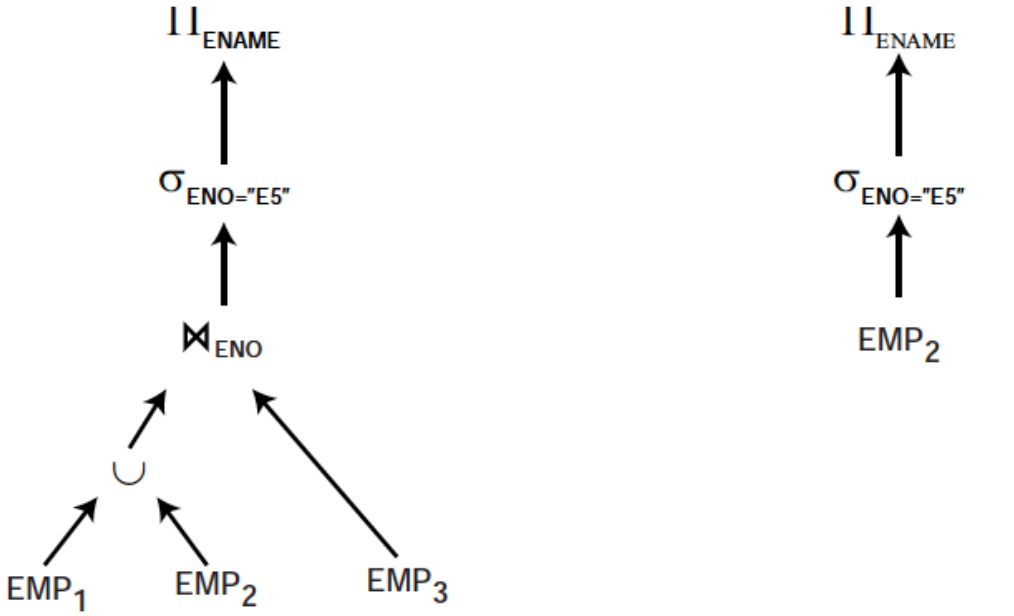
➡ Fragmentação híbrida

$EMP = (EMP_1 \cup EMP_2) \bowtie_{ENO} EMP_3$

➡ Programa de localização

EXEMPLO

```
SELECT ENAME
FROM   EMP
WHERE  ENO="E5"
```



(A) LOCALIZED QUERY (B) REDUCED QUERY

➡ O fragmento EMP1 é eliminado por causa da seleção
➡ O fragmento EMP3 é eliminado por causa da projeção

Conclusão

- Decomposição de consultas e localização de dados são duas funções sucessivas
- Muitas consultas algébricas podem ser equivalentes à mesma consulta de entrada
- Abordagens ingénuas são ineficientes
- Reestruturação com a utilização de regras e regras de redução
- Mais otimizações são geradas nas camadas subsequentes, considerando informações sobre o ambiente de processamento

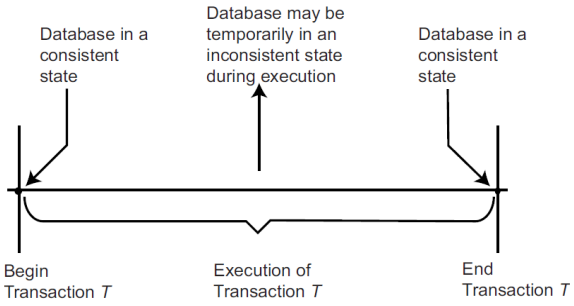
Capítulo 6

Transações

Transação (T) é uma execução de um programa de utilizador visto pelo SGBD como uma série de operações de leitura e escrita.

- É um programa em execução ou processo que inclui um ou mais acessos à base de dados que efetuam leituras ou atualizações dos seus registos
- É uma unidade atómica de trabalho que estará completa ou não realizada

➡ Uma transação é um conjunto de ações que transformam a BD de um estado consistente para outro, durante a execução a BD pode ser inconsistente.



Lista de Ações de uma Transação:

- Leitura
- Escrita
- Especificação da ação final:
  - Commit
  - Abort

Estados de uma Transação

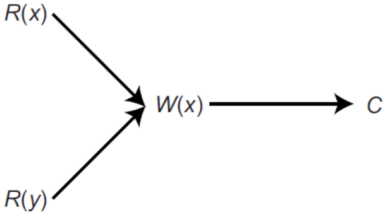
Ativa	Durante a execução
Parcialmente submetida	Depois da declaração final ter sido submetida
Falhada	Depois de descobrir que a execução não pode mais proceguir
Abortada	Depois de fazer *rollback* e a BD é restaurada ao seu estado inicial
Submetida	Transação completa com sucesso

Formalizando

Read(x)
Read(y)
$x \leftarrow x + y$
Write(x)
Commit

$\Sigma = \{R(x), R(y), W(x), C\}$
$\prec = \{(R(x), W(x)), (R(y), W(x)), (W(x), C), (R(x), C), (R(y), C)\}$

$\sum$ - domínio
$<$ - ordem



Propriedades das Transações

Acrónimo ACID

- **Atomicidade**
  - A propriedade da atomicidade garante que as transações sejam atómicas (indivisíveis)
  - A transação será executada totalmente ou não será executada. A atomicidade exige que, se a execução de uma transação for interrompida por qualquer falha, o SGBD seja responsável pela determinação do que fazer com a transação após a recuperação da falha.
    - **Tipos de falha:**
      - Cancelada ou terminada sem sucesso devido a alguma anomalia durante a execução
      - Fonte de alimentação interrompida
      - Incapacidade de aceder a um dado no disco
  - Um SGBD garante a atomicidade da transação desfazendo as ações das transações incompletas.

- **Consistência**
  - A propriedade de consistência garante que a BD passará de uma forma consistente para outra forma consistente.
    - **Níveis de Consistência:**
      - Grau 0: Transações não substituem "dirty data" de outras transações
      - Grau 1: Grau 0+T não submete nenhum *write* antes do fim da transação (EOT)
      - Grau 2: Grau 1+T não lê "dirty data" de outras transações
      - Gray 3: Grau 2+T qualquer T que não "suje" a informação lida pela transação antes da transação estar completa
  - dirty data: dados atualizados pelas transações mas ainda não submetidos
  - A transação preserva a integridade da BD.

- **Isolamento**
  - A propriedade de isolamento garante que a transação não será interferida por nenhuma outra transação concorrente.
  - Uma transação em andamento não pode revelar os seus resultados a outras transações concorrentes antes de se consolidar (*commited*).
    - **Situações / Fenómenos:**
      - dirty data: T_1 modifica x que depois é lida por T_2 antes de T_1 terminar; se T_1 abortar, T_2 leu um valor que nunca existiu na BD
      - leitura não repetida: T_1 lê x; T_2 modifica ou apaga x e faz commit; T_1 tenta ler x novamente, mas lê um valor diferente ou não o consegue encontrar
      - fantasma: T_1 procura na BD de acordo com um predicado P enquanto T_2 insere tuplos novos que satisfaçam P. Quando a meio de uma transação existe outra transação que insere novos dados que satisfaçam a primeira.

    - **Níveis de Isolamento:**
      - read uncommited: os três fenómenos (dirty data, leitura não repetida, fantasma) acontecem.
      - read commited: acontece o fenómeno de leitura não repetida e fantasma
      - repeat and read: acontece o fenómeno fantasma
      - anomaly serializable: nenhum dos fenómenos acontece

- **Durabilidade**
  - A propriedade da durabilidade garante que o que foi salvo, não será perdido
  - Propriedade que assegura que, uma vez que a transação se consolida (*commited*), os seus resultados tornam-se permanentes e não podem ser apagados da BD.

Plano de Execução

- É uma lista de de ações (read, write, abort, commit) de um conjunto de transações.
- A ordem na qual 2 opções de uma transação T aparecem num plano de execução deve ser a mesma em que elas aparecem em T.

• **Tipos de Planos de Exeução**

Completo	Contém um aborto ou commit para cada transação, cujas as ações estão listadas nele.
Serial	Transações são executadas no inicio ao fim, uma por uma.

Tipos de Transações

Em relação à duração	Em relação à organização da leitura e escrita
• Online ou Curtas: - Tempos de execução/resposta muito curtos e pelo acesso a uma porção relativamente pequena da BD • Lotes ou Longos: - Tempos de execução/resposta longos e acedem a uma grande porção da BD	• Geral: - Lê em qualquer ordem • Restrita: - 2 passos: Faz todos os *reads* e depois os *writes* - lê antes de escrever: lê antes da escrita - modelo ação: faz por pares <*read*, *write*> executados atomicamente

Estrutura das Transações

- Flat Transaction: Apenas um bloco *Begin … End*
- Nested Transaction:
  - Open: Sucesso dos *commits* interiore depende do sucesso dos *commits* exteriores
  - Close: *Commits* independentes em cada transação

Arquiteturas das Transações

O monitor de execução distribuída consiste em:

Coordenador de Transações (TM)	É responsável pela coordenação da execução das operações na BD em nome da aplicação
Planeador (SC)	É responsável pela implementação de um algoritmo especifico de controlo de concorrência para sincronizar o acesso à BD
Coordenador de Recuperação Local (LRM)	Existe em cada site e é responsável por implementar os procedimentos locais pelos quais a BD pode ser levada a um estado consistente depois de uma falha

O Coordenador de Transações implementa uma interface para os programas da aplicação que consiste em 5 comandos:

Begin_Transaction	Indica ao TM que uma nova transação está a começar. O TM regista o seu nome, a aplicação originária…
Read	Se os dados estão armazenados localmente, o valor é lido e retornado à transação. Caso contrário, o TM encontra onde os dados estão e solicita o seu valor
Write	Se os dados estão armazenados localmente, o seu valor é atualizado. Caso contrário, o TM encontra onde estão os dados e solicita a atualização desse valor
Commit	O TM coordena os sites envolvidos, a fazer atualização dos dados em nome da transação para que a atualização seja permanente em todos os sites
Abort	O TM assegura-se de que os efeitos da transação não são refletidos em nenhuma BD dos sites onde os dados foram atualizados

Capítulo 7

Controlo Distribuído da Concorrência

- Problema de sincronização de transações simultâneas de tal forma que as 2 propriedades seguintes sejam atingidos:
  - A consistencia da BD é mantida
  - O grau máximo de concorrência é atingido
- O controlo da concorrência lida com os propriedades de transações: isolamento, consistência
- A execução das transações em série atinge a consistência mas diminui a performance
- Objetivo: equilíbrio entre consistência e concorrência.

Serialização

- Escalonamento: ordem cronológica pela qual as instruções de várias transações são executadas

- A serialização de escalonamento é usado para identificar quais escalonamentos estão corretos quando as execuções da transição tiverem intercalação das suas operações nos escalonamentos.
- A função primária de um controlador de concorrência é gerar um escalonamento serializável para a execução de transações pendentes.
- Se num escalonamento S as operações não estão intercaladas (ou seja, as operações de cada transação ocorrem consecutivamente) dizemos que o escalonamento é serial.
- A execução serial de um conjunto de transações mantém a consistência da base de dados, porém pode acarretar estados de inatividade da CPU, desperdiçando processamento.
- A execução concorrente de transações deve deixar a base de dados num estado que possa ser alcançado por uma execução sequencial em alguma ordem.
- Caso essa situação seja alcançada serão resolvidos problemas como os de atualizações perdidas.
- Assegurando o isolamento e não permitindo que os resultados incompletos sejam acedidos por outras transações.
- Um escalonamento S (ou Schedule, também chamado de Histórico) é definido sobre um conjunto de transações T={T1, T2, …, Tn} e especifica uma ordem intercalada de execução dessas operações de transações.
- Um escalonamento pode ser especificado como uma ordem parcial sobre T.

• **Tipos de Escalonamento em SGBDDs:**
  - Escalonamento local - Histórico da execução da transação em cada site
  - Escalonamento global - Se a BD não é replicada e cada histórico local é serializável, a sua união (histórico global) também é se a ordem da serialização local é idêntica (união da BD não replicada e histórico local serializável)

EXEMPLO

$$T_1: \quad Read(x) \qquad\qquad T_2: \quad Read(x)$$
$$x \leftarrow x + 5 \qquad\qquad x \leftarrow x * 10$$
$$Write(x) \qquad\qquad Write(x)$$

Schedule:

* Site1: $S_1 = \{R_1(x), W_1(x), R_2(x), W_2(x)\}$
* Site2: $S_2 = \{R_2(x), W_2(x), R_1(x), W_1(x)\}$

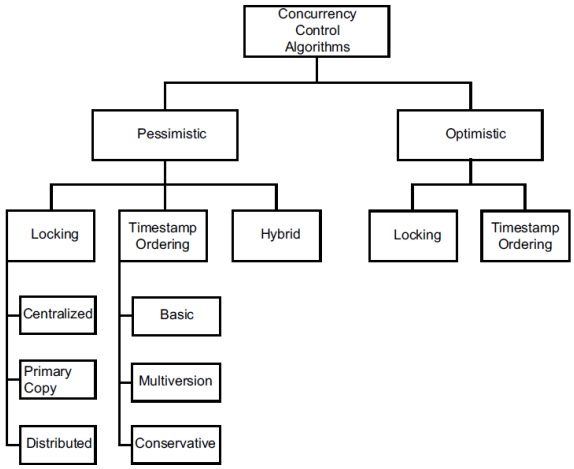
- São serializáveis localmente.
- No entanto não são globalmente pois violam consistência tendo em conta que produzem valores diferentes.

Taxonomia da Concorrência

Modos de abordagem de controlo de concorrência:
- Distribuição de BD - total ou parcialmente replicada
- Topologia da rede - “star”, circular ou broadcasting
- Primitiva de sincronização - timestamp ordering ou locking-based

Podem ser usados em algoritmos com 2 pontos de vista:
- Pessimista: muitos transações entrarão em conflito, portanto a sincronização da execução concorrente ocorre mais cedo no seu ciclo de execução.
- Otimista: poucas transações em conflito portanto a sincronização de execução concorrente ocorre até ao fim

Algoritmos de controlo de concorrência



➡ Locking-Based

- Controlo de concorrência baseado em bloqueio, que assegura que os dados compartilhados por operações conflitantes sejam acedidas por uma única operação de cada vez.
- É conseguido pela associação de um “bloqueio” a cada unidade de bloqueio
- Esse bloqueio é definido por uma transação antes de ser acedida e é redefinida no final do seu uso.
- Modos de Bloqueio: Bloqueio de Leitura e Bloqueio de Escrita

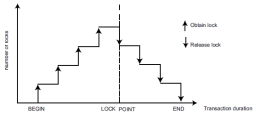
➡ Locking 2 fases (2PL)

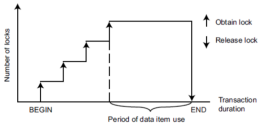
- A regra de bloqueio de 2 fases estabelece que nenhuma transação deve solicitar um bloqueio após libertar um dos seus bloqueios.
- Como alternativa: uma transação não deve libertar um bloqueio até ter a certeza de que solicitará outro bloqueio

As 2 fases são:
- Fase de Crescimento: Obtém bloqueios e acede a itens de dados
- Fase de Contração: Liberta os bloqueios

Qualquer escalonamento / histórico gerado por um algoritmo que obedece à regra 2PL é serializável.

Existem 2 tipos de 2PL:
- Cascading Aborts: Se uma T aborta depois de libertar alguns bloqueios pode causar outra T a abortar
- Strict 2PL: Todos os bloqueios são libertados simultaneamente para evitar abortos em cascata





➡ 2PL Centralizado

- Um único site é responsável pelo coordenador de bloqueios (LM). Os bloqueios são solicitados pelo TM ao LM.

Vantagens	Desvantagens
- Fácil de implementar	- Provoca "engarrafamento" - Pouca Fiabilidade

➡ 2PL Distribuído

- O coordenador de bloqueio é distribuído pelos sites todos. O LM de cada site é responsável pelo bloqueio dos seus dados.
- As mensagens são enviadas para todos os LMs participantes

Desvantagens	
- Difícil gestão de *deadlocks* - Custos de comunicação elevados	Quando 2 ou mais operações ficam bloqueadas, esperando uma pela outra

➡ 2PL Cópia-Primária

- Extensão direta do 2PL centralizado
- Distribui LM em vários sites e cada um irá administrar um dado conjunto de unidades de bloqueio

➡ Timestamp Ordering (TO)

- Os algoritmos de controlo de concorrência do tipo timestamp definem uma ordem de serialização para cada transação e executam-nas de acordo com ela.
- Para estabelecer essa ordem, o TM atribui a cada transação um timestamp
- Transações são individualmente serializados e executados
- O TM atribui um TS único a cada T
- TS são únicos localmente e globalmente e crescem monotonamente

• TO regra:

Dadas 2 operações O1 e O2 respetivas às T1 e T2:

- T1 é executada antes de T2 apenas se o timestramp de T1 for menor que o timestamp de T2

$$ts(T1) < ts(T2)$$

- Neste caso diz-se que T1 é a transação mais velha e a T2 a transação mais nova.

• 2 Tipos de Timestamp:

- read (rts)
- write (wts)

➡ TO Básico

- O TM atribui um timestamp a cada T
- T são executadas imediatamente
- Scheduler compara a TS do item de dados com o TS de operação
- Se o TS do item entrar em conflito com o TS de operação então T é abortada
- Não existe bloqueios - T são abortados e reiniciados em caso de conflito (degradação da performance)
- Como as transações nunca esperam enquanto mantêm direitos de acessos aos dados o algoritmo básico de TO nunca provoca impasses. No entanto, o preço para se livrar de impasses é a reiniciação potencial de uma transação várias vezes.

➡ TO Conservativo

- Usado para minimizar o número de reiniciações de TS
- S atrasa cada operação até não existir uma operação com o TS menor (mais velhas)

(S atrasa (O) até não existirem (O) mais velhas)

- Se isto for assegurado, não se rejeita mais TS

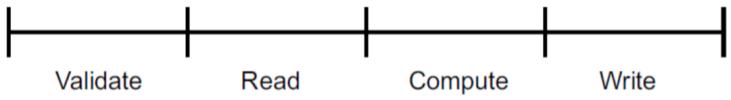
Cada S:

- Tem uma fila para cada TM
- Armazena informação de ordem ascendente (e executa)
- Mesmo assim a reiniciação pode ocorrer

➡ TO Multiversão

- Os writes de operações não modificam os itens, apenas criam uma nova versão de item de dados
- Os reads são sempre bem sucedidos no seu TS (T_i)
- Writes são rejeitados se uma T que tem um TS maior já leu o item de dados
- Pode ser: Otimista ou Pessimista

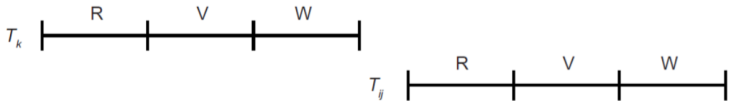
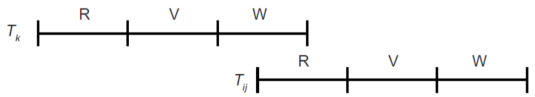
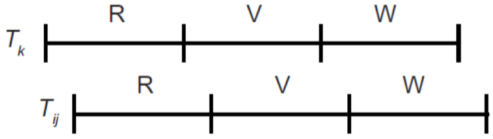
Pessimista:



Otimista:

• Políticas de Timestamp:

- TS na transação e não nos itens de dados (no rts e wts)
- Quando a primeira subtransação atinge a fase de validação
  - O mesmo TS é definida a todas as subtransações

Regra 1	Se toda a T_k onde $TS(T_k) < TS(T_{ij})$ completou a fase *write* antes de T_{ij} começar a fase *read* a validação é bem sucedida 
Regra 2	Se uma T_k em que $TS(T_k) < TS(T_{ij})$ e T_k completa a sua fase de *write* enquanto de T_{ij} está na fase de *read*, a validação é bem sucedida se: $$WS(T_k) \cap RS(T_{ij}) = \emptyset.$$ 
Regra 3	Se existe um T_k que $TS(T_k) < TS(T_{ij})$ e que T_k completa a sua fase de *read* antes de T_{ij} completar a sua fase de read, a validação é bem sucedida se: $$WS(T_k) \cap RS(T_{ij}) = \emptyset, \text{ and } WS(T_k) \cap WS(T_{ij}) = \emptyset.$$  Atualizações de T_k não afetam a fase de *read* ou *write* de T_{ij}