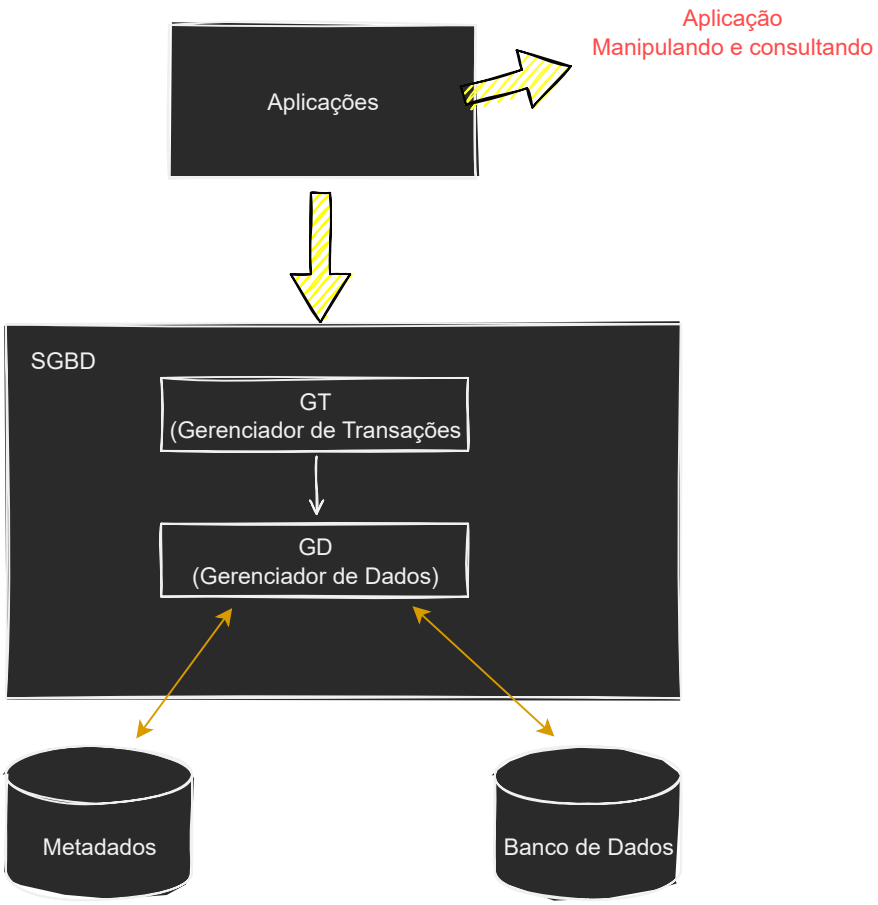
# Banco de Dados 1.1

## Arquivos de dados:

Vamos lembrar de uma representação simplificada de Banco de dados:

- Aplicações: programas para manipular e consultar os dados
- GT (Gerenciados de Transações): é responsável pelo processamento sobre os dados
- GD (Gerenciador de Dados): responsável pelo acesso aos dados
- Metadados: informações sobre os dados
- BD (Banco de dados): conjunto de dados armazenados
- SGBD (Sistema Gerenciados de Banco de Dados): programa responsável pelo gerenciamento do banco de dados

Em formato de diagrama, vamos ter:




Antes de entrar no contexto sobre Banco de Dados, precisamos entender como antigamente era utilizado para o armazenamento de dados. Dessa forma, temos então dois conceitos

importantes: Registro e Arquivo

- Registro:

  > O Registro é um conjunto de campos ou unidades, do qual cada campo ou unidade corresponde a um atributo do registro. É melhor compreendida como uma Estrutura de Dados Composta Heterogênea, ou de outro modo, como uma Struct em C ou uma classe em Java.

- Arquivo:

  > Conjunto de registros armazenados em um dispositivo de memória.
  > Esse Arquivo é dito como um armazenamento não volátil, ou seja, é um armazenamento persistente. A organização dos registros dentro de um arquivo é variada, do qual busca distribuir as unidades de informações de maneira mais eficiente ao seu uso.
  > O Arquivo pode ser melhor compreendido como um Recurso Computacional de Armazenamento Secundário de dados, permitindo alocação de espaço.

Dessa forma, possuímos os meios físicos de armazenamento ordenados do mais rápido e caro até o mais lento e mais barato:

- Cache (Primária e armazenamento volátil em relação a falta de energia)
- Memória Principal (Primária e armazenamento volátil em relação a falta de energia)
- Memória Flash (Secundária)
- Disco Magnético (Secundária e o disco fornece o maior volume de memória secundária)
- Disco Óptico (Terciária - Offline)
- Fita Magnética (Terciária, comumente usadas para backup e armazenamento de dados. Tem muita capacidade de armazenamento e podem ser removidas da unidade de fita)

OBS: Alguns aspectos são importantes para decidir qual será o tipo de armazenamento para guardar os dados manipulador por sistemas computacionais. Os aspectos são:

- Capacidade de Armazenamento
- Velocidade de Operação entre os dados armazenados
- Custo ou valor financeiro para aquisição e manutenção

Temos diferentes técnicas para armazenar os dados em Arquivos. Segue abaixo algumas delas:

## Sequencial

- Possui uma chave primária, chamada de chave de ordenação, do qual os registros obedecem essa sequência ordenadamente. O acesso é feito sequencialmente.
- Tem vantagem na questão do armazenamento de registro aleatórios no arquivo
- Sua desvantagem está relacionada a perda de flexibilidade nas operações de modificações dos dados no arquivo. Para o acesso ao registro é preciso de argumentos que coincidam com a chave de ordenação ou em atualizações por lotes (arquivos batch).

## Sequencial Indexado:

- Para ter o acesso aleatório ao grande volume de dados, essa técnica utiliza de uma estrutura de acesso associada ao arquivo de dados (índice) para que torne eficiente o acesso.
- Esse índice é formado por uma coleção de pares entre o valor da chave de acesso e o endereço físico no arquivo, do qual é sempre específico a uma chave de acesso.
- Possui áreas de extensão para a inserção de novos registros.
- Os índices agilizam na consulta e ainda permanecem em memória, entretanto precisa-se de áreas de extensão que estão sempre sendo necessitadas por organização.

## Direto:

- Não necessita de uma estrutura auxiliar como a técnica de Sequencial Indexado, e utiliza-se da instalação de registro em endereços determinado de acordo com o valor da chave primária aonde possa ter acesso rápido aos mesmos apenas especificando os argumentos de pesquisa.
- Obtém-se a eficiência no acesso aleatório ao utilizar a chave do próprio registro ou uma função para calcular o endereço do registro a partir do argumento. Chamamos isso de Hash. O mesmo é indicado para aplicações com atualizações arbitrárias.
- Tem menos espaço por não precisar de mais de um arquivo, entretanto precisa determinar funções que irão gerar o menor número de colisões.
- Para melhor entendimento, um campo do registro (atributos) é escolhido como uma Chave Primária, e a mesma possui a responsabilidade de identificar um único registro no arquivo.

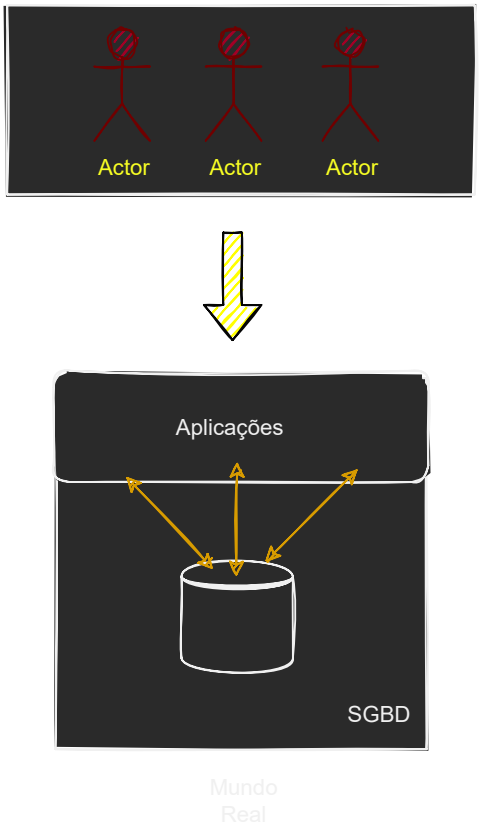
# SGBD (Sistema Gerenciador de Banco de Dados Relacional):

A definição geral de Banco de Dados é a ideia de coleção ou um conjunto de dados que servem ou são usados por algumas situações específicas. Vale lembrar que um arranjo aleatório de dados não pode ser considerado um banco de dados.

Lembre-se que a expressão Banco de Dados se refere ao SGBD, ou seja, a marca, tecnologia ou a empresa proprietária pelo SGDB. Outra expressão é Base de dados, do qual nos

referimos a nossa organização estrutural implementada em nosso Banco de Dados.

O Banco de Dados envolve tipos de dados e estruturas para serem gravador no BD. A sua construção é o processo de gravar inicialmente esses dados, e a manipulação é sobre incluir funções com o propósito de consultar e atualizar os dados específicos.




- Um banco de dados não armazena somente os dados, mas também as definições e descrições das estruturas que forma o BD (metadados). O mesmo contém definições da estrutura de cada arquivo, tipo de dados e formato de armazenamento de cada item.
- Esse dicionário é usado pelo SGBD e por algum usuário do banco de dados.

O maior objetivo em um Banco de Dados é oferecer os dados com uma abstração para que a disponibilidade dos dados se torne eficiente. Os detalhes sobre como os dados estão mantidos e armazenados não interessa ao usuário final.

O conceito de abstração se baseia em:

- se caracteriza apenas na questão de observar somente os aspectos de interesse, sem precisar se preocupar com maiores detalhes envolvidos
- na abstração de dados, o banco de dados pode ser visto sem se preocupar com a forma em que está implementado e armazenado fisicamente.

- o banco de dados representar algum aspecto do mundo real e a alteração nesse mundo tem que ser refletida no banco de dados.

Quando falamos em abstração, precisamos também dizer sobre o que seria um Modelo de Dados, logo que é a principal ferramenta de informações sobre a abstração do mundo real:

- Consiste de um conjunto de conceitos utilizados para descrever a estrutura de um banco de dados, sendo eles tipos de dados, relacionamentos e restrições sobre estes dados.
- Elaborar um Modelo de Dados é o mesmo que fazer uma Modelagem dos Dados
- O Modelo de Dados pode ser dividido entre:
  - Conceitual (Alto nível | Externo), aonde fornece os conceitos próximos da percepção lógica dos usuários acerca dos dados
  - Implementação (Nível Lógico), do qual utiliza SGBDs comerciais, e o mais popular é o Modelo Relacional (MR)
  - Física (Baixo nível | Nível Interno) que descreve como os dados são armazenado (fisicamente) de fato.

Quando falamos sobre Banco de Dados, devemos saber diferenciar o que é Dados e Informações. Os dados são aqueles que denota um fato e que pode ser registrado, possuindo um significado implícito (nome e endereço de pessoas). A informação é aquela que se trata de uma organização relacionado ao conteúdo ou uma novidade, um exemplo é uma tabela com nome dos amigos, telefone e idade.

Um banco de dados possui algumas propriedades que são implícitas a ele, são elas:

- É uma coleção logicamente coerente de dados com um significa inerente
- É projetado e construído com dados para um propósito específico
- Possui um grupo de usuário e aplicação pré-concebidas do qual esses usuários estão interessados
- Um banco de dados sempre representa alguns aspecto do mundo real e a alteração nesse mundo tem que ser refletida no BD.

A arquitetura de um Banco de Dados pode ser dividida em três níveis:

- Nível Externo (Visão):

  > Aqui descreve o nosso banco de dados com estruturas mais simples entretanto ainda apresenta alguma complexidade devido ao tamanho do banco. Pode ser comparada o Nível Conceitual

- Nível Lógico (Conceitual):

> Nesse caso descrevemos como os dados estão armazenados de fato e quais são as suas relações. É totalmente descrito em termos de estruturas relativamente simples.

- Nível Interno (Físico):

> Descreve como os dados estão realmente armazenados, e onde as estruturas complexas do banco de dados são descritas em detalhes.

Em um banco de dados grande, aonde possui milhares de usuário e com uma restrição de acesso, possui perfis diferentes de pessoas que interagem com o Banco de Dados:

- Administrador de Banco de Dados (DBA)

> Supervisor do banco de dados e responsável pela autorização de acesso ao banco, monitoramento e coordenação de seu uso. (Aspectos Físicos)

- Projetista do Banco de Dados

> Responsável pela identificação dos dados e elaboração de estruturas apropriadas para armazená-los. (Precisa compreender os requisitos necessários antes de implementar o banco)

- Analista de Sistemas

> É o responsável em definir os requisitos dos sistemas para os usuários e desenvolver especificações que atendam esses requisitos

- Programador de Aplicações

> Implementa as especificações no formato de Aplicações, elabora as documentações da implementação

- Usuário (final / mais importante)

> BD existe por conta do usuário final do qual o seu trabalho requer consultas e atualizações.

Havíamos resumido acima sobre como funciona o armazenamento secundário Arquivo de Dados. Agora, vamos falar sobre as suas principais diferenças entre si:

- O BD apenas necessita de um único repositório de dados, enquanto o Sistema de Arquivos precisa implementar os arquivos necessários para uma aplicação específica.

- Dentro do banco de dados temos o acesso a todos os usuários com um único espaço de armazenamento e as atualizações dos dados em somente uma estrutura. Já no Sistema de Arquivos, temos a redundância de arquivos com os mesmos dados armazenados e com uma perda de espaço no armazenamento, além do esforço adicional quando precisamos atualizar os dados.

Entendemos como funciona e a importância de um Banco de Dados, mas precisamos também falar do intuito desse resumo, que é o SGBD (Sistema Gerenciador de Banco de Dados). O mesmo é uma coleção de PROGRAMAS (Aplicações) que habilitam os usuários a criar e manter um banco de dados.

- Possuí um propósito geral do qual facilita o processo de definição, construção e manipulação de um banco de dados.
- Possui um controle em relação a redundância
- Tem pontos positivos no compartilhamento de dados
- Restrição de acesso não autorizado
- Possuí sistema de Backup e Recovery
- Força as restrições de integridade, como por exemplo a identificado do tipo de dados, unicidade de um dado, o dado não ser informado e o seu relacionamento entre os dados
- Possui dezenas de vantagens, sendo elas o Desenvolvimento de padrões, flexibilidade, tempo de desenvolvimento reduzido e disponibilidade de informações atualizadas
- Tem a constante preocupação do DBA e identificação dos perfis elaboras pelo analista e o projetista.

Em outros casos, dependendo do projeto em que está participando, também precisa pensar em quando não é necessário utilizar um SGBD. E as característica a seguir te mostram quando não usar um banco de dados:

- Custo desnecessário em relação ao uso de Sistema de Arquivos
- Alto investimento inicial
- Aplicações em tempo real com overhead de segurança, concorrência, recuperação e funções de integridade
- Banco de dados simples bem definido mas não vai se ter muitas alterações
- Múltiplos acessos não são necessários

Um banco de dados também possui suas linguagens, no caso proporciona dois tipos de linguagens, uma específica para as estruturas do banco (DDL) e a outra para as consultas a atualizações nas estruturas(DML e SQL).

- Linguagem de Definição de Dados (DDL):

    - É uma estrutura de dados representada por um conjunto de definições expressas por uma linguagem
    - O resultado do uso de uma DDL nos gera um arquivo chamado de dicionário ou diretório de dados, do qual é um arquivo de metadados.
    - Os metadados são as informações a respeito dos dados. Esses metadados são consultados no BD antes mesmo do dado "real" que está armazenado seja acessado ou então manipulado. O seu objetivo é fornecer mais informações aos dados, tornando mais fácil a sua organização e o armazenamento mais eficiente.
- Linguagem de Manipulação dos Dados (DML):
    - Permite o nosso acesso ou então a manipulação de dados de uma forma compatível ao modelo de dados apropriado.
    - A manipulação de dados pode ser resumida em Recuperar os dados, Inserir novos dados, Remover os dados ou modificar os dados já existentes no banco de dados
- Linguagem de Consulta dos Dados (SQL):
    - Sua responsabilidade é sobre a recuperação dos Dados
    - Pode ser definida como uma linguagem de consulta, mas ela também oferece meios para definir ou modificar uma estrutura de dados, além de ter uma especialização na questão de restrições de segurança

# ME-R (Modelo de Entidade-Relacionamento):

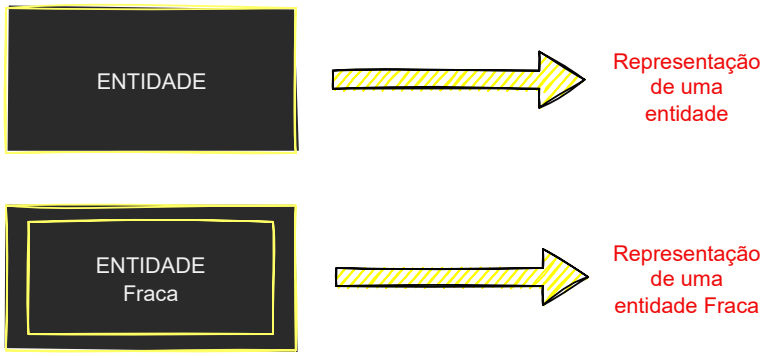
Em resumo, o ME-R é baseado na percepção do mundo real que consiste em um conjunto de objetos básicos chamados ENTIDADES e RELACIONAMENTOS entre esses objetos. O mesmo foi desenvolvido para facilitar o projeto de banco de dados, dessa forma permitindo uma melhora na especificação de um esquema de "negócio" (estrutura lógica geral no BD).

Antes de falarmos mais sobre o ME-R, precisamos saber o que é ENTIDADE, RELACIONAMENTO, Atributo e Chave:

## Entidade:

- Conceito fundamental da abordagem entidade-relacionamento (E-R) é o conceito de Entidade.
- As entidades são escritas em letra MAIÚSCULAS ou Caixa alta como substantivos
- São representados por um Retângulo
- Conjunto de objetos da realidade modelada sobre os quais deseja-se colecionar dados
- Pode ser concreta ou abstrata, por exemplo (disco, pessoa | curso, conceito...)
- É um conjunto de objetos que se deseja guardar dados, e é muito importante para o seu mundo real.

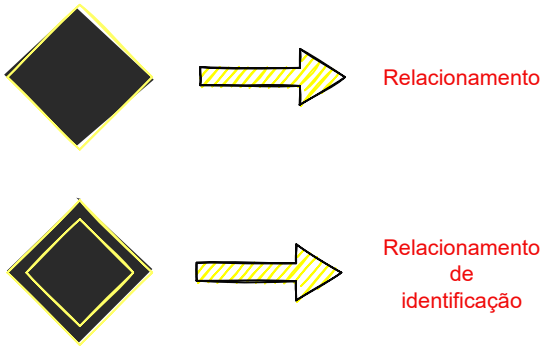
    - Suponha que haja um relacionamento de lançamento entre as duas entidade CONTA e TRANSAÇÃO. Caso a CONTA seja apagada, as suas transações também serão apagadas, entretanto caso a TRANSAÇÃO for apagada, a CONTA permanecerá intacta. Isso significa que a CONTA é dominante e a TRANSAÇÃO dependente da CONTA (entidade fraca)
- As entidades também pode ser classificadas em duas categorias que apresentam características diferentes, sendo elas Entidade Forte e Entidade Fraca. Elas são classificas assim por meio de um relacionamento de identificação aonde a Entidade Fraca não consegue existir sem estar relacionada a Entidade Forte.
- Pode ser representado da seguinte forma:




## Relacionamento:

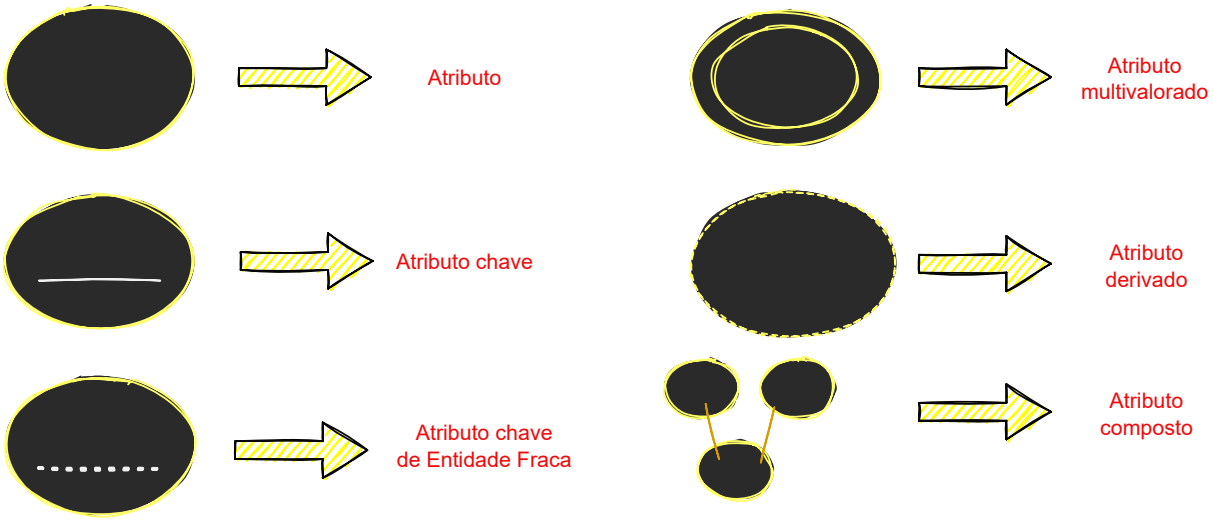
- É uma coleção de ocorrências das entidades relacionadas ou associadas
- Os relacionamentos são escritos em minúsculos e são representadas por verbo, pois implica que o relacionamento é uma AÇÃO.
- São representados por um Losango
- Quando uma entidade está associada/relacionada a um relacionamento, a função dessa entidade é chamada de papel de forma implícito.
- O relacionamento também pode possuir atributos descritivos onde ocorre tal relacionamento
- Tendo uma ocorrência particular de um relacionamento dentro de um conjunto de relacionamento do mesmo tipo, isso tem o nome de Instância de Relacionamento.

- Pode ser representado da seguinte forma:




## Atributo:

- é um dado associado a cada ocorrência de uma entidade ou um relacionamento
  - Uma melhor forma de visualizar é pelo Diagrama de Ocorrência, aonde possui a finalidade de reconhecer a forma como acontece um determinado relacionamento entre entidades.
- São representados por uma Elipse
- Uma entidade pode ser associada por um único ou vários atributos, e cada atributo existe um conjunto de valores permitidos que chamados de domínio.
- O atributo é capaz de mapear uma entidade em um domínio
- Existem atributos com características que podem ser representados da seguinte forma em um DE-R:




## Chave:

- É um dos atributos que permitem identificar unicamente uma entidade dentro de um conjunto de registros. Nela possuímos:
  - Chave Candidata: composição de mais que um atributo que pode identificar unicamente uma entidade

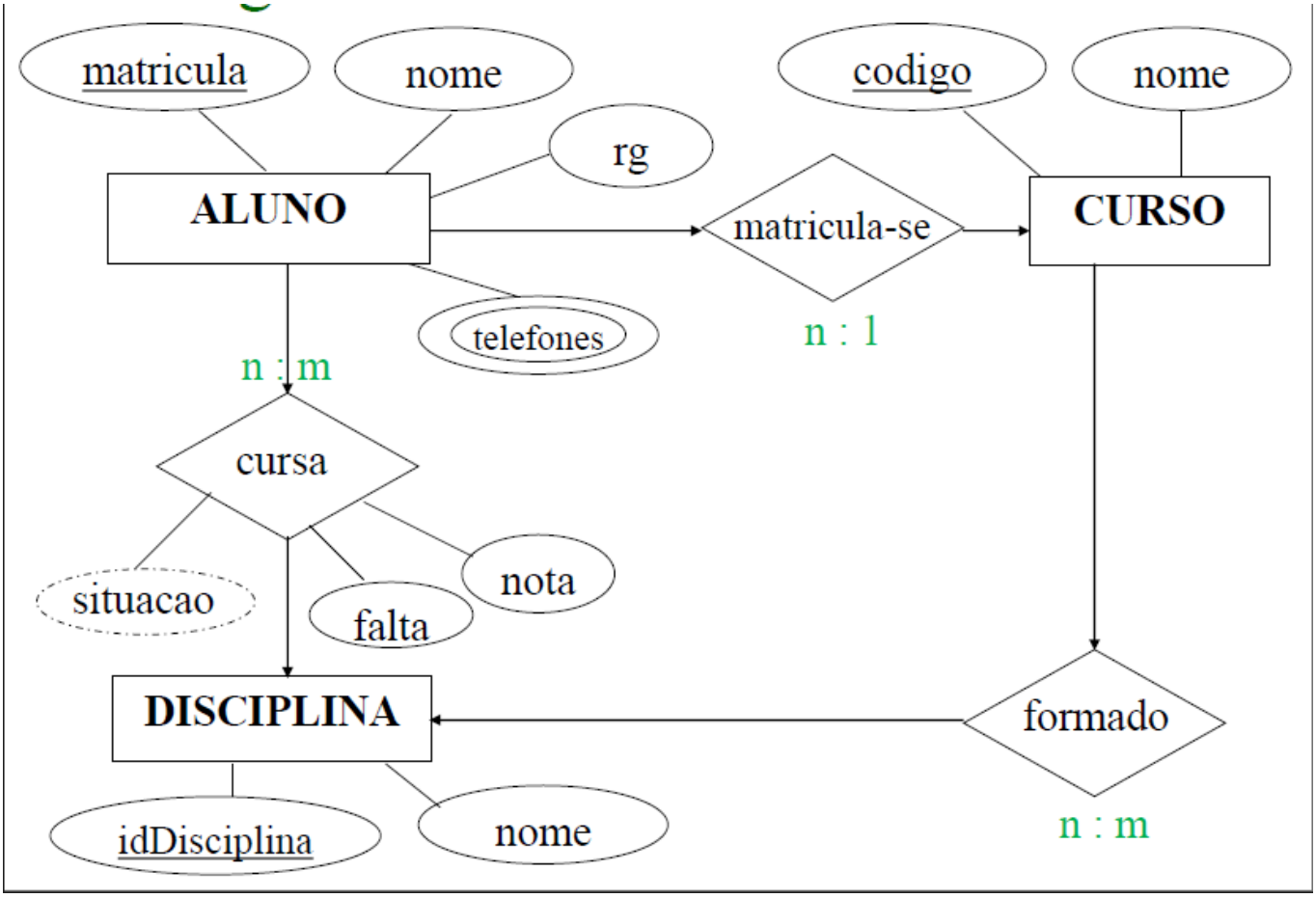
    - Chave Primária: Chave escolhida pelo projetista de dados para identificar unicamente os registros de uma entidade.

Agora que vamos sobre o básico do ME-R, iremos passar a falar sobre como funciona a sua diagramação. Esse mesmo possui um nome que é chamado de Diagrama Entidade-Relacionamento (DE-R).

- Estrutura lógica de um BD que pode ser representada graficamente em um diagrama
- O DE-R é composto por:
    - Retângulos que representam as Entidades
    - Losangos que representam os Relacionamentos
    - Elipses que representam os Atributos
    - Linhas (ou arcos) que ligam e fazem o direcionamento aos seus atributos e as entidades aos seus relacionamentos.

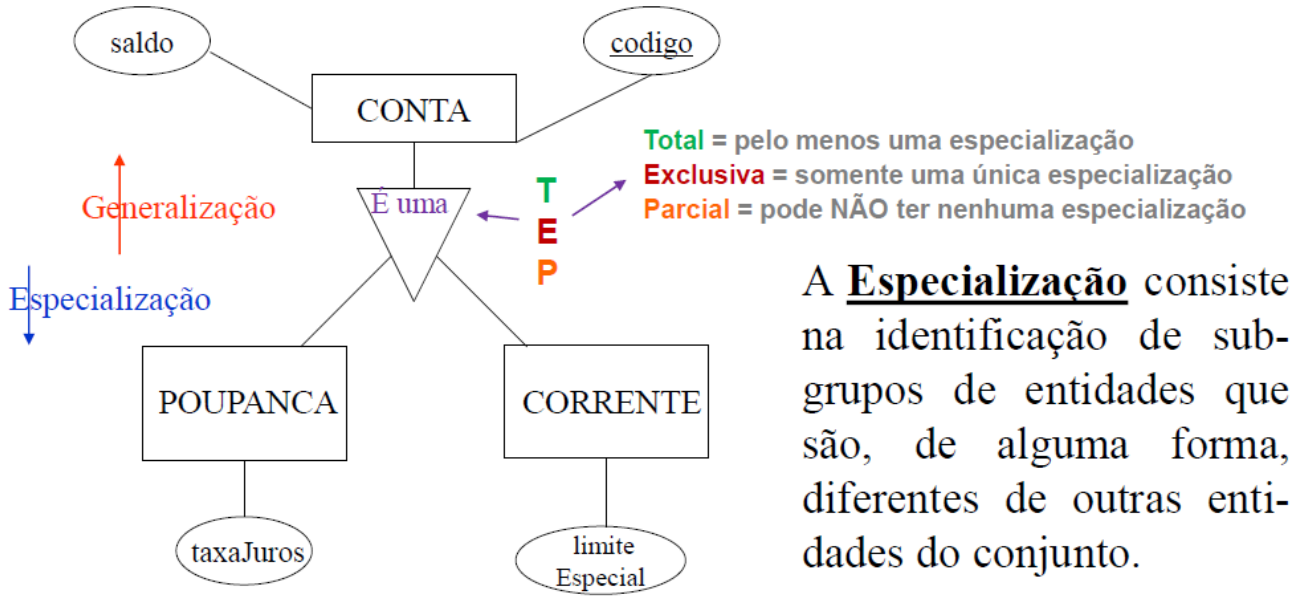
Dentro de um DE-R é necessário que haja o que chamados de Cardinalidade, do qual é uma restrição de mapeando que vai expressar o número de entidades dos quais outra entidade pode ser associada via um conjunto de relacionamentos. Nele temos:

- Um para Um (1:1): Entidade A está associada a uma única Entidade B, do mesmo modo que uma Entidade B está associada a uma única entidade A
- Um para Muitos (1:n): Entidade A está associada a várias Entidades de B, e uma Entidade de B está associada somente a uma única entidade de A.
- Muitos para Um (n:1): Entidade de A está associada a uma Entidade de B, e uma entidade de B pode estar associada a qualquer entidade de A.
- Muitos para Muitos (n:m): Entidade A está associada a qualquer Entidade B, e uma Entidade B está associada a qualquer Entidade A.
- Um exemplo de DE-R está logo abaixo:




Agora que entendemos sobre como é um **Diagrama Entidade-Relacionamento**, precisamos entender sobre alguns outros conceitos que ocorrem dentro do DER. Por exemplo, suponha que você tenha uma entidade CONTA e essa possua os atributos saldo e código da conta. De acordo com a regra de negócio adotada dentro da base de dados de um banco, o cliente pode criar dois tipos de conta, sendo eles Poupança e/ou Corrente, do qual possuem os mesmos atributos da entidade Conta mas cada entidade (Poupança e Corrente) tem os seus próprios atributos. Esse caso, é conhecimento como **Generalização** e **Especificação**.

- A **Generalização** é usada para priorizar as semelhanças entre tipos de entidades de nível superior e ocultar as suas diferenças, ou seja, esse conceito nos diz que agrupam entidades que possuem algumas diferenças entre elas mas também tem semelhanças com uma Entidade de nível superior. O melhor exemplo é a Conta, Poupança e Corrente, no qual as entidades Poupança e Corrente são um tipo de conta e são agrupada como sub grupos, enquanto a Conta é a entidade de nível superior e compartilha atributos semelhantes para todas as entidades de nível inferior que pertencem a essa categoria.
- A **Especialização** consiste na identificação de subgrupos de entidades que são, de alguma forma, diferentes de outras entidades do conjunto.

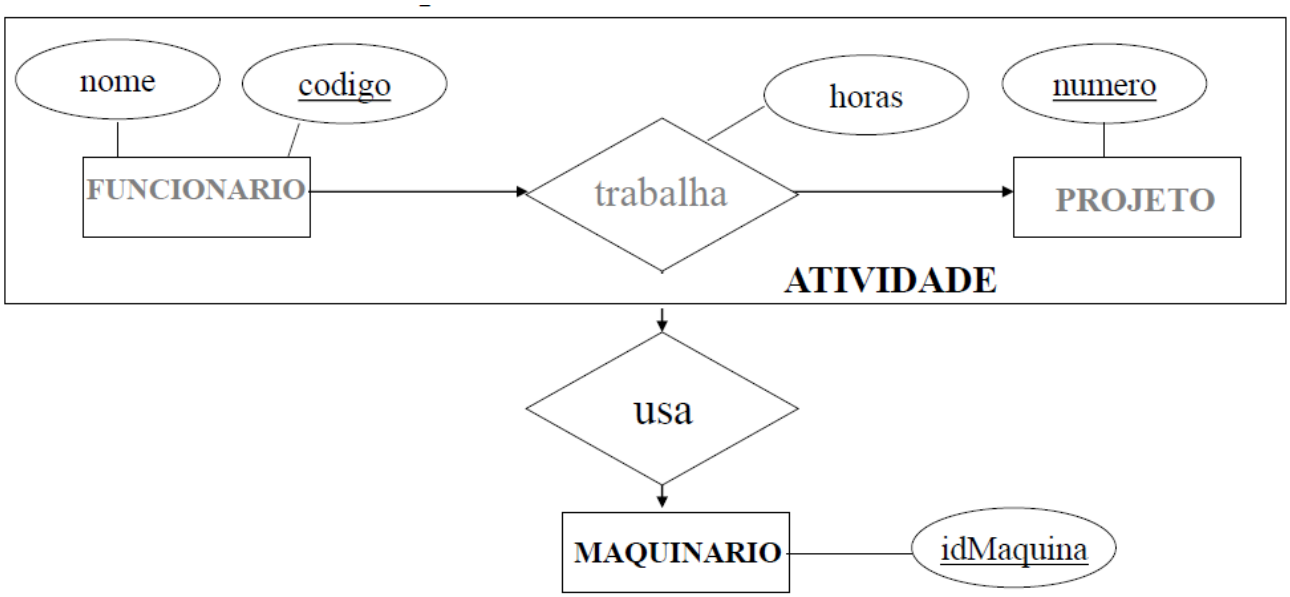



Nesse diagrama temos a presença de um triângulo de cabeço para baixo em vez de um losango, isso está significando que temos um tipo de Generalização e Especialização, aonde a super classe CONTA tem ali subgrupos como Poupança e Corrente. Dentro do losango, adicionamos uma das letras T, E ou P:

- T (Total): Significa que é obrigatório que a superclasse CONTA tenha uma subclasse, senão a CONTA não existe, ou seja, **pelo menos** uma especialização.
- E (Exclusiva): Significa que é obrigatório que a superclasse CONTA tenha uma subclasse e que **tenha uma** Especialização.
- P (Parcial): Significa que pode apenas existir a superclasse CONTA sem ter alguma subclasse, mas não pode ser tudo, ou seja, na Parcial a CONTA **pode ser Poupança ou Corrente**.

Temos também o conceito que é muito comum também dentro de um DER é o **Autorrelacionamento** aonde a Entidade se relaciona com ela mesma. Por exemplo, uma PESSOA pode se casar com nenhuma ou outra PESSOA, e a outra PESSOA só poderá se casar com uma PESSOA.

Temos então outro conceito chamado de **Agregação** (Entidade Associativa) no qual foi criado por conta de uma limitação do ME-R, aonde não é possível expressar relacionamentos entre relacionamentos.




Observando a imagem acima, percebemos que a entidade Funcionario se relaciona com a entidade Projeto, no qual gera o relacionamento **trabalha**. Sabendo que um funcionário trabalha em um projeto, esse mesmo pode usar ou não um maquinário, certo? Mas não é possível fazer relacionamento entre relacionamento (**trabalha** com **usa**). Por conta disso vem o conceito de Agregação, aonde é uma abstração por meio de qual relacionamentos são tratados como entidades de nível superior. Explicando melhor, o relacionamento entre Funcionario e Projeto é visto como um conjunto de entidades de nível superior, assim criado uma entidade nova chamada ATIVIDADE: essa entidade nasce do relacionamento entre Funcionario e Projeto. Dessa forma, é possível firmar o relacionamento entre ATIVIDADE e MAQUINARIO.

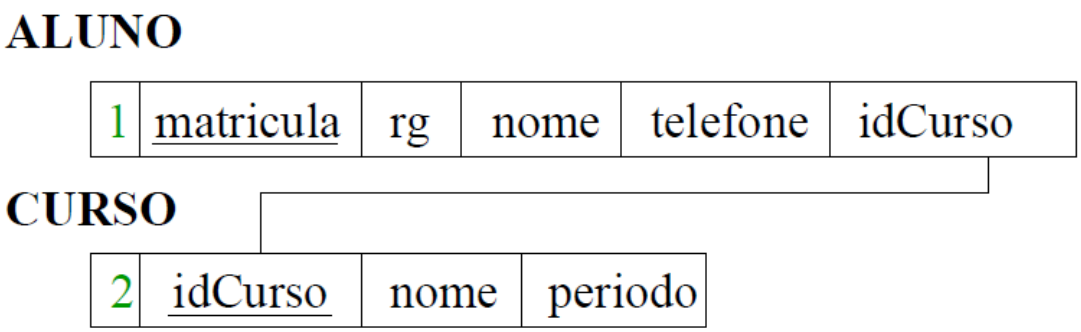
> Uma observação nesse conceito, ele é apenas usado quando aparecer um relacionamento entre relacionamento e não poder fugir disso. Dessa forma, somente dessa forma, é possível utilizar o conceito de Agregação
>
> Pode ocorrer também um relacionamento **ternário** aonde só é possível isso quando realmente for obrigatório associar, ao mesmo tempo, um par de entidades com uma terceira entidade. Quando não ocorre esta obrigatoriedade, recomenda-se o uso da agregação.

## Descrição da Relação:

A descrição da relação é utilizada quando queremos descrever as nossas entidades que estejam de acordo com o DE-R.

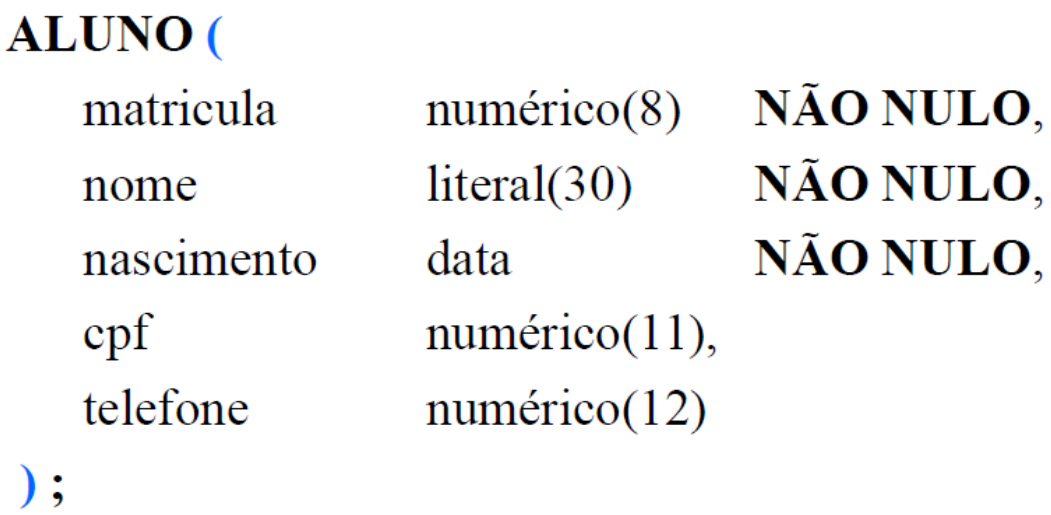
- Para esses casos, temos o **Diagrama de Esquemas**, utilizado para representar as entidades e seus atributos, assim facilitando o processo de identificação das entidades por

suas respectivas tabelas.




- **Número sequencial** do esquema indica uma única relação do projeto de banco de dados;
- **Arco** (ou linha) **de ligação** indica como o relacionamento é estabelecido entre as relações.

- Temos também o querido **Descrição dos Esquemas**. É utilizando para descrever os esquemas existentes no banco de dados, do qual podem ser descritos para posterior implementação física.
  - Essa descrição segue com uma sintaxe e semântica para serem observadas com precisão para que posteriormente ocorra a implementação física em SQL
  - Nesse momento é destacado os **tipos de dados** para serem reconhecimentos no banco de dados e as **restrições** que serão implementadas no nível de armazenamento dos dados no SGBD. Os tipos de dados são (n = comprimento/tamanho do valor | d= quantidade de dígitos depois da vírgula):
    - numérico(n, d) (-45 | 0 | 25.57)
    - literal(n) ('José Roberto' | 'porcelana' | 'A' )
    - data (dia, mês e ano | 23 / 10 / 2001)
    - horário (hora, minuto e seguindo | 10:32:51)
  - Cada esquema pode ser descrito como uma tabela, mas alguns aspectos são relevantes nesta descrição para identificar as restrições coerentes ao seu armazenando adequado.
  - O armazenando de dados em uma **relação** (tabela) normalmente, necessita de alguns dados obrigatórios e outros que são opcionais, ou seja, somente informados quando a situação necessita. A obrigatoriedade de um atributo é identificado pela expressão **NAO NULO** em uma Descrição de Esquema. Esse expressão é vinculada aos atributos que são obrigatórios na descrição de um esquema.
    - Logo, como ficaria então a descrição de esquema da entidade ALUNO, que possui como atributos: matricula, nome, nascimento, cpf e telefone?

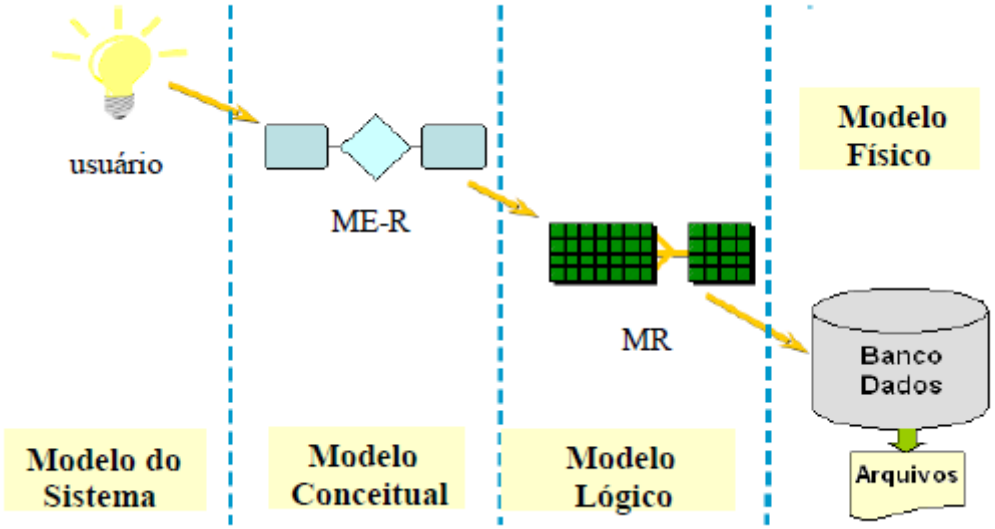



- Nesse esquema, além de tudo isso, ainda é necessário atender algumas restrições informadas que indicam os tipos de dados, tamanho e obrigatoriedade. Temos as seguintes restrições:
  - temos a restrição de chave primária (tupla que não pode ser repetida) para o atributo matricula, então ficaríamos com: **restrição Aluno_PK de Chave Primária (matricula)**
  - caso essa entidade necessite receber uma chave primária de outra entidade que está se relacionamento com a entidade Aluno, então teríamos a seguinte restrição: restrição ALUNO_EXEMPLO_FK de chave estrangeria (chave) que referencia a entidade EXEMPLO (chave)

## Modelo Relacional de Dados

Relembrando o que já vimos atrás:

- Conjunto de conceitos que são utilizados para descrever a estrutura de um banco de dados e é a principal ferramenta no fornecimento de informações sobre a abstração efetuada
- Temos os mapeamentos de modo de implementação: Modelo Conceitual -> Modelo Lógico -> Modelo Físico




## Características e Conceitos:

Agora vamos entrar no conteúdo sobre o Modelo Relacional de Dados e o nome correto para as entidades, atributos e etc em um banco de dados.

- **Esquema**: descrição da organização dos dados de um BD
- **Instância**: dados armazenados em um BD, em um momento específico, é chamado de instância

No Modelo Relacional, os dados são representados em um BD por meio de um conjunto de relações, denominadas de **tabelas**. Dentro dessas relações, é onde possui as informações sobre as entidades e relacionamento existente no **domínio da aplicação**.

De forma informal, as relações podem ser consideradas como uma tabela de valores, aonde cada linha nessa tabela vai representar uma coleção de valores de dados inter-relacionados.

Voltando sobre a terminologia do Modelo Relacional:

- **Tupla**: linha da tabela
- **Relação**: a tabela
  - Não há tuplas duplicadas em uma relação
  - A ordem não é relevante para diferenciar uma relação da outra
  - Possui ordem dos valores nas tuplas, uma vez que o esquema é definido como um conjunto de atributos
- **Atributo**: nome da coluna
  - Os valores dos atributos devem ser atômicos, não sendo divisíveis no componente
  - Os atributos multivalorados são representados por meio de uma **outra relação**. Os atributos compostos são representados pelos seus **componentes**.

- **Domínio**: é o conjunto de valores em que cada atributo pode assumir em uma determinada relação. Ou seja, se temos o atributo Sexo, o domínio desse atributo é Masculino (M) e Feminino (F).
- **Esquema da relação**: conjunto de atributos que descreve as características dos elementos a serem modelados.
- **Grau da relação**: é a quantidade de atributos da relação.
- **Instância da relação**: conjunto de valores que cada atributo, definido no esquema, assume em um determinado instante, assim gerando o **conjunto de tuplas**. Essas instâncias são as que formam os dados que são armazenados no BD.

## Chaves e Restrições de Integridade:

Agora que vimos sobre as terminologias e algumas das características das relações em um BD, precisamos falar sobre as **Chaves** e **Restrições**:

- Como foi dito acima em que não pode haver tuplas duplicadas, então deve haver um ou mais atributos que terá a responsabilidade de **identificar unicamente cada tupla**.
  - Esse atributo é chamado de **chave da relação**, e a sua definição é identificada como chave primária da relação.
  - Uma chave primária composta por mais de um atributo é chamada de **chave primária composta**
- As Restrições de Integridade são definidas por meio de:
  - **Restrições de Chaves**: cada atributo das chaves candidatas deve possuir valor único em todas as tuplas da relação.
  - **Restrição de Integridade de Entidade**: chave primária não pode assumir valor nulo nas tuplas da relação.
  - **Restrição de Integridade Referencial**: a inclusão de um novo atributos de uma relação que referência outra tupla de outra relação, vai surgir no meio disso um **novo tipo de chave**.
  - **Restrição de Integridade Semântica**: valores ou características que alguns atributos podem assumir no contexto de uma determinada aplicação (Ex: atributo sexo)
- A **Restrição de Participação** nos diz que a existência de uma entidade depende de sua participação em um tipo de relacionamento (**total** e **parcial**), sendo que a participação **total** está relacionada a **dependência de existência**.
  - **Participação Total**: resumidamente, essa participação significa em que uma entidade só existe somente se ela participa em uma instância de relacionamento. Um exemplo é que um empregado deve trabalhar em um departamento, e não existem empregados que não estejam em algum departamento.
  - **Participação Parcial**: é onde uma entidade não se limita a existência de outra. Considerando o exemplo acima mas agora com o relacionamento "gerencia", a

entidade Departamento não precisa se limitar a existência total da entidade Empregado, pois somente uma parte dos empregados gerenciam um departamento.

## Como mapear o ME-R para Modelo Relacional:

Para essa parte agora, é importante a gente entender como podemos mapear o nosso ME-R para o Modelo Relacional, sabendo que o ME-R é responsável por realizar uma representação conceitual dos dados de uma aplicação. Entretanto, essa representação é um pouco distante da forma como realmente as entidades e relacionamentos serão implementados. Visto isso, o Modelo Relacional traz uma forma de representação de dados que é mais próxima em que os dados se encontrarão quando forem definidos os arquivos para o Banco de Dados.

- Todas as entidades são mapeadas para uma relação contendo os mesmos atributos do ME-R
- Uma entidade fraca é criada a relação contendo todos os seus atributos mais a chave primária da entidade forte, sendo ela representada dentro da entidade fraca como uma **chave estrangeira**.
- Quando estamos falando de **cardinalidades**, devemos seguir algumas regras para cada tipo:
  - Cardinalidade 1:1 -> dentre as entidades que estão participando na relação, escolha aquela que possui a participação **Total** e inclua a sua chave primária em outra entidade como chave estrangeira.
  - Cardinalidade 1:N -> observe a relação e escolha a entidade presente no lado N, assim acrescente como chave estrangeira, a chave primária da entidade que está do lado 1. Lembrando que a regra não se aplica nos relacionamentos de identificação)
  - Cardinalidade N:M -> para esse caso é criado uma nova relação (tabela) aonde é adicionando como chaves estrangeiras, as chaves primárias das entidades que participam do relacionação, e os atributos do relacionamento.
- Para os **atributos multivalorados**, vamos chamá-los de *A*. Para esse caso, vai ser criado uma nova relação denominada *R* na qual terá os mesmos atributos de *A*, e mais a chave primária da entidade (ou relacionamento) em que *A* é atributo. Caso o atributo multivalorado seja composto, então será incluído os seus componentes na relação *R*.

Sabendo sobre como identificar os tipos de dados de cada atributos e as restrições (regras de negócio) que envolvem as tabelas, podemos elaborar uma **descrição do esquema**.

- A **chave primária** é a que permite o relacionamento com outras relações de forma consistente, por meio de uma **chave estrangeira**. Para que haja o relacionamento entre duas entidades, tem que existir essas duas chaves.
- Dentro da descrição, temos o seguinte formato geral para a descrição da chave estrangeira e algumas siglas:

    - restrição {**identificador**} de chave estrangeira (**atributo da relação**) que referencia {**outra relação**}(atributo)
    - os **identificadores** das restrições são usadas com siglas PK (Primary Key) e FK (Foreign Key) para identificar as chaves primárias e estrangeiras existentes em uma relação.
- Lembrando da restrição de integridade referencial aonde a implementação de uma chave primária em outra relação, na qual estabelece um relacionamento consistente com essa outra relação, **torna a chave primária em chave estrangeira na outra relação**.
    - A chave primária está "visitando" a outra relação (esquema) que participa do relacionamento. É por conta disso que se dá o nome de estrangeira. Geralmente, as relações "visitadas" possuem uma chave primária, além da chave estrangeira.

# Controle de Consistência

Vimos sobre o que é um SGBD, como documentar utilizando o ME-R e como evoluir para chegar até o nosso Modelo Relacional de Dados e depois o físico. Pois bem, agora precisamos falar sobre a **Consistência dos dados** e a **Normalização**.

- O projetista avalia o mapeamento (ME-R) e o resultado disso é um mapeamento aonde respeita algumas características que o conceitue como um **mapeamento eficiente**. A mesma é identificada eficiente quando as relações representam os dados de forma consistente e com o mínimo de redundância controlada.
- Para esse caso, o projetista é orientado a partir de uma aplicação de um conjunto de critérios informais que vai auxiliar ele:
    - **Semântica dos atributos de uma relação**: Quando um conjunto de atributos é agrupados em uma relação (Tabela), é determinado/associado um significado a eles. Esse significado é o que identifica como os mesmos devem ser interpretados os atributos.
    - **Redundância de Valores**: É necessário evitar a duplicação dos dados em um banco de dados, pois com isso pode causar algumas anomalias no DML, como:
        - Anomalia no processo de Inserção
        - Anomalia no processo de Remoção
        - Anomalia no processo de Alteração
        - Anomia no processo de Consulta
    - **Valores nulos em tuplas**: Valores nulos podem possuir diversas interpretações, como por exemplo: o atributo não é aplicado para aquela tupla; valor do atributo para a tupla é desconhecido ou o valor é conhecimento, mas encontra-se ausente. É sempre bom evitar atributos que possam ser nulos, caso não seja possível evitar alguns, então é importante verificar se as tuplas acontecem em pequeno número na relação.

É importantíssimo manter a Consistência dos dados na definição de esquemas relacionais. Esse controle pode ser exercido pelos seguintes três níveis:

- Gerenciador (SGBD)
- Programa ou Aplicativo
- Pela própria construção de um sistema (É o mais eficiente)

O terceiro nível é o mais eficiente dos três, já que ele não implica em perda de desempenho no SGBD durante a sua execução. Esse mesmo controle é obtido por meio do Modelo Relacional, aonde construímos as relações seguindo algumas regras que irão garantir a manutenção de certas propriedades, ou seja as Forma Normal.

- Esse caso é chamado de Normalização, e é considerada como a aplicação de uma série de regras e normas que são denominadas **Formas Normais**, e algumas delas são baseadas nas dependências funcionais.
- Os valores de alguns atributos contribuem para a identificação de outras, por tanto uma dependência funcional tem o significado que se conhecer o valor de um atributo, então é possível determinar o valor de outro. A notação para essa teoria é a seguinte:

> **A → B, ou seja B depende de A**
>
> Para o caso de uma dependência multivalorada (conhecer o valor de um atributo -> determinar os valores de um conjunto de outros atributos) a notação é:
> A →→ B, no qual A determina muitos B's

- Uma das maneiras de controlar a consistência é por meio das Dependências Funcionais existente entre os atributos armazenados.

O armazenamento de dados em uma tabela, normalmente, necessita de alguns dados que sejam obrigatórios, mesmo que outros dados podem ser informados somente para alguma situações. Um exemplo é quando um Aluno vai ser matrícula e possua os atributos (matricula, nome, nascimento, rg, cpf, telefone) mas o mesmo não possui ainda o CPF e nem o Telefone. Para esse caso, temos as seguintes informações:

- A obrigatoriedade de um atributo é definido pela expressão **NOT NULL**, no qual deve ser vinculada para todos os atributos da relação que são obrigatórios. Ou seja, esse atributo não pode ser Nulo.

Dessa forma, a gente fica com o seguinte exemplo de uma Descrição da Relação utilizando o que vimos acima:

```
CREATE TABLE ALUNO(
    matricula INT(8) NOT NULL,
```

```
    nome VARCHAR(30) NOT NULL,
    dtNascimento DATE NOT NULL,
    rg VARCHAR(10) NOT NULL,
    cpf BIGINT(11),
    telefone BIGINT(12)
);
```

Levando em conta o exemplo acima, também vamos ter a identificação de restrição de integridade referencial, no qual a chave primária simples é o atributo **matricula**. Assim, essa restrição deve ser descrita da seguinte maneira:

```
CREATE TABLE ALUNO(
    matricula INT(8) NOT NULL,
    nome VARCHAR(30) NOT NULL,
    dtNascimento DATE NOT NULL,
    rg VARCHAR(10) NOT NULL,
    cpf BIGINT(11),
    telefone BIGINT(12),
    CONSTRAINT ALUNO_PK PRIMARY KEY (matricula)
);
```

Observação: para uma chave composta basta escrever: CONSTRAINT ALUNO_PK PRIMARY KEY (matricula, dtNascimento).

A restrição de **integridade referencial** implanta a chave primária na qual vai permitir um relacionamento com outras relações por meio de uma chave estrangeira, denominada de chave secundária (FK). Esse mesmo tipo de chave pode ser identificada de forma clara na descrição da relação na qual será criada.

As restrições são identificadas por Constraint e correspondem a implementação de regras negociais relevantes relacionadas diretamente as tabelas do projeto de banco. Dessa forma, com a implementação correta, identificação adequada e significada para cada uma dessas regras, respeitando as suas normas de definição as seguintes regras:

- Nome da restrição começa com o nome da tabela ao qual a regra de negócio pertence
- A chave estrangeira o nome da tabela é referenciada com um underline
- A identificação termina com uma sigla correspondente ao tipo de restrição implementar(PK ou FK)

Além disso, uma última observação é a organização de cada tabela e os seus registros, aonde na descrição deve primeiro possuir os atributos mais importantes e obrigatórios e sem seguida deles os atributos que não são obrigatórios.

# Normalização:

Com os estudos do tópico anterior, os mesmos vão contribuir com as definições de algumas Formas Normais para trazer um nível de consistência na modelagem proposta.

A Normalização permite ao Projetista controlar o quanto de Consistência será garantida pela maneira de construção do sistema(estrutura). Entretanto deve analisar até que ponto a Normalização pode chegar, pois normalizar demais diminuir a eficiência dos aplicativos que utilizam a base de dados, ou então, normaliza menos facilita as probabilidades das inconsistências na BD.

- O processo consiste em analisar cada uma das relações que compõem o projeto e verificar qual forma normal cada tabela vai atender. Assim, levando em conta o número mais alto que todas as tabelas atendam, então será classificado que o projeto proposto atende essa determinada Forman Normal.
- Normalizando um projeto acaba provocando a decomposição de esquemas insatisfatórios de algumas relações em novas relações que irão atender as formas normais, permitindo um projeto mais eficiente, e evitando as Anomalias.
- O processo de Normalização é identificado por meio de números (primeira, segunda e terceira) forma normal.

As Formas normais são:

- **PRIMEIRA (1 FN)**: ela é a normalização principal por conta de ser considerada parte da própria definição de uma relação. Essa forma normal diz que a relação só está na primeira forma normal se todos os domínios de atributos possuem apenas valores atômicos (simples e indivisíveis), ou seja, sem atributos compostos e multivalorados. Essa 1 FN é uma das maneiras de controlar a consistência por meio da própria estrutura do sistema.
- **SEGUNDA (2 FN)**: somente está na segunda forma normal se a relação já estiver na 1 FN e todos os atributos que não participam da chave primária são dependentes diretos de toda a chave primária. Essa normalização evita a inconsistência devido a duplicação de informações, além da perda de dados em operações que contenham Anomalias. Dessa forma, deve ser verificado o seguinte:
  - Os grupos de atributos que dependem da mesma parte da chave primária
  - Retirar da relação todos os atributos de um desses grupos
  - Criar uma nova relação contendo esse grupo como atributo não chave, e os atributos que determinam esse grupo como chave primária
- **TERCEIRA (3 FN)**: Somente está na terceira se a relação já estiver na 2 FN e não possuir nenhuma dependência transitiva (ou indireta). Uma dependência transitiva é quando temos: **A → B e B → C, logo A → C**. Dessa forma, evita que qualquer atributo não chave seja dependente funcional de outro atributo não chave.

- É valido lembrar que na 3FN não vai ter na descrição da relação o atributo derivado do ME-R, porém o mesmo vai aparecer no DE-R com a expressão explícita (derivado) como comentário no respectivo atributo, pois o dado é de interesse do usuário.

# Álgebra Relacional

Além do controle de consistência e da normalização para a nossa base de dados, também devemos falar sobre a álgebra relacional que abrange as operações de conjuntos e relacionais dentro do banco de dados.

## Operações de Conjuntos:

As operações de conjuntos é quando aplicam operadores em duas relações que respeitam a Compatibilidade de União, na qual ambas as relações precisam ter atributos em seus esquemas que pertençam respectivamente aos mesmos domínios.

As operações de conjuntos são:

- União (∪): resultado da união de duas relações que consiste no conjunto de todas as tuplas que pertençam em ambas as relações.

  > A união B (A ∪ B) = Relação com todas as tuplas dos dois conjuntos

- Interseção (∩): resultado em que duas relações consiste no conjunto de todas as tuplas que aparecem ao mesmo tempo nas duas relações.

  > A interseção B (A ∩ B)

- Diferença (-): resultado em que a diferença entre as duas relações (R e S) consiste no conjunto de tuplas que aparecem na relação R, mas não na relação S.

  > A minus B (A-B) | B minus A (B-A) => Conjuntos/Resultados diferentes

- Produto Cartesiano (x): aqui as relações não precisam respeitar a compatibilidade de união, assim o conjunto gerado é uma relação aonde possui tuplas formadas pela combinação dos atributos pertencentes a ambas as relações.

  > A cartesiano B (AxB) = Conjunto de todos os possíveis pares de A com B

## Operações Relacionais:

Fora as operações de conjuntos, temos então as Operações Relacionais na qual são operações de relacionamento entre as relações da sua base de dados. As operações são:

- Operação de Seleção (SELECT): gera uma relação com os mesmos atributos que satisfazem a uma determinada seleção. É um operador unário, assim só é executado sobre apenas uma relação, uma tupla de cada vez.
    - Ele segue com a seguinte fórmula: **σ** [condição] (relação)
    - É possível utilizar os operadores relacionais (!=, =, <, <=, >, >=, ^ e ⋁)
- Operação de Projeção (PROJECT): selecionar e exibir na tela os atributos de uma relação de acordo com a lista de atributos selecionados e na mesma ordem que foram colocados na lista. A relação resultante não permite repetições nas tuplas produzidas
    - Ele segue com a seguinte fórmula: **π** [atributos] (relação)
    - Uma operação relacional vai sempre resultar em uma outra relação que pode ser utilizada na elaboração de consultas mais complexas.
- Operação de Junção (JOIN): utilizada para combinar tuplas de duas relações (operação binária) em uma tupla simples. É combinada de acordo com uma condição indicada
    - Ela segue a seguinte fórmula: ⊙ [<condição>] (relação1, relação2)

Vamos ver alguns resumos relacionamos a Linguagem SQL a partir de agora:

- A linguagem SQL é declarativa e os SGBDs Relacionais a têm como padrão. Suas características são originárias da Álgebra Relacional. Então logo temos:
    - Linguagem declarativa (Não-procedural): É a linguagem que escreve O QUE se deseja realizar, sem se preocupar em dizer como será feita. Assim o sistema que vai determinar a forma de como fazer.
        - Um programa é declarativo se descreve o que ele faz e NÃO como seus procedimentos funcionam.
        - Oracle PL/SQL é procedural e MySQL é não procedural
    - Linguagem procedural: Linguagem que fornece uma descrição detalha de como um processamento será realizado, operando sobre um registro ou uma unidade de dados de cada vez.
- A linguagem proporciona dois tipos principais de recursos em sua linguagem:
    - DDL (Data Definition Language): específica para manipular as estruturas do Banco de Dados
        - o resultado do uso da DDL é constituído em um arquivo chamado de dicionário de dados, no qual é um arquivo de metadados. Os metadados são dados a respeito dos dados. Dentro de um BD, esse arquivo é consultado antes mesmo que o dado real seja modificado ou então manipulado.
    - DML (Data Manipulation Languagee): utilizada para expressar consultar e atualizaões sobre o que é armazenado nessas estruturas.
        - na questão da manipulação dos dados, a gente pode ser entender por: Recuperação dos dados armazenados no BD; Inserção de novos dados no BD

e/ou Remoção e modificação de dados do BD.

    - SQL (Linguagem de Consulta de Dados): essa linguagem é uma parte do DML responsável pela recuperação dos dados (pesquisa ou consulta)
        - mesmo indicando como uma linguagem de consulta, ela também possui recursos para definição de estruturas de dados, de modificação de dados e especificação de segurança.

- Por meio da linguagem SQL, um SGBD realiza alguns processos:
    - Definição: é a criação descritiva do esquema que atenderá as necessidades do BD
    - Construção: é a inserção das instâncias iniciais no banco de dados
    - Manipulação: é a realização de consultar e atualizações sobre os dados decorrentes do dia a dia que usa este Banco de Dados.
- Uma coisa a lembrar que é bastante importante, é que nas características de um SGBD, tem que se preocupar com a restrição de acesso não autorizado.
- Para a criação de um Banco de Dados, no qual é o conjunto de registros de dados dispostos em estrutura regular que possibilita o seu armazenamento, então teremos os seguintes códigos:

```
CREATE DATABASE <nome da base de dados>;
DROP DATABASE <nome da base de dados>;
SHOW DATABASES; -- lista as bases de dados (só no mysql)

-- Para estabelecer conexão com uma base de dados já criada, use o comando abaixo
USE <nome da database>;
```

- A primeira fase para qualquer banco de dados se inicia com o uso das instruções DDL, pois só com elas vai ser possível criar recursos no Banco de Dados.
- Em uma base de dados, vai haver relações que possuam atributos que sejam obrigatórios. Para esse caso, a obrigatoriedade de um atributo é representada pela expressão **NOT NULL** (não nulo). Essa restrição deve estar vinculada a todos os atributos que são obrigatórios em uma tabela. Segue o exemplo:

```
CREATE TABLE ESTADOS (
    sigla CHAR(2) NOT NULL
    nome VARCHAR(20)
);
```

- A expressão **NULL** significa que não existe valor, assim ela não contém dados. O maior erro que se comete sobre esses atributos, é a atribuição de um valor nulo em um atributo

numérico. Explicando melhor, o valor nulo é diferente de zero, pois zero é um valor, enquanto que o NULO é a ausência de valor.

- Para a remoção de uma tabela em uma base de dados, podemos utilizar o seguinte comando:

```
DROP TABLE <nome_da_tabela>;
```

- Agora que entendemos como funciona a parte de Definição, podemos começar a manipular as relações identificando o domínio e armazenando seus dados. Dessa forma, vai começar a construção do banco de dados. Então temos para esse caso as seguintes instruções:

```
INSERT INTO <nome_da_tabela> (valores) VALUES ('exemplo1', 'exemplo2'); --
utilizada para inserir tuplas em uma relação

DELETE FROM <tabela>; -- usada para remover uma ou mais tuplas da relação
DELETE FROM <tabela> WHERE <condicao>;

UPDATE <tabela> SET <atributo> = <valor>; -- ou
UPDATE <tabela> SET <atrbuto> = <valor> WHERE <condicao>; -- permite a
atualização de um ou mais atributos em uma tupla, ou em um grupo de tuplas de
uma relação.
```

- Falando a respeito da instrução INSERT, ela deve respeitar as características de cada tipo de dados para inserir dados na base de dados corretamente nas formas exigidas para cada tipo de dado em SQL.
- Agora precisamos falar da instrução que é a essência da linguagem SQL. Por meio dessa instrução, é possível recuperar dados de um banco de dados de uma maneira simples apenas dizendo ao BD quais informações foram selecionadas para serem recuperadas. Essa instrução pode ser dividida em quatro partes básicas:
  - SELECT (seguido de atributos que se esperar ver (obrigatório))
  - FROM (seguido da origem dos dados no BD (obrigatório))
  - WHERE (seguido das restrições de recuperação (opcional))
  - ORDER BY (seguido da forma como os dados serão ordenados (opcional))
  - O símbolo asterisco significa que todos os atributos da relação informada deverão ser recuperados.
- Segue o exemplo:

SELECT | idEMpregado, nome
FROM | EMPREGADO

WHERE | salario < 0
ORDER | BY nome, salario;

- Temos também a questão da implementação de um atributo com auto incremento na tabela. Esse auto incremento, permite que o atributo inteiro único seja gerado quando um novo registro for inserido no MySQL, assim nunca repetindo esse número.
- A expressão para essa implementação é AUTO_INCREMENT e deve ser indicada em um único atributo da tabela. O seu valor inicial padrão e 1 e o incremento será de 1 em 1.
  - ao inserir novos valores na tabela, não é necessário especificar o valor do atributo de auto_incremento
  - só é permitido usar um atributo de auto incremento por tabela, geralmente do tipo inteiro.
  - é necessário também da restrição (constraint) NOT NULL, que será configurado automaticamente, mas deve ser incluída na documentação e no script que cria a tabela corretamente
  - exige criação da chave primária (PK) no atributo.
- Segue o exemplo logo abaixo:

```
CREATE TABLE EQUIPE (
    idEquipe INT(5) NOT NULL AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,

    CONSTRAINT EQUIPE_PK PRIMARY KEY (idEquipe)
) ENGINE = InnoDB AUTO_INCREMENT = 1;
```

- Entretanto, outros bancos de dados empregam outras formas de controle sobre valores únicos e sequenciais relevantes as implementações relacionais em SGBD's. UM exemplo é que o ORACLE e o PostgreSQL utilizam o SEQUENCE. A instrução para realizar isso é: CREATE SEQUENCE <nome_da_sequencia>
- A mesma apresenta alguns parâmetros, vamos vê-los:
  - nome_da_sequencia: Nome da sequência, não podendo ser o mesmo de uma tabela
  - Increment by **n**: Especifica de quanto será o incremento ou decremento. O padrão é 1.
  - Start with **n**: Especifica o primeiro número a ser gerado. O padrão é 1.
  - MaxValue n: Especifica o valor máximo que a sequência gerada pode atingir. O padrão é nomaxvalue, indo até 1027.
  - Cycle | nocyle: Indicia que ao atingir o valor máximo a numeração continuará a partir do valor inicial. O default é nocycle.
  - Cache n | nochace: Especifica quantos valores o banco de dados pré-aloca e mantém em memória. O padrão é 20.

# Controle de Acesso e Engenharia Reversa:

Neste conteúdo, iremos tratar sobre o que é e como funciona o Controle de Acesso e também sobre o que se trata Engenharia Reversa:

## Engenharia Reversa:

- Consiste em uma análise diferente da sequencial habitual que estudamos até agora. Esse tipo de engenharia corresponde a uma atividade com um produto existente (sendo software, peça mecânica, um banco de dados) e tenta entender:
  - COMO este produto funciona.
  - PARA QUE ele serve.
  - O QUE ele faz,
  - QUAIS são suas propriedades, etc.
- Pode ser compreendido também como um processo de investigação sobre um sistema para criar sua representação em um nível de abstração mais alto. Essa abstração acontece mapeando as propriedades de uma base de dados gerando seus modelos Conceitual e Lógico para melhor compreensão dos seus objetivos e funcionamento.

## Controle de Acesso:

- Pelo nome, faz referência ao controle empregado para permitir ou não o acesso de pessoas a localidades (prédio, propriedade, sala, etc) especialmente, abordado sobre a segurança física.
- Aplicando o Controle de Acesso para a segurança de dados, a ideia se mantém a mesma que é permitir ou não o acesso aos dados e informações armazenadas pelos seguintes processos:
  - Autenticação: identifica quem acessa o sistema
  - Autorização: determina o que um usuário pode fazer no sistema
  - Auditoria: diz o que o usuário fez usando o sistema
- Em resumo e contextualizado encima da Segurança dos dados, o controle de acesso seria em questão a habilidade de permitir ou negar o uso de um objeto (entidade passiva, como um sistema ou arquivo) por um sujeito (entidade ativa como um indivíduo ou um processo).
- Temos um processo em dois passos que identifica a autentica quem pode acessar o sistema e seus dados
  - Identificar quem é o usuário que está solicitando o acesso ao sistema (normalmente pelo seu nome).
  - Autentica a identidade do usuário verificando sua credencial, por exemplo a senha pessoal.

- A autorização pode definir quais direitos e permissões o usuário do sistema tem acesso, em que após a sua autenticação determinará o que ele pode fazer ou visualizar no sistema. Temos algumas técnicas de controle de acesso, que são três:
  - **Discricionário**: política de controle de acesso determinada pelo proprietário do recurso
  - **Obrigatório**: política de acesso é determinada pelo sistema e não pelo proprietário do recurso.
  - **Baseado em Papéis (roles)**: abordagem que define os direitos e permissões baseados no papel que determinado usuário desempenha na empresa.
    - é uma abordagem para restringir o acesso aos usuários autorizados.
    - aqui define os direitos e permissões baseados no papel que cada usuário realiza na organização. Essa estratégia simplifica o gerenciamento das permissões dadas aos usuários e pode constituir perfis comuns entre todos.
    - as permissões de acesso e direito sobre objetos são dados para qualquer grupo ou indivíduos. Esses indivíduos podem pertencer a um ou mais grupos, então adquirir permissões cumulativas ou retirar qualquer permissão não faz parte de todo seu grupo.
- É possível trabalhar com Controle de Acesso por meio da linguagem SQL. Veja o exemplo:

```
CREATE USER <<nomeUsuario>
    IDENTIFIED BY <<senha>> | EXTERNALLY
    DEFAULT TABLESPACE
    TEMPORARY TABLESPACE
    QUOTA ON
    PROFILE <<nomeProfile>>
    PASSWORD EXPIRE
    ACCOUNT
```

- Precisamos ter uma ideia sobre o que significa os Privilégios, possui em todo SGBD possui um conjunto de usuário que se utilizam de seus dados e recursos, e cada vez que tenham autorizações e permissões concedidas através de privilégios para usufruírem dos recursos e/ou conteúdos armazenados no SGBD.
- Privilégio é a autorização fornecida ao usuário do SGBD para acessar e/ou manipular recursos, estruturas e dados armazenados. Nesse caso, temos alguns tipos de recursos do SGBD.
  - Sistema: Permissão de executar ações sobre o SGBD, seus objetos e estruturas (vários tipos de privilégios distintos)
  - Objeto: Permissão para acessar e manipular um objeto ou estrutura específica (os dados armazenados)
- Dessa forma, para deixar melhor explicado, temos

| Sistema | Objeto |
|---|---|
| Create Table | SELECT |
| CREATE USER | INSERT |
| ALTER TABLE | UPDATE |
| DROP USER | DELETE |
| e mais | e mais |

O usuário é dono dos objetos que cria, tendo todos os privilégios sobre ele, além de poder conceder privilégios para outros usuários sobre os seus objetos.

- A gente temos o que seria os Papeis, ou seja as suas atribuições. Um papel agrupa privilégios e simplifica a administração dos usuários.
  - Temos que Criar Papel (perfil) => CREATE ROLE papel
  - Temos que excluir Papel (perfil) => DROP ROLE papel
- Essas atribuições pode ocorrer por meio de outros usuários ou então pelas próprias ROles, que posteriormente terão os usuário associados. Essa forma de passa ao outros usuários a sua permissão é vista da seguinte forma:
  - Da perspectiva do SISTEMA temos: GRANT privilégio | papel TO usuário PUBLIC WITH ADMIN OPTION
  - Da perspectiva do OBJETO temos: GRANT privilégio ON objeto WITH GRANT OPTION;
  - E para remover o privilégio, temos: REVOKE privilégio ON OBJETO FROM USUARIO | PUBLIC
- Então, segue o exemplo:

```
GRANT CREATE TABLE, CREATE VIEW TO DAVI -- atribuir alguns privilégios para o usuário Davi

-- ou

CREATE ROLE FUNCIONARIO -- atribuir privilégios ao perfil
GRANT FUNCIONARIO TO DILSON
```

- Para conceder privilégios para outros usuários, podemos permitir também que esse usuário possa repassar os privilégios para outros usuários, da seguinte forma:
  - Utilizando o WITH ADMIN OPTION
    - Aqui teremos a opção para privilégios do sistema.
    - Pode ser concedida a usuários ou papéis

        - e permite ao usuário conceder ou revogar o privilégio de qualquer usuário ou papel, e também alterar ou remover o papel concedido.
    - Utilizando o WITH GRANT OPTION
        - É uma opção para privilégios de Objetos
        - Fornecida somente para usuários
        - permite ao usuário conceder o privilégio para qualquer usuário ou papel e alterar ou remover o papel concedido.

# Visão em Banco de Dados:

Aqui iremos apresentar sobre o que se trata de Visão em Bancos de Dados:

Conceitualmente, uma visão em um banco de dados é o que corresponde a um conjunto de tuplas resultantes de uma consulta armazenada sobre uma ou mais tabelas do SGBD. Dessa forma, os usuários podem visualizar a View para consultar os seus conjuntos de dados assim como fariam com as tabelas convencionais que armazenam dados fisicamente no SGBD.

Podemos apresentar combinações ou subconjuntos lógicos de dados por meio da criação de visões sobre as tabelas existentes no SGBD. Além disso, uma view é uma tabela lógica baseada em uma ou mais tabelas reais existentes no SGBD ou sobre outra view.

- A view em si não contém dados, é igual a uma janela, ali você exibir alguns dados provenientes de tabelas que persistem do SGBD.

Vamos falar sobre as suas propriedades:

- A View é baseada em uma tabela denominada Tabela Base.
- É armazenada como uma instrução SELECT no Dicionário de Dados.
- As VIEW não são objetos físicos do SGBD, e por conta disso não ocupam espaço em disco. Assim, elas são definidas como um objeto que não armazena dados, pois a mesma não é uma tabela, pois é composta dinamicamente por uma consulta que é previamente analisada e otimizada.
- A instrução de criação de uma VIEW é DDL.

Para criar uma VIEW, basta seguir o exemplo abaixo:

```
CREATE VIEW CANDIDATOS_2004 (
 registro candidato, cargo, cidade, estado) AS
SELECT p.codigo, p.nome, l.cargo, l.cidade, l.sigla
FROM PESSOA p, LOCAL l
WHERE p.idPssoa = l.idPessoa
AND l.ano = 2004;
```

Assim, pode conferir com o comando DESC CANDIDATOS_2004. E instruções DDL como Create, ALTER e DROP view podem operar sobre uma view.

Em relação as suas características:

- Se alterar os dados da tabela base da view, consequentemente, irá alterar os resultados gerados pelas consultas armazenadas na view.
- Utilizar a view simplifica a interação entre usuário final e banco de dados. Além de poder utilizar como segurança, restringindo o acesso aos usuários.
- Em bancos com tecnologia No-SQL, as view são a única maneira de consultar dados.
- Restringe o acesso a dados
- Facilita consultas complexas
- Permite independência de dados
- Apresenta exibições diferentes dos mesmos dados
- Representação de dados contidos em outras tabelas (tabelas base)
- Trata resultado de uma consulta como uma tabela (consulta armazenada | tabela virtual)
- Espaço de armazenando apenas para consulta (SELECT) que define a view. A consulta é executada cada vez que a view é acessada.

As suas principais utilidades são:

- Aumento de **segurança** por propiciar uma visão limitada e controlada dos dados que podem ser obtidos de uma base de dados por seus usuários. Vai depender do SGBD utilizado.
- Não será necessário o processo de **otimização** da consulta quando for realizada.
- **Armazenamento** de consultas complexas ou executadas com muita frequência traz: Simplicidade para o usuário e abstração.
- **Apresentação** dos dados com menor complexidade ou em diferentes perspectivas
- **Isolamento** de aplicações em relação a alterações de esquema.

As Visões podem ser classificadas em Simples ou Complexas. A diferença entre ambas está relacionada na realização das operações DML pela view, assim manipulando as tabelas reais (base):

- View Simples:
  - Deriva dados de uma única tabela
  - Não contém funções ou agrupamentos de dados
  - Permite a execução de operações DML
- View Complexa:
  - Deriva dados de várias tabelas

- Contém funções ou agrupamentos de dados
- Nem sempre permite a execução de operações DML

Temos as questões das operações e privilégios sobre as views:

- Visões não atualizáveis (read-only) => Seleção
- Visões atualizáveis (updatable) => Seleção, inserção, remoção e atualização
- Os privilégios da view podem ser:
  - Proprietários da view (owner):
  - Operações requerem privilégios adequados sobre a tabela base. E para conceder basta ser o dono das tabelas base ou se tiver recebido os privilégios com grant option
  - Outros usuários requerem privilégios para view.

Temos a opção WITH CHECK OPTION para casos aonde restringimos o acesso a inserção de tuplas que não atendam as definições da visão, por exemplo:

```
CREATE OR REPLACE VIEW V_PROFESSOR_DOUTOR AS
SELECT * FROM PROFESSOR WHERE titulação = 'Doutor'
WITH CHECK OPTION;
```

Dessa forma, caso tente realizar alguma operação DML de inserção aonde 'titulação' seja diferente de 'Doutor', essa operação não será realizada.
A opção CHECK OPTION só deve ser usada em visões atualizáveis, que não permitirão a execução de operações que violem a condição de seleção que define a visão.

Temos a questão sobre a VIEW de junção de tabelas. O exemplo abaixo mostrará como será feito:

Suponha as tabelas a seguir:

- ALUNO = {nome, matricula, idade, dtNasc}
- DISCIPLINA = {sigla, nome, nCred, professor, livro}
- MATRICULA = {sigla, numero, aluno, nota}

Então teremos:

```
CREATE VIEW V_MATRICULA
 (matricula, nome, sigla, disciplina) AS
SELECT a.matricula, a.nome, d.sigla, d.nome
FROM ALUNO a JOIN MATRICULA m ON a.matricula = m.aluno
JOIN DISCIPLINA d ON m.sigla = d.sigla;
```

O mesmo pode ocorrer para view atualizáveis (Updatable Join views). A regra geral para esse caso é que as operações DML podem modificar apenas 1 das tabelas base por fez.

- O conceito fundamental é a respeito da Preservação de Chave em ao menos uma das tabelas da junção

A Preservação de Chave diz que se toda chave na tabela base é única (chave candidata) no resulta da junção, ela depende da semântica e não da instância atual da view. Para que uma view seja atualizável, é suficiente que pelo menos uma tabela subjacente tenha preservação de chave, entretanto mesmo que todas tenham preservação de chave, a atualização só pode ocorrer em uma delas por vez.

Assim, voltamos as características para a questão do Updatable join view:

- Para INSERT:
  - Somente pode envolver colunas provenientes de 1 tabela com preservação de chave
  - WITH CHECK OPTION => Não são permitidas inserções
- Para UPDATE:
  - Colunas atualizáveis são aquelas provenientes de 1 tabela com preservação de chave
  - WITCH CHECK OPTION => atributos de junção (ON) e atributos de tabelas, usadas mias do que uma vez não são atualizáveis.
- Para DELETE:
  - De acordo com a Doc. do Oracle: "somente se há exatamente uma tabela com preservação de chave"
  - Testes: remoção de registros da primeira tabela usada na definição da view
  - WITH CHECK OPTION => se a tabela base que possui preservação de chave for usada mais do que uma vez, não é possível deletar.

# Transação em Banco de Dados:

As transações são as operações que formam uma única unidade lógica de trabalho. Por exemplo, a transferência de um valor de uma conta para outra conta, na visão do cliente, consiste de uma operação única e simples. Entretanto, no Banco de Dados ela envolve várias operações para que a transferência seja executada com sucesso.

Resumidamente, é uma unidade de execução de programa que acessa a manipula dados no BD. Geralmente ela consiste na execução de um programa (instruções) elaborado com:

- linguagem de manipulação de dados (alto nível)
- linguagem de programação
- uma ou mais instruções DML, uma instrução DDL e uma instrução DCL.

**A transação consiste em todas as operações a serem executada a partir do começo até o fim da transação**.

Para que possa assegurar a integridade dos dados, o BD deve garantir algumas propriedades, dos quais são conhecidas por **A.C.I.D.** :

- **A**tomicidade => Todas as operações da transação são refletidas corretamente no BD ou nenhuma será.
- **C**onsistência => A execução de uma transação isolada preserva a consistência no BD (inicio-fim)
- **I**solamento => Cada transação não toma conhecimento das outras transações concorrentes. **Assegurar está propriedade é responsabilidade de um componente do sistema de BD denominado Controle de Concorrência**.
- **D**urabilidade => Depois da transação completar-se com sucesso, as mudanças que ela faz no BD, persistem até mesmo se houver falhas no sistema. **Assegurar esta propriedade é responsabilidade de um componente do sistema de BD denominado Gerenciamento de Recuperação.**

As transações passam por diversos Estados até serem concluídas, assim vamos ter os seguintes:

- **Ativa**: permanece neste estado enquanto está sendo executada.
- **Efetivação Parcial**: após execução da última declaração.
- **Falha**: descobre-se que a execução não poderá ser efetivada.
- **Abortada**: transação desfeita, restabelecendo o BD ao início.
  - Diz-se que uma transação foi abortada (rolled back) somente se ela entrar nesse estado.
- **Efetivada**: após a conclusão com sucesso.
  - Diz-se que a transação foi efetivada (commited) só se ela entrar nesse estado.

Uma transação é concluída apenas se estiver no estado Efetivada ou Abortada. Uma transação de compensação pode desfazer os efeitos de uma transação efetivada, porém nem sempre isso é possível. Essa transação tem a responsabilidade de criação e execução deixada a cargo do usuário.

Caso a transação esteja no estado de falha (erro de hardware, lógico, leitura, etc...) não pode prosseguir com a sua execução normal, assim vai precisar ser desfeita (UNDO). Dessa forma, ela passa para o estado de Abortada, aonde pode:

- Reiniciar a transação => possível somente para erros de hardware ou software e não pela lógica da operação

- Encerrar a transação => erro lógico normalmente, pois a aplicação (programa) deverá ser refeito.

Necessária observação que a operação que reinicia uma transação consiste na criação de uma nova transação para ser processada.

Agora, podemos falar a respeito de transações concorrentes:

- Agiliza a realização da tarefa desejada, mas pode trazer complicações em relação a consistência dos dados no BD. Seria bem mais fácil se mantivesse as execuções das transações sequencialmente.
- Há duas possibilidades que incetiva a concorrência:
  - Operações da CPU e E/S podem ser feitos em paralelo
  - Mistura de transações simultâneas no sistema (curtas e/ou longas)
  - Acessa diferentes partes do BD
  - Reduz atrasos imprevisíveis
  - Diminui o tempo médio de resposta
  - Reduz ociosidade da CPU, discos e outros dispositivos.

As transações concorrentes acabam comprometendo a propriedade de consistência do BD, por conta disso, para permitir a concorrência eficiente, é necessário analisar a escala de execução (schedules) das transações envolvidas. Por exemplo:

> Suponha que há diversas contas com vários lançamentos que acessam e atualizam estas contas, e essas transações são denominadas T1 e T2 que transferem fundos de uma conta A para outra conta B.
>
> Assim, temos o nome de Throughput, aonde é a quantidade de transações que podem ser executadas em um determinado tempo.

Assim, a transação é feita da seguinte forma:

> T1: leia(A);
> A = A - 50;
> escreva(A);
> leia(B);
> B=B+50;
> escreva(B);
>
> T2: leia(A);
> aux = A * 0.10;
> A = A - aux;

```
escreva(A);
leia(B);
B = B + aux;
escreva(B);
```

Essas escalas são sequenciais, aonde cada uma consiste em uma sequencia de instruções de várias transações em que as instruções que pertencem a uma única transação aparecem agrupadas.

Para o caso da execução concorrente, o BD deve garantir a sua consistência por meio da execução concorrente, garantindo que qualquer escala concorrente executada tenha o mesmo efeito de usa execução sequencial.

- O sistema BD deve controlar a execução concorrente das transação
- Deve ser necessário para que o estado BD permaneça consistente

# Índice:

O índice é uma estrutura na base de dados que é conveniente para os processos que precisam ter agilidade em relação aos registros que não se encontram ordenados fisicamente. Um grande exemplo é imaginar uma grande consulta aonde deseja obter todos os resultados de um única empresa ou agência dentro de várias contas. Acabar lendo todos os dados para encontrar somente aqueles que lhe interessa se torna ineficiente comparado ao caso de obter um acesso direto a esses registros. O índice garante esse acesso direto ao registro que gostaríamos, e assim torna a busca mais eficiente.

Portanto, o(s) atributo(s) que serão usados para procurar um registro em um arquivo é chamado de **chave de procura** ou então **chave de acesso**. E por meio dessas chaves é possível então acrescentar em um arquivo várias chaves de acesso que estarão de acordo com a necessidade da situação. Geralmente, é interessante que se tenha mais de um índice para um arquivo de dados.

Essa nova estrutura adicional consiste na definição de índices para os arquivos de dados, assim definindo os **arquivos indexados**.

- Nos arquivos indexados podem existir tantos índices quantas forem as chaves de acesso aos registros.
- Um índice irá servir como uma entrada para cada registro do arquivo, e essas entradas estarão ordenadas pelo valor da chave de acesso.
- Os índices são formatos pelo valor de um atributo chave do registro e também pela sua localização física no interior do arquivo.

- É gerada também uma estrutura aonde essa é a tabela de índices na qual é armazenada e mantida em disco.

Como dito acima sabemos que os índices são construídos a partir de um ou mais atributos de uma relação, e em SQL ele é criado por meio da instrução: **CREATE INDEX**.

- A sua forma geral é: **CREATE [UNIQUE] INDEX <nome_indice_idx> ON <nome_tabela> (<atributos(s)>)**.
- As instruções CREATE INDEX e DROP INDEX são DDL.
- Um exemplo em SQL:

```
CREATE INDEX ESTADOS_NOME_IDX ON ESTADOS (nome);
```

- O "UNIQUE" é opcional mas utilizado na criação do índice possui o efeito no qual irá assegurar que não existirá valores duplicados no índice em que foi criado.
- É válido lembrar que um índice único é criado a partir do momento em que uma instrução SQL cria uma chave primária em uma tabela. E o bancos de dados avisam quando é realizado uma tentativa de criar um índice em uma tabela que já exista sobre o mesmo atributo.
- Uma tabela pode possuir vários índices diferentes envolvendo o atributo ou atributos que serão as chaves de pesquisas importantes.

Para apagar um índice no SQL basta utilizar a instrução DDL "DROP INDEX". Essa operação é utilizada a partir da alteração da tabelo do índice, como pode ver no exemplo abaixo:

```
ALTER TABLE ESTADOS
    DROP INDEX ESTADOS_NOME_IDX;
```

Essa mesma instrução apaga os índices independente deles serem UNIQUE ou então envolverem mais que um atributo em sua construção.

Para grandes arquivos de índices há um problema na questão da sua eficiência na manipulação, e para esse problema é necessário algumas técnicas sofisticadas nos índices. Existem dois tipos básicos de índices:

- **Ordenados**: se baseiam na ordenação de valores
- **Hash**: se baseiam na distribuição uniforme dos valores determinados por uma função.

Das técnicas listadas abaixo, nenhuma é melhor que a outra. O que difere é o seu contexto em que serão aplicadas e avaliadas pelos seguintes fatores:

- **Tipos de acesso**: Encontrar registros com um atributo determinado ou dentro de uma faixa de valores.
- **Tempo de acesso**: Tempo gasto para encontrar um item de dados ou um conjunto de itens (tuplas).
- **Tempo de inserção**: Tempo gasto para incluir um novo item de dados, incluindo o tempo de localização do local correto e a atualização da estrutura de índice.
- **Tempo de exclusão**: Tempo gasto para excluir um item de dados, sendo incluído o tempo de localização do registro, além do tempo de atualização do índice.
- **Sobrecarga de espaço**: espaço adicional ocupado pelo índice, compensando este sacrifício pela melhoria no desempenho.

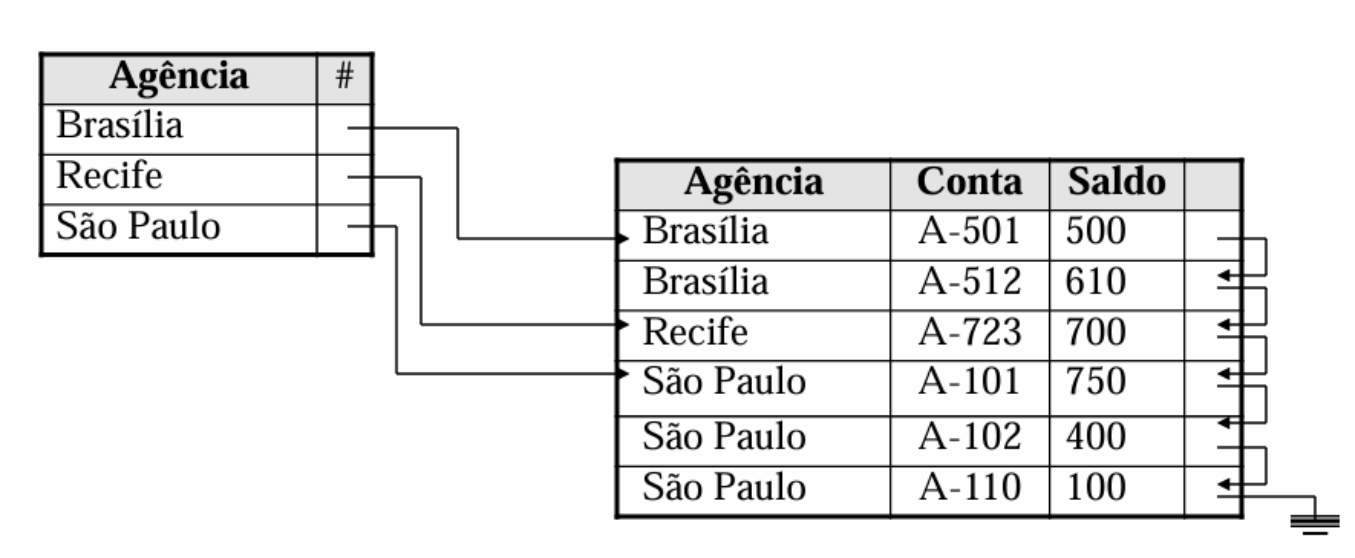
## Índices Ordenados:

É usado para quando se deseja conseguir acesso aleatório rápido sobre os registros de um arquivo.

- Os índices estão associados a uma chave de procura, na qual armazena a chave de procura de forma ordenada e associa a ela os registros que possuem aquela chave.
- Os registros em um arquivo indexado podem ser armazenados (eles próprios) em alguma **ordem**.
- Um arquivo pode ter diversos índices, com diferentes chaves de procura, estando ele ordenado sequencialmente pelo seu **índice primário**.
- Os índices primários em um arquivo são as **chaves primárias**, entretanto isso nem sempre ocorre.
- O índice primário consiste em um arquivo ordenado cujos registros são de tamanho fixo, no qual contém dois campos: 1 - mesmo tipo do campo chave no arquivo de dados; 2 - ponteiro para o bloco do disco.
- É um índice seletivo por conta de possuir entradas para um subconjunto de registros (bloco) ao invés de possuir apenas uma entrada para cada registro do arquivo de dados.
- Por conta de sempre manter os registros no arquivo ordenados, os processos de inserção e remoção acabam sendo um grande problema para esse tipo de estrutura.

Há dois tipos de índices ordenados, que são o denso e o esparso:

- Índice Denso (Não-Clusterizado):
  - Um registro de índice aparece para cada valor distinto da chave de procura no arquivo
  - O registro contém o valor da chave de procura e o ponteiro para o primeiro registro de dados com esse valor.
  - Há autores que utilizam essa expressão para identificar quando um registro de índice aparece para cada registro no arquivo de dados.




- Índice Esparso (Clusterizado):
  - Um registro de índice é criado apenas para alguns dos valores do arquivo de dados.
  - Parecido com os índices densos, o registro contém o valor da chave de procura e um ponteiro para o primeiro registro de dados com esse valor.
  - O sistema usa as entradas no índice para navegar até o bloco certo e da seguinte maneira:
    - A maior chave de procura no índice que ainda seja menor ou igual ao valor que você quer encontrar. Um exemplo para isso seria em uma tabela ordenada por CPF no qual o índice esparso é criado com base em blocos de 10 registros. Quando queremos encontrar o CPF "00123456789", o índice é consultado para encontrar a maior chave menor ou igual ao CPF que queremos. Ele pode acabar encontrando o CPF "00123456780" por exemplo, que aponta para um bloco específico, e assim o sistema fica responsável de fazer uma busca sequencial nesse bloco para encontrar o registro com o CPF que queremos.
  - Também é conhecido por Índice de Cluster, no qual tem os registros de um arquivos fisicamente ordenados por um campo não chave, e podendo serem distintos ou não.

Também há o **Índice Secundário**:

- Ele consiste em um arquivo ordenado que não usa o mesmo campo de ordenação como índice.
- As chaves de procura especificam uma ordem diferente da ordem sequencial do arquivo, podendo seus valores serem distintos para todos os registros ou não.
- Eles melhoram o desempenho das consultas que usam chaves diferentes da chave de procura do índice primário, mas impõem uma sobrecarga significativa na atualização da base de dados.

- O projetista da base de dados tem o dever de decidir quais índices secundários são desejáveis baseado na estimativa da frequência de consultas e atualizações.

## Índices de Níveis Múltiplos:

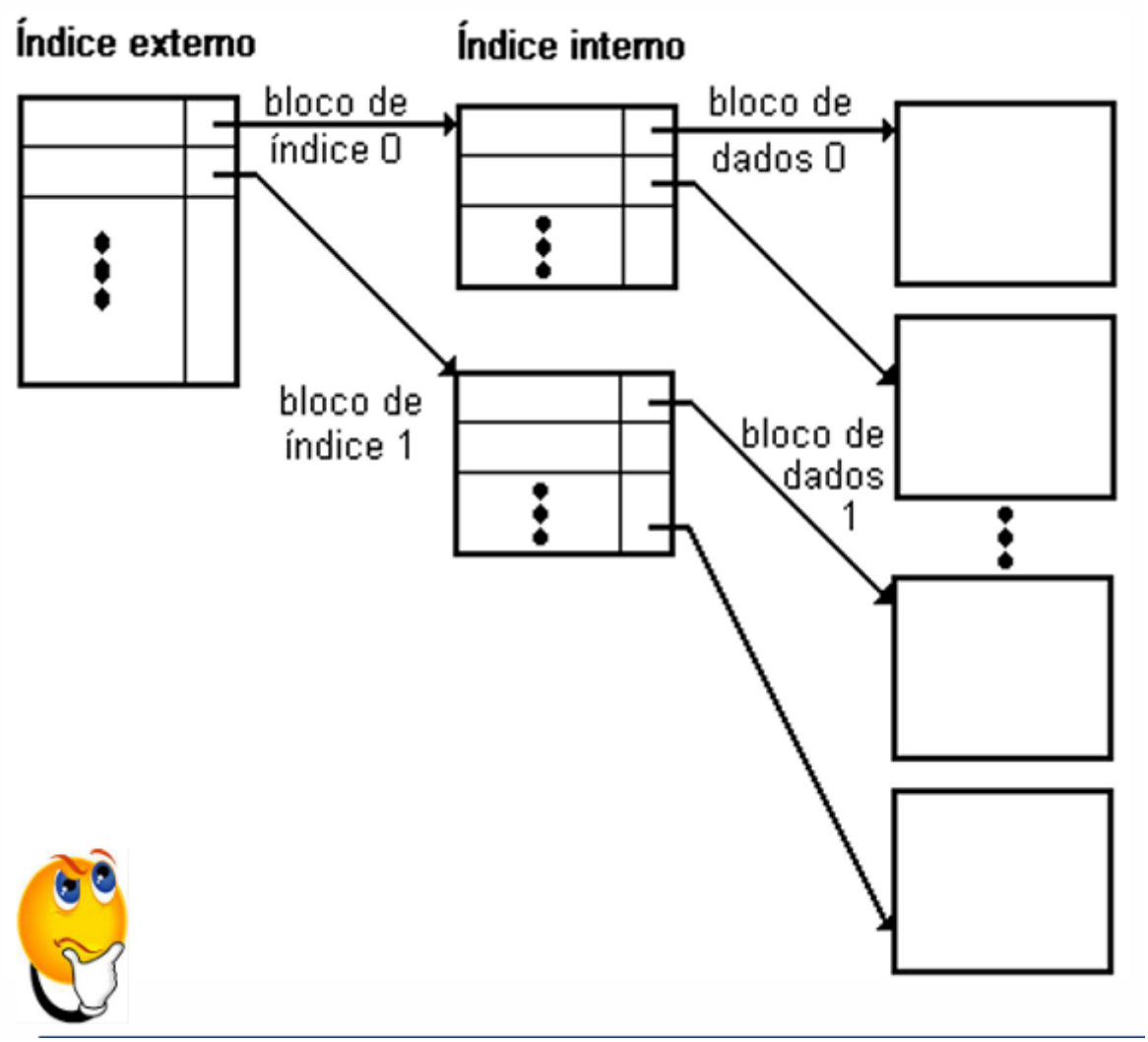
- Vai ser utilizado para casos onde o arquivo de índice é muito grande para ser eficiente.
- Registros de índices são menores que os registros de dados
- Índices grandes são armazenados como arquivos sequenciais em disco.
- A busca em um índice grande também é muito onerosa

As possíveis soluções para o caso da agilização de grandes índices, seriam:

1. Cria um índice Esparso como índice primário
2. Realizar busca binária sobre o índice mais externo, ou seja, encontrar maior valor da chave de procura que seja menor ou igual ao valor desejado.
3. O ponteiro indica o bloco do **índice interno** que será varrido até encontrar o maior valor da chave de procura que seja menor ou igual ao valor desejado.
4. O ponteiro neste registro indica o bloco do arquivo que contém o registro procurado.

É utilizado dois níveis de indexação:

- Caso o nível mais externo ainda seja muito grande;
- Não cabe totalmente na memória principal;
- Pode ser criado um novo nível;
- Criar quantos níveis foram necessários;




## Busca ou Procura usando Níveis Múltiplos:

- Para esse caso vai requerer um número significativamente menor de operações de entrada e saída que a busca binária.
- Cada nível do índice corresponde a uma unidade de armazenamento físico (trilha, cilindro ou disco).
- Os níveis de índices múltiplos estão relacionados as estruturas de árvores como as árvores binárias usadas na indexação de memória.
- Independente do índice que for utilizado, o mesmo deve ser atualizado sempre que um registro for inserido ou removido do arquivo de dados:
  - Remoção: Localizar o registro; Remover o registro, caso exista; Atualizar o índice (denso ou esparso).
  - Inserção: Localizar o registro utilizando a chave de procura; Incluir os dados (registro); Atualizar o índice (denso ou esparso).

## Organizando os arquivos com Hashing:

Caso deseje que a localização de dados seja eficiente nos arquivos sequenciais, então é necessário o acesso a estrutura de índice (tabela) ou então o uso da busca binária na qual vai acarretar em mais operações de entrada e saída. Uma alternativa para esse caso seria o uso de técnicas de hashing, na qual os arquivos seriam organizados por meio de uma função Hash:

- Utilizando o Hash seria evitado o acesso a uma estrutura de índice.
- Proporcionaria um meio para a construção de índices.
- Teria acesso direto ao endereço do bloco de disco que contém o registro desejado.
- Aplica-se uma função sobre o valor da chave de procura do registro para conseguir-se a identificação do bloco de disco correto.
- É utilizado o conceito de Bucket para representar uma unidade de armazenamento de um ou mais registros.
- Normalmente equivalente a um bloco de disco, mas pode denotar um valor maior ou menos que um bloco de disco.

Em relação a Função Hash, a mesma é ideal distribuir as chaves armazenadas uniformemente por todos os buckets, de forma que todos os buckets tenham o mesmo número de registros. E no momento do projeto, não se sabe exatamente quais os valores da chave de procura que serão armazenadas no arquivo. Dessa forma, desejamos que a função Hash atribua os valores da chave de procura aos buckets atentando a uma distribuição:

- Uniforme
- Aleatória: o valor de hash não será correlacionado a nenhuma ordem visível externamente parecendo ser aleatória)

Em resumo, as funções Hash necessitam ter um pleno cuidado ao ser utilizada:

- Ruim: resultaria em procuras que consumam um tempo proporcional a quantidade de chaves no arquivo
- Boa: oferecia um tempo de procura médio (constante e pequeno), independente da quantidade de chaves de procura no arquivo.
- Existem outras variações na aplicação das técnicas de hashing, por exemplo o hashing aberto, linear probing e etc, mas a técnica descrita é a preferida para banco de dados.
- Há algumas desvantagens com essa técnica:
    - A função de Hash deve ser escolhida até o momento no qual o sistema será implementado.
    - Como a função de hash mapeia os valores da chave de procura para um endereço fixo de bucket, ela não poderá ser trocada facilmente, caso o arquivo que esteja sendo indexado aumente muito ou diminua.

## Overflow de Bucket:

Buckets nem sempre tem os espaços para inserir um novo registro, e quando um bucket não possui o espaço suficiente então logo dizemos que ocorreu um Overflow de Bucket.

Pode ocorrer por várias razões, como por exemplo:

- Buckets insuficientes: conhecer o total de registro quando a função de hash for escolhida.
- Desequilíbrio (skew): Pode ocorrer por duas razões abaixo
  - Registros múltiplos com a mesma chave de procura
  - Distribuição não uniforme realizada pela função de hash escolhida.

Se for levar em consideração alguns cálculos é possível reduzir a probabilidade de overflow de buckets, no qual esse processo é conhecido como **fator de camuflagem**.

São necessários ter cuidados com o algoritmo de procura para o overflow:

- A função hash é usada sobre a chave de procura para identificar qual o bucket do registro desejado.
- Todos os registros do bucket devem ser analisados, quando a igualdade com a chave de procura acontecer, inclusive nos bucket de overflow

# Processamento e Otimização de Consultas:

## Processamento de Consultas:

O processamentos de consultas é as atividades que estão envolvidas em extrair dados de um banco de dados, por exemplo:

- Tradução da linguagem de alto nível do BD para expressões que podem ser implementadas no nível físico do sistema de arquivo
- Otimização
- Avaliação das consultas

Em relação as consultas e o seu custo, é dito que o custo de uma consulta é determinado, principalmente, pelo seu acesso ao disco que no qual é mais lento em relação ao acesso a memória principal. Em virtude disso, há muitas estratégias que são possíveis para processar uma consulta, ainda mais se ela for complexa. Por tanto, uma **BOA** estratégia é quando o custo é baixo em relação ao nível de acesso ao disco, e ela é **RUIM** quando o custo é alto.

Quando a estratégia é boa, então vale a pena o sistema gastar tempo e esforço para o processo de uma consulta. Os passos básicos no processamento de uma consulta são:

- Análise sintática e tradução
- Otimização
- Avaliação

Ela acaba seguindo o seguinte fluxo:

> Consulta -> Analisador Sintático e Tradutor -> Expressão algébrica relacional -> Otimizador -> Plano de Execução -> Avaliação -> Saída da Consulta

Nesse processo, o banco vai avaliar se a sua consulta feita é uma boa consulta ou uma ruim. No final, o banco sempre vai entregar uma consulta melhor em relação ao que você fez. Por isso que se a estratégia for boa, levando em conta a sua consulta, o banco vai gastar um pequeno tempo para processar e responder a consulta.

Sabemos que as linguagens de consulta relacionais são declarativas ou algébricas, assim permite que nós podemos especificar o que uma consulta deve gerar sem precisar se preocupar em informar ao sistema sobre como deve operar para fornecer o resultado da consulta. Dessa forma, baseado na especificação da consulta, um otimizador irá gerar uma variedade de planos equivalentes de execução para cada consultas e dentre os planos irá escolher a que é menos onerosa ao sistema.

Logo abaixo iremos poder ver um exemplo sendo aplicado em uma consulta na qual ela pode ser tradução para expressões algébricas relacionais:

```
SELECT saldo
FROM conta
WHERE saldo < 2000
```

```
σ [saldo < 2000] (π saldo (Conta))
π saldo (σ [saldo < 2000] (Conta))
```

Na consulta acima percebe-se que ele realiza uma procura em todas as tuplas de contas por saldos menores que dois mil (saldo < 2000). Logo, isso nos levar ao que seria uma **Avaliação Primitiva**, que é no qual as expressões algébricas somadas com as anotações sobre como avaliar cada expressão.

## Otimização de Consultas:

Logo, uma sequência de operações **primitivas** possui um dos papéis de avaliar uma consulta, e essa sequência consiste em um **plano de execução de consulta** ou **plano de avaliação de consulta**.

O plano de avaliação é diferente do que seria o custo (desempenho), pois é responsabilidade do sistema encontrar o plano mais eficiente do qual pode ser resumida na otimização, trazendo assim a consulta mais eficiente. E a mesma é seguida nesses passos aqui:

1. Encontrar as expressões equivalente (o mesmo resultado) a álgebra relacional desejada na qual diminua o acesso ao disco.
2. Selecionar uma estratégia detalhada para o processamento da consulta. Nesse caso estamos falando da escolha do algoritmo, índices, ...)
3. Estimar os custos dos planos de avaliação. Aqui já entra os otimizadores que fazem uso das informações estatísticas sobre as relações - sendo elas tamanho das relações, profundidade dos índices - para realizar uma melhor estimativa de custo de um plano.
4. Após a escolha do plano de avaliação, a consulta é executada de acordo com o mesmo e o resultado da consulta é produzido.

Vamos falar um pouco mais sobre o que seria as **Estimativas de Custos** para que possamos entender melhor essa parte e sobre como ocorre.

Sabemos que os otimizadores de consulta usam dados estatísticos armazenados no catálogo (metadados) do BD para verificar os custos de um plano. Esses dados são:

- Número de tuplas
- Número de blocos que contém as tuplas
- Tamanho em bytes das tuplas
- Número de valores distintos em uma relação para um atributo
- Cardinalidade da seleção do atributo da relação

Logo então sabemos que o catálogo armazena os dados sobre as relações, além das informações sobre os índices existentes. Logo, os dados estatísticos são usados a fim de estimar o tamanho do resultado e do custo para várias operações e algoritmos.

As atualizações das estatísticas ocorrem normalmente na carga do sistema. Essas informações não são completamente precisas e precisam ser avaliadas para saber se as estimativas oferecem estatísticas suficientemente precisas para fornecerem os custos coerentes relativos aos diferentes planos. Essa carga de atualizações/modificações no BD acaba gerando um Overhead significativo.

Em relação as **Medidas de Custo de uma Consulta** e o custo da avaliação para uma consulta onde ser medido por meio de vários recursos diferentes, sendo eles por exemplo o acesso ao disco, tempo de CPU, custo de comunicação e entre outros. Para os grandes BD, o número de transferência de blocos do disco são os mais relevantes devido a lentidão na sua velocidade em relação a memória.

Quando falando do segundo passo em relação a uma boa estratégia para o processamento da consulta, nesse caso estamos falando da **Operação de Seleção**. Podemos utilizar a estratégia de varredura, que nesse caso temos dois tipos:

- Busca Linear: cada bloco de arquivo é varrido e todos os registros são testados para verificar se satisfazem a condição de seleção.
- Busca Binária: o arquivo deve estar ordenado em um atributo e a condição de seleção é uma comparação de igualdade no atributo.
  O tipo de busca linear pode pareces ineficiente em muitos casos, mas ele sempre pode ser aplicado a qualquer arquivo indiferentemente da ordem do arquivo ou da disponibilidade de índice.
- A escolha de uma estratégia para avaliar uma operação depende de: Tamanho de cada relação e Distribuição de valores dentro das colunas.
- Procurando conseguir uma estratégia baseada em informação confiável, os sistemas de BD podem armazenar estatísticas para cada relação.
- As estatísticas podem:
  - Permitir estimar os tamanhos dos resultados de várias operações.
  - Possibilitar estimar os custos para executar estas operações
  - Útil quando vários índices estão disponíveis para ajudar no processamento de uma consulta.
  - A presença de índices tem influência significativa na escolha de uma estratégia de processamento de consulta.
  - A estrutura de um índice também é chamada de Path (caminho de acesso aos dados). Há a possibilidade de overhead aos blocos que contém o índice.

É possível processar consultas que envolvam:

1. Seleções simples por meio da execução de:
   1. Busca Linear
   2. Busca Binária
   3. Uso de Índice
2. Seleções complexas (computando uniões e interseções de seleções simples)
3. Ordenar relações maiores que a memória usando o algoritmo de merge-sort externo.
4. Consultas com junção natural podem ser processadas de vários modos, de acordo com a disponibilidade de índices e da forma de armazenamento física nas relações.

Um exemplo para cada caso:

- Se caso o resultado da junção é quase tão grande quanto o produto cartesiano das duas relações, uma estimativa de junção de laço aninhado de bloco pode ser vantajoso.

- Se os índices estão disponíveis, a junção de laço aninhado indexada pode ser usada.
- Se as relações estão classificadas, uma merge-junção pode ser desejável
    - Pode ser vantajoso ordenar uma relação antes de calcular a junção para uso da merge-junção
    - Também pode ser vantajoso calcular um índice temporário com o propósito exclusivo de permitir a utilização de uma estratégia de junção mais eficiente.
- O algoritmo de hash-junção particiona as relações em vários pedações, de forma que cada pedaço caiba na memória (particionamento feito por uma função de hash sobre os atributos da junção, para que se possa fazer independentemente a junção de pares correspondentes de partições).

Por exemplo o Merge-Sort:

1. Lê o primeiro bloco para a memória principal
2. Escolhe a primeira tupla do primeiro bloco e escreve em um arquivo temporário
3. Apaga a tupla lida do primeiro bloco e vai para a próxima tupla (ou novo bloco)
4. Repete o processo até todos os blocos estarem vazios
   Isso gera o resulta de uma **relação classificada**.

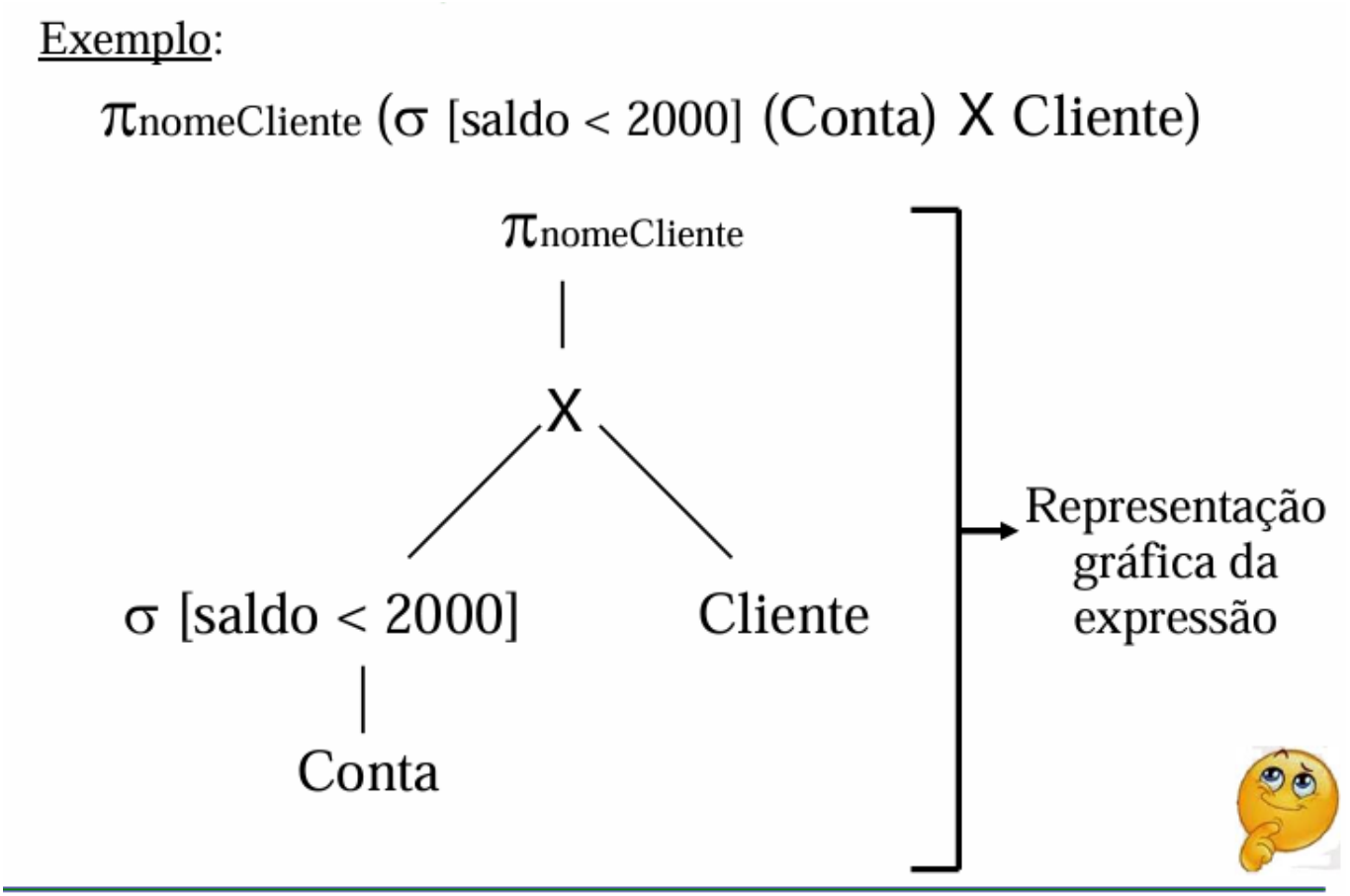
A classificação dos dados tem um papel importante em BD:

- As consultas podem solicitar que seus resultados sejam apresentados ordenadamente.
- Diversas das operações relacionais podem ser implementadas eficazmente, se as relações de entrada forem primeiramente classificadas.
- Classificação externa é a classificação de relações que não cabem completamente na memória principal.

As avalições de expressão com operações múltiplas podem ser avaliadas da seguinte maneira:

- Uma operação por vez, em uma ordem apropriada, gerando um resultado (materializado, as vezes a única possibilidade) para cada avaliação em uma relação temporária para ser usada na sequência.
- Avaliar várias operações simultaneamente em uma pipeline (alternativo), com os resultados de uma operação sendo passados para a próxima, sem a necessidade de relações temporárias.

Observe o seguinte exemplo:




Analisando o exemplo acima, teremos:

- Inicia no nível mais baixo da árvore;
- Resultados das operações mais baixas são **armazenadas em relações temporárias** que serão usadas na execução dos níveis mais altos;
- Cria-se uma nova relação temporária na execução de uma nova operação (nível acima);
- Executando a operação da raiz da árvore têm-se o resultado final da expressão desejada.

O custo da materialização deve considerar também o tempo da escrita no disco dos resultados intermediários (relações temporárias).

Os custos dessas operações podem ser reduzidas pela combinação de várias operações relacionais em uma Pipeline de operações, em que os resultados de uma operação são passados diretamente para a próxima operação, sem a necessidade de criação das relações temporárias.

- **Pipeline dirigido por demanda**: as solicitações de tuplas são operações repetidas no topo da pipeline. É a maneira mais utilizada devido a facilidade de implementação, logo DEMANDA = puxão para uma árvore de operações.

- **Pipeline dirigido pelo produtor**: as operações não esperam solicitações para produzir tuplas, elas geram tuplas "ansiosamente" até encher seu buffer. De maneira contínua as tuplas vão sendo removidas, após serem usadas, enchendo o buffer novamente até a geração de todas as tuplas. Logo PRODUTOR = empurrão para uma árvore de operações.

As **Expressões Equivalentes** são aquelas aonde temos expressões que possuem diversas formas e geram o mesmo resultado, e também que possuem diferentes custos de avaliação, e nisso geral as expressões equivalentes ou alternativas.

A **Função do Otimizador** é propor um plano de avaliação da consulta que gere o mesmo resultado da expressão fornecida (planos alternativos), e encontrar uma maneira menos onerosa de gerar o resultado desejado (plano menos caro).

Para a geração das expressões são feitas com os seguintes passos:

- Geração das expressões que são logicamente equivalentes a expressão solicitada.
- Escrever as expressões resultantes de maneiras alternativas para gerar planos de avaliação alternativos

Nesse processo todo temos também a **Regra de Equivalência**. Ela pode transformar uma expressão em outra e preservar a equivalência, que no qual significa que as relações geradas pelas duas expressões tem o mesmo conjunto de atributos e contém o mesmo conjunto de tuplas, embora seus atributos possam estar ordenados de forma diferente.

- O Otimizador usa as regras de equivalência para transformar expressões em outras logicamente equivalentes.
- As regras identificam somente as equivalências das expressões, sem dizer qual é a melhor em termos de custo (onerosidade).
- Regras de equivalência múltipla podem ser usadas em uma consulta ou em partes dela.
- O conjunto de regras de equivalência é dito mínimo se nenhuma regra pode ser derivada de qualquer combinação das outras.
- É possível eliminar vários atributos de um esquema por meio da projeção. Dessa forma, os únicos atributos a serem mantidos são aqueles que irão aparecer no resultado final da consulta e os que são necessários para processar as informações subsequentes. Reduzindo as colunas necessárias para a manipulação da consulta infere logo na diminuição ao custo da operação.

A **Ordenação de Junções** é importante para reduzir o tamanho dos resultados intermediários, e apesar das expressões serem equivalente, seus custos podem diferir. A junção é Associativa nesse caso pois produz o mesmo resultado, como pode ver no exemplo abaixo:

$$(\text{relação1} \times \text{relação2}) \times \text{relação3} = \text{relação1} \times (\text{relação2} \times \text{relação3})$$

E a **Junção Natural** também é comutativa, dessa forma facilitando a modificação na ordem de apresentação dos atributos no resultado final. Assim, produzem o mesmo resultado apesar da ordem ser diferente.

Logo, visualize o exemplo abaixo utilizando esses conceitos de Associatividade e Comutativa:

```
Temos o seguinte exemplo:

-> π nomeCliente ((σ[cidade = "São Paulo"](Agencia)) X Conta X Cliente)

A junção de (Conta X Cliente) irá trazer uma relação grande pois conterá uma
tupla para cada conta. Dessa forma, podemos transformar essa sub-expressão em:

-> σ[cidade = "São Paulo"](Agencia) X Conta

Assim a junção será menor por conta de se tratar da junção dos dados somente
para as contas da cidade de SP. Dessa forma, o resultado final será:

-> π nomeCliente(((σ [cidade = "São Paulo](Agencia)) X Cliente) X Conta)
```

Dessa forma, podemos analisar da seguinte maneira:

- Faz primeiro a seleção depois a junção com conta.
- Não irá existir atributos em comum entre Agencia e Conta, logo irá produzir o produto cartesiano entre estas relações.
- Entretanto, a relação temporária provavelmente irá produzir um produto cartesiano grande, então essa estratégia será rejeitada pelo nosso SGBD.

Gerar as equivalências é um processo caro que consome espaço e tempo, e é por conta disso que possuímos as sub-expressões compartilhadas que reduz a necessidade de espaço significativamente e é muito usada em otimizadores de consultas.

A **Otimização Baseada em Custo** gera uma faixa de planos de avaliação a partir de uma determinada consulta.

- Utiliza das regras de equivalência para gerar novos planos e escolher entre eles o de menor custo.
- Sendo uma consulta complexa, a quantidade de planos de avaliação diferentes podem ser grandes.
- Não é necessário gerar todas as expressões equivalentes a uma determinada consulta,

Com o uso de técnicas de propagação dinâmica reduz significativamente o número de expressões examinadas (avaliadas). A ordem em que as tuplas são geradas pela junção são importantes para encontrar a melhor ordem global de junção, pois isso pode afetar o custo das junções posteriores.

Temos também a **Otimização Heurística** que faz o uso de heurísticas para reduzir a quantidade de planos de avaliação que são completamente examinados:

- Faz uma primeira suposição heurística para um "bom" plano, com estimativas sobre seus custos.
- Uma desvantagem da otimização baseada no custo seria o próprio custo da otimização.
- Reduzem significativamente o overhead da otimização de consultas
- Alguns sistemas usam somente a heurísticas, não usando a otimização baseada em custos.

# Linguagem Procedural (PL/SQL):

Vamos revisar algumas coisas em relação a Base de Dados e na programação em Banco de Dados:

- A chave estrangeira é definida na relação filho e a relação contendo o atributo referenciado é a relação pai.

Veja um exemplo:

```
CREATE TABLE CIDADE (
    idCidade number          NOT NULL,
    cidade varchar2(40)      NOT NULL,
    estado varchar2(2),
    CONSTRAINT CIDADE_PK PRIMARY KEY (idCidade),
    CONSTRAINT CIDADE_ESTADO_FK FOREIGN KEY (estado)
    REFERENCES ESTADO (sigla) ON DELETE CASCADE
);
```

- **REFERENCES** é a restrição que identifica a relação pai e seus atributos.
- **FOREIGN KEY** é a que identifica os atributos na relação filho.
- **ON DELETE CASCADE** identifica que as tuplas da relação filho também serão apagadas quando a tupla da relação pai for apagada.

As restrições podem variar conforme a necessidade de implementação nas tabelas relacionadas por uma chave estrangeira no MySQL, sendo assim:

- **CASCADE**: Atualizar ou excluir os registros da tabela filha automaticamente, ao atualizar ou excluir uma tupla da pai.
- **RESTRICT**: Rejeita a atualização ou exclusão de um registro da tabela pai, ao atualizar ou excluir o registro da tabela pai
- **SET NULL**: Define como NULL o valor do campo na tabela filha, ao atualizar ou excluir o registro da tabela pai.
- **NO ACTION**: Equivale a opção RESTRICT mas a verificação de integridade referencial é executada após a tentativa de alterar a tabela. É a opção padrão se nenhuma opção for definida na criação de chave estrangeira.
- **SET DEFAULT**: Define um valor padrão para coluna da tabela filha, aplicado quando um registro da tabela pai for atualizado ou excluído.

Sabemos também que a restrição CHECK especifica uma condição que deve ser satisfeita para cada tupla da relação, estando no nível de atributo ou da relação. Um único atributo pode possuir várias restrições CHECK, não havendo limite no número destas restrições que podem ser definidas em um atributo.

A restrição CHECK pode usar as mesmas construções condicionais elaboradas para consultas SELECT, entretanto há algumas exceções:

- Não é permitida referências a pseudocolunas: CURRVAL, NEXTVAL, LEVEL e ROWNUM;
- Não é permitida chamadas as funções SYSDATE, UID, USER e USERENV;
- Não é permitido nas consultas que se referem a outros valores em outras tuplas.

Por conta da última exceções, há uma alternativa que permite trabalhar dessa forma. Ela é chamada de ASSERTION. Ela pode servir como alternativa quando temos que implementar regras de integradas mais complexas na qual precisamos trabalhar com outras tuplas e resultados de consultas, da qual o CHECK não consegue.

- Permite especificar restrições mais complexas:
  - Envolve a comparação entre tuplas diferentes
  - Analise comparativa com resultado de consultas (select)
- Sempre que uma tabela for alterada a asserção é verificada pelo SGBD
- Ações que violem a asserção são rejeitadas pelo SGBD.

Um exemplo: Suponha que estamos em uma empresa e precisas elaborar uma SQL na qual não pode existir o armazenamento de mais que um único empregado no cargo de presidente na empresa. Para isso funcionar, iremos usar a ASSERTION:

Exemplo 1

```
CREATE ASSERTION um_unico_presidente AS CHECK
(
    (
        SELECT COUNT(*)
        FROM EMPREGADO e
        WHERE e.cargo = 'presidente'
    ) <= 1
);
```

Exemplo 2

```
CREATE ASSERTION salario_menor_supervisor AS CHECK
(
    NOT EXISTS (
        SELECT *
        FROM EMPREGADO e, EMPREGADO s
        WHERE e.supervisor = s.matricula
        AND e.salario > s.salario
    )
);
```

No Exemplo 2 ele especifica uma consulta na asserção que irá recuperar os empregados que ganham mais que seu supervisor, e a clausula 'NOT EXISTS' irá garantir que nenhuma tupla seja recuperada. Caso a consulta não retornar vazio, então a restrição foi violada.

As restrições que são implementadas no SGBD por asserções promovem a sobrecarga (overhead) no SGBD, ainda mais quando muitos usuários podem atualizar a base de dados do SGBD.

Falando um pouco em relação ao contexto da programação, temos que:

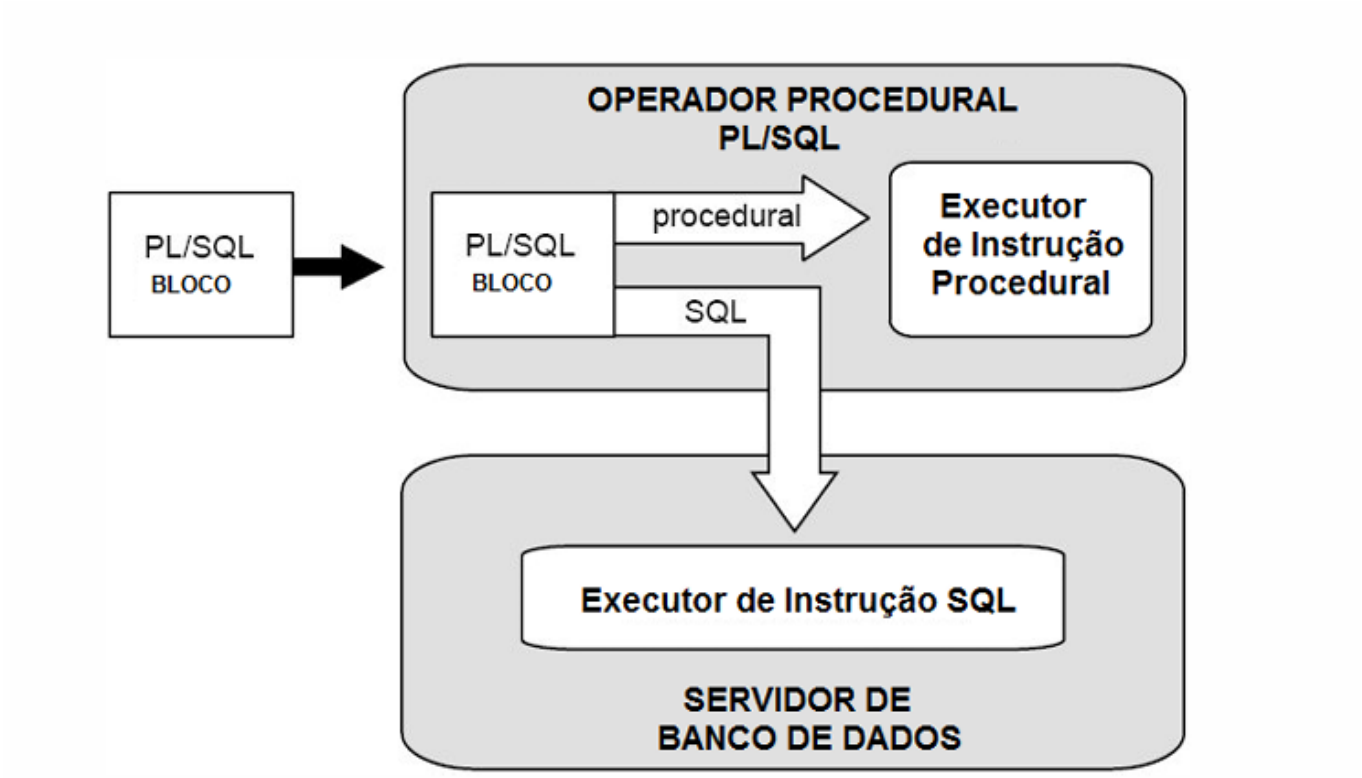
- 1GL -> Linguagem de máquina (binário em 0 e 1)
- 2GL -> Assembly, mnemônicos como LOAD e STORE
- 3GL -> Programação de alto nível, como C e Java
- 4GL -> Declarações que abstraem os algoritmos e estruturas, como SQL
- 5GL -> Programação visual

O PL/SQL combina a flexibilidade da SQL (4GL) com construções procedimentais do PL/SQL (3GL), assim podendo estender o SQL para:

- Variáveis e tipos
- Estruturas de controle

- procedimentos e funções
- tipos de objetos e métodos

A engine do operador PL/SQL funciona da seguinte forma:




E as suas vantagens são as seguintes:

- Suporte a SQL
- Suporta a programação OO
- Performance
- Produtividade
- Integração com Oracle
- Resolve "encruzilhadas" SQL
- Definição de regras de negócio não abrangidas pelo projeto relacional

Os seus recursos são:

- Estruturas em blocos
- Variáveis e tipos
- Tratamento de erros
- Estruturas de controle: Condicionais e Repetição
- Cursores

- Procedimentos e funções
- Pacotes
- Coleções
- Conceitos OO

Ele possui a estrutura de 3 blocos, por exemplo:

```
DELCARE
-- Variáveis, tipos, cursores, subprogramas, ...

BEGIN
-- instruções

EXCEPTION
-- tratamento de exceções

END;
```

Exemplo 3

```
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM ALUNOS;
    dbms_output.put_line('Qtd.Alunos =' || v_count);
END;
```

Há também o recurso do uso do %TYPE, do qual faz com que o SGBD descubra qual é o tipo daquele dado no BD.

Exemplo 4

```
DECLARE
    v_nome MORADOR.mnome%TYPE;
    v_cpf MORADOR.mcpf%TYPE;
BEGIN
    SELECT m.mnome, m.mcpf INTO v_nome, v_cpf
    FROM MORADOR m
    WHERE m.mcpf = 33125421039
    dbms_outuput.put_line('Nome ' || v_nome || ', CPF ' || v_cpf);
EXCEPTION
    WHEN NO_DATA FOUND THEN
```

```
        dbms_output.put_line('Morador não encontrado');
    WHEN TOO_MANY ROWS THEN
        dbms_output.put_line('Há mais de um morador com este CPF');
```

Também possui o recurso %ROWTYPE que faz com que SGBD descubra qual é o tipo de tuplas inteiras, por exemplo:

```
DECLARE
    v_vinculo SBD2_VINCULO_UNB%ROWTYPE

Que equivale a

DECLARE
    v_vinculo VARCHAR2(100),
    V_matricula NUMBER(9,0) ...
```

Vamos para um exemplo melhor:

Exemplo 5

```
DECLARE
    v_morador MORADOR%ROWTYPE
BEGIN
    SELECT * FROM v_morador
    FROM MORADOR M
    WHERE M.cpf = 33323112123;
    dbms_output.put_line('Nome ' || v_morador.mnome || ', CPF ' ||
v_morador.mcpf);
EXCEPTION
    WHEN NO_DATA FOUND THIS
        dbms_output.put_line('Morador não encontrado');
    WHEN TOO_MANY_ROWS THEN
        dbms_put_line('Há mais de um morador com este CPF');
```

Há também a presença de Estruturas de Controle de Fluxo Condicional:

- IF ... THEN ... END IF;
- IF ... THEN ... ELSE .. END IF;
- IF ... THEN .. ELSIF .. THEN ... ELSE ... END IF;
- CASE <variável>
    - WHEN valor THEN instruções WHEN ... THEN ... ELSE ... END CASE;

Vejamos um exemplo:

Exemplo 6

```
DECLARE
    v_count_turma NUMBER;
    v_count_aluno NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count_turma FROM SBD2_TURMA t
    WHERE t.codDisc = 'FGA0060' AND
    t.ano = EXTRACT (YEAR FROM SYSDATE) AND t.nroTurma = 1;

    IF v_count_turma = 0 THEN
        INSERT INTO SBD2_TURMA VALUES (1, EXTRACT(YEAR FROM SYSDATE),
'FGA0060', 31);
        dbms_output.put_line('Nova turma criada');

    SELECT COUNT(*) INTO v_count_aluno FROM SBD2_MATRICULA m
    WHERE m.codDisc = 'FGA0060' AND
    m.ano = EXTRACT (YEAR FROM SYSDATE) AND m.nroTurma = 1;

    IF v_count_aluno < 50 THEN
        INSERT INTO SBD2_MATRICULA (matricula, codDisc, ano, nroTruma, nota)
        VALUES (1, 'FGA0060', EXTRACT (YEAR FROM SYSDATE), 1, 0);
        dbms_output.put_line('Aluno Matriculado');
    ELSE dbms_output.put_line('Turma Lotada');
    END IF;
END;
```

Há também os loops das Estruturas de Repetição:

- Loop (instruções) EXIT WHEN (condicao de parada) END LOOP;
- WHILE (condicao de parada) LOOP (instrucoes) END LOOP;
- FOR (contador) IN (reverse) min...max LOOP (instruções) END LOOP;

Exemplo 7

```
DECLARE
    v_disciplina SBD2_TURMA.codDisc%TYPE;
    v_anoTurma SBD2_TURMA.ano%TYPE;
BEGIN
    v_disciplina := 'FGA0060';
    v_anoTurma := EXTRACT (YEAR FROM SYSDATE);

    FOR nroTurma IN 1..6 LOOP
```

```
        INSERT (nroTurma, v_anoTurma, v_disciplina, 31);
        dbms_output.put_line('Turma ' || nroTurma || ' criada.');
    END LOOP;
```

Agora vamos falar sobre **Cursores**.

- Na Área de Contexto
  - área de memória com informações de processamento de uma instrução
  - inclui conjunto ativo -> linhas retornadas por uma consulta
- Cursor
  - Handle para uma área de contexto (Não é uma variável de memória)
  - E possuí os tipos implícito e explícito

Os atributos do tipo Cursor são:

- FOUND
  - NULL se ainda não houve nenhum FETCH
  - true se o FETCH anterior retornou uma tupla
  - false caso contrário
- NOT FOUND: !FOUND
- ISOPEN
- ROWCOUNT -> Número de tuplas já lidas por FETCH

Exemplo de Cursor Explícito

```
DECLARE
    CURSOR c_alunos IS SELECT * FROM SBD2_ALUNO;
    v_alunos c_alunos%ROWTYPE;
BEGIN
    OPEN c_alunos;  -- Abre o cursos - executa a consulta

    LOOP
        FETCH c_alunos INTO v_alunos -- recupera tupla
        EXIT WHEN c_alunos%NOTFOUND;

        dbms_output.put_line('Matricula: ' || v_alunos.matricula || ' - Idade:
' || v_alunos.idade);
    END LOOP;

    CLOSE c_alunos -- fecha e libera memória
END;
```

No **Cursor Implícito** todas as instruções SQL são executadas dentro de uma área de contexto, então existe um cursor implícito que aponta para essa área de contexto -> cursos SQL. O PL/SQL implicitamente abre o cursos SQL, processa a instrução SQL e fecha o Cursor.

É bem utilizado para processas as seguintes instruções:

- INSERT
  - FOUND
    - TRUE -> Se o comando anterior alterou alguma trupla
    - FALSE -> caso contrário
  - NOT FOUND -> (!FOUND)
  - ROW COUNT -> Número de linhas alteradas pelo comando anterior
  - ISOPEN -> sempre FALSE, propriedade útil apenas para cursores explícitos
- UPDATE
- DELETE
- SELECT ... INTO

Exemplo de Cursor Implícito

```
DECLARE
    v_nota CONSTANT SBD2_MATRICULA.NOTA%TYPE := 5.0;

    BEGIN
        UPDATE SBD2_MATRICULA SET nota = v_nota
        WHERE nota > 3.0 AND nota < 6.0
        AND codDisc = 'FGA0060';

    IF SQL%FOUND -- cursor implícito associado ao UPDATE
    THEN dbms_output.put_line(SQL%ROWCOUNT || ' alunos tiveram a nota
alterada');
    ELSE dbms_output.put_line('Nenhum aluno teve a nota alterada');
    END IF;
END;
```

# Triggers:

Uma Trigger é um subprograma associado a uma tabela, view ou a um evento. Ela pode ser acionada uma vez quando um determinado evento ocorrer ou várias vezes para cada tupla afetada por uma instrução DML: Inserção, Exclusão e Alteração.

- Pode ser acionada após um determinado evento para registrar ou efetuar alguma atividade anterior e/ou posterior a este evento.

- Pode ser também acionada antes do evento para que possa prevenir operações indevidas ou ajustar novos dados para que estes estejam de acordo com a regra de negócio envolvida.

Existem alguns tipos de Triggers, por exemplo:

- Para as tabelas temos as triggers de DML (insert, update, delete) e as triggers de sistema (DDL e Logs)
- Para as visões temos as triggers de DML Instead-OF (insert, update, delete)

Para uma trigger ser disparada temos 3 instruções de disparo para ela, sendo: Insert, Update e Delete. Além disso, a trigger é temporal ou seja, ela possui o seu tempo para ser executada. Então ela pode ser BEFORE ou AFTER, e ocorrer por linha ou instrução (Nível).

O exemplo pode ser visto na tabela abaixo:

| **INSTRUÇÃO** | **:old** | **:new** |
|---|---|---|
| INSERT | NULL | valores que serão inseridos |
| UPDATE | valores antes da atualização | novos valores para a atualização |
| DELETE | valores antes da remoção | NULL |

Um exemplo de Trigger em código:

```
CREATE OR REPLACE TRIGGER NroDeAlunos
AFTER DELETE ON Matricula
FOR EACH ROW -- nível da linha
BEGIN
    UPDATE Turma
        SET NAlunos = NAlunos - 1
        WHERE Sigla = :old.Sigla AND
            Numero = :old.Numero;

EXCEPTION
....
END NrodeAlunos;
```

A Trigger é igual a Stored Procedures na questão de poderem ser armazenadas no banco de dados e também de serem compostas de instruções SQL e PL/SQL no Oracle. As demais instruções DDL que podem ser executadas sobre as triggers são: ALTER e DROP trigger. Porém, as triggers e as stored procedures se diferem em como são acionadas:

- Store Procedure é explicitamente acionada por um usuário, aplicação ou trigger.
- A Trigger é implicitamente disparada pelo servidor de banco de dados quando um determinado evento ocorre para gerar o evento.

Uma trigger não executa as seguintes coisas:

- Não é permitido os comandos transacionais dentro de uma trigger, como por exemplo: SET TRANSACTION, COMMIT, SAVEPOINT e ROLLBACK
- Não é permitido comandos DDL (CREATE, ALTER e DROP) pois eles disparam COMMIT automaticamente
- Não é permitido procedimentos/funções que impliquem em controle transacional
- A trigger fica sujeita apenas a transação definida na sessão em que o trigger foi disparado.
- A trigger não executa DML sobre tabelas mutantes. E uma tabela mutante é uma tabela que está sendo alterada por insert, update ou delete.

Exemplo em código de uma Tabela Mutante:

```
CREATE OR REPLACE TRIGGER Emp_count
AFTER DELETE ON Emp_tab
FOR EACH ROW
DECLARE
    nro INTEGER;
BEGIN
    SELECT COUNT(*) INTO nro FROM Emp_tab;
    DBMS_OUTPUT.PUT_LINE('There are now' || nro || 'employees.');
END;

-- Fora da Trigger
DELETE FROM Emp_tab WHERE Empno = 7499;
```

O motivo desse exemplo ser uma Tabela Mutante é por conta dos seguintes passos:

- A instrução DELETE é a que vai ocasionar o disparo da Trigger
- Dentro da trigger ocorre uma consulta na mesma tabela que está sendo modificada pela operação de exclusão.
- Dessa forma, a tabela está em transição onde as suas mudanças ainda não estão completamente consolidadas, assim não podendo ser lida ou escrita diretamente dentro de uma trigger que foi disparada por essa mesma operação.
- No Oracle ocorreria um erro informando que a tabela é mutante e impedindo a leitura da tabela em transição para evitar inconsistências de dados.

Para evitar a tabela mutante, um recurso seria a edição do conteúdo da variável :new em timing BEFORE. Vamos ver o exemplo em código.

```
CREATE OR REPLACE TRIGGER AcertaNota
BEFORE INSERT OR UPDATE ON Matricula
FOR EACH ROW
BEGIN
    -- Update Matricula SET nota = 5
    -- Daria erro de tabela mutante

    -- WHERE Sigla = :new.sigla;
    :new.nota := 5.0;
EXCEPTION
....
END AcertaNota;
```

Agora o exemplo de uma Trigger para a criação de Log:

```
CREATE OR REPLACE TRIGGER LogDisciplina
AFTER INSERT OR UPDATE OR DELETE ON Disciplina
FOR EACH ROW

DECLARE
    v_operacao CHAR;
BEGIN
    IF INSERTING THEN v_operacao := 'I';
        ELSIF UPDATING THEN v_operacao := 'U';
        ELSIF DELETING THEN v_operacao := 'D';
    END IF;

    INSERT INTO logTabelaDisciplina
        VALUES (USER, SYSDATE, v_operacao);
END LogDisciplina;
```

As Triggers de Sistemas são disparadas por:

- Instruções DDL
  - CREATE (BEFORE/AFTER)
  - ALTER (before/after)
  - DROP (BEFORE/AFTER)
  - DDL (before/after)
- Eventos do banco de dados
  - STARTUP - AFTER

    - SHUTDOWN - BEFORE
    - LOGON - AFTER
    - LOGOFF - BEFORE
    - SERVERERROR - AFTER
- Níveis são:
    - DATABASE
    - SCHEMA

Exemplo:

```
CREATE OR REPLACE TRIGGER TodosUsuarios
AFTER LOGON ON DATABASE -- ou ON SCHEMA
BEGIN
    INSERT INTO logUser VALUES (USER, 'Trigger TodosUsuarios');
END;
```

Os motivos para usar Trigger são:

- Restrições de consistência e validade que não possam ser implementadas com constraints - por exemplo envolvendo múltiplas tabelas
- Criar conteúdo de uma coluna derivado de outras
- Atualizar tabelas em função da atualização de uma determinada tabela
- Criar logs - segurança e auditoria

# Projeto Físico e Conceitos Relacionais:

Veremos alguns conceitos na Organização do Banco de Dados Relacional (BDR).

Uma observação mais detalhada sobre os conceitos e objetos organizados em uma abordagem lógica e física de um banco de dados relacional (BDR) permite a melhor compreensão de seu funcionamento.

Diferentes implementações disponíveis nos BDRs demonstra que o material está baseado na arquitetura Oracle, sendo muito similar aos outros BDRs.

Alguns Conceitos:

- Database: Conjunto de registros de dados dispostos em estrutura regular que possibilita o seu armazenamento organizado produzindo informação.
- Schema:
    - Representado por uma coleção de vários objetos de um ou mais usuários do BDR (tabela, sequências, índices, ...)

    - Associado a uma base de dados (database) na razão de vários esquemas para um BD.
- Tablespace:
    - É onde o BDR é armazenado, logicamente, em um ou mais tablespaces, que por sua vez, armazena, fisicamente, em recurso magnético e não volátil, um ou mias arquivos para cada tablespace guardar os dados de forma organizada.

O Banco de Dados possui uma estrutura física e uma lógica:

- Estrutura Lógica: representam os componentes que podem ser vistos no BD (tabelas, índices, etc)
- Estrutura Física: representam os recursos de armazenamento usados internamente pelo BD, por exemplo os arquivos físicos. As estruturas lógicas podem ser idênticas, independente do hardware e sistema operacional
    - O ORACLE mantém separadas essas estruturas.
- Instância: É composto pelas estruturas de memória e pelos processos de segundo plano (background)
    - No Oracle as estruturas de memória são: SGA (System Global Area) e PGA (Program Global Area)
    - Os principais processos de segundo plano:
        - Database Writter (DBW0)
        - Log Writer (LGWR)
        - System Monitor (SMON)
        - Process Monitor (PMON)
        - Checkpoint Process (CKPT)

O armazenamento físico em um banco de dados Oracle consiste em três tipos de arquivos, que são:

- Data Files -> Arquivos que armazenam os dados no BD
    - Guardam todo os dados no Banco de dados relacional, armazenando dados (tabelas), índices, áreas temporárias, dicionário de dados, objetos do usuário
    - Cada BDR é formado por um ou mais Data Files
    - Cada Data File está associado a uma única tablespace
    - Uma tablespace pode consistir de um ou mais Data Files.
- Control Files -> Arquivos de controle do BD que incluem os metadados
    - Mantém as informações sobre a estrutura física do BDR
    - Cada BD Oracle tem pelo menos um Control File

    - As informações armazenadas no Control File permitem conservar e verificar a integridade de um BDR. E é recomendado manter múltiplas cópias dos Control Files.
    - O Control File contém o nome do BD e o timestamp de sua criação, bem como os nomes e a localização de todos os Data Files e Redo Log Files.
- Redo Log Files -> Arquivos que registram as alterações no BD, sendo utilizados nas operações de recuperação (recovery)
    - Armazena também o log de todas as transações do BD
    - Oracle possui dois ou mais Redo Log Files, sendo gravados de forma crítica
    - Pode-se obter informações sobre os dados alterados, e são fundamentais nas operações de recuperação
    - Em caso de falhas do BDR, eles são usados para recuperar as transações na sua ordem apropriada
    - É aconselhável manter múltiplas cópias dos Redo Log Files em dispositivos diferentes

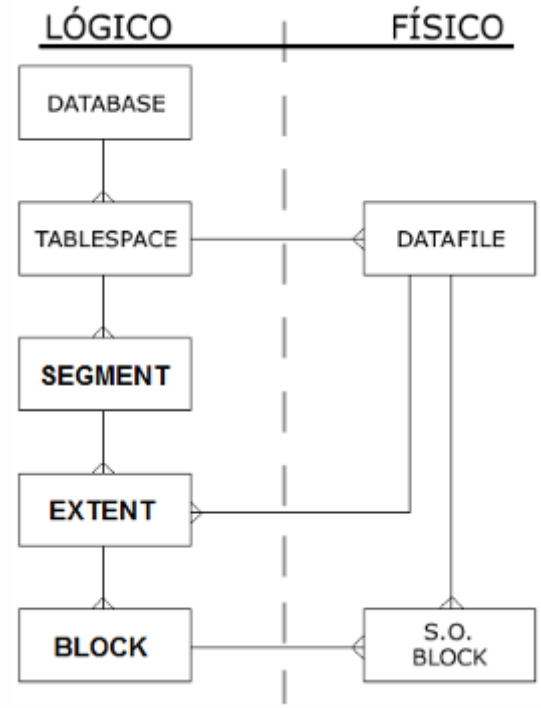
O Banco de Dados armazena os dados logicamente em tablespaces e fisicamente em data files (arquivos de dados). Embora que seja estreito o inter-relacionamento entre arquivos de dados e tablespaces, existem algumas diferenças significativas entre eles:

- BD consiste em uma ou mais unidades de armazenamento lógicas (tablespaces) que guardam todos os seus dados
- Cada tablespace de um BD consiste em um ou mais arquivos de dados, que são estruturas físicas compatíveis com o sistema operacional onde o BD é executado.
- Os dados de um BD são armazenados coletivamente nos datafiles que constituem cada tablespace do BD

O BD é um conjunto de arquivos de dados (Data files), então o entendimento sobre como estes arquivos são agrupados é relevante a compreensão do funcionamento de um BD, e esse agrupamento acontece por meio do objeto chamado tablespace.

- Sabemos que um tablespace é constituído por um ou mais arquivos, e que um arquivo de dados só pode pertencer a um único tablespace e um tablespace só pertencerá a um único BD.

O Oracle divide o BD em unidades menores a partir do tablespace para gerenciar, armazenar e recuperar os dados de maneira eficiente. As estruturas lógicas consistem em:




O Segmento (Segmento) é:

- Objetos menores que ocupam espaço em um BD, sendo chamados também de Segmentos de Dados. Eles armazenam as linhas de dados associadas as tabelas ou clusters.
- Um segmento é composto por um conjunto de Extents (extensões) alocados para uma estrutura lógica. Cada segmento possui um cabeçalho que serve como um diretório de espaço para o segmento.
- Um tablespace pode consistir em um ou mais segmentos. E existem vários tipos de segmentos como de tabelas, índices, LOB e outros.

A Extensão (Extent) é:

- Espaço usado por um segmento em um tablespace, sendo o próximo nível de agrupamento lógico no BD. Quando um segmento é criado, ele adquire pelo menos uma extensão inicial que armazenará os dados até não ter mais nenhum espaço livre.
- Existe em apenas um datafile.
- Quando um objeto de BD é expandido, os espaço adicionado ao objeto é alocado como uma extensão. Depois que as extensões existem não pode mais conter novos dados, sendo necessário ao Segmento obter uma extensão para as novas inserções de dados feitas no BD.
- Um segmento é composto por extensões, que são conjuntos contíguos de um ou mais blocos de BD (blocks);
- O processo de extensão continuará continuamente até que não haja mais espaço disponível nos datafiles do tablespace ou até que um número máximo interno de extensões

por segmento seja atingido. Quando os dados adicionais são incluídos no segmento, este se estende obtendo uma nova extensão.

O gerenciamento das Extensões podem ser feitos de duas formas no Oracles: localmente, onde o tablespace gerencia seus espaços e pelo dicionário de dados na qual novas alocações de extensões são atualizadas no dicionário de dados.

Um Bloco (Block) é:

- menor unidade física transportável entre arquivos de dados e memória. O bloco é a menor estrutura de armazenamento do BD e corresponde a um número específico de bytes.
- O tamanho de um bloco é normalmente um múltiplo do tamanho de um bloco do sistema operacional podendo ser baseado no parâmetro DB_BLOCK_SIZE e determinado quando o BD Oracle é criado. O bloco Oracle consiste em um ou mais blocos do S.O. e seu tamanho é definido na criação do tablespace.
- Pode ser composto por: Cabeçalho, Espalho Livre e Dados.

Os três níveis de um Projeto de Banco de Dados possui aspectos e características importantes aos objetivos de cada um em relação aos momentos e necessidades do projeto. Veremos ele:

- Conceitual -> Abstração e aprendizagem sobre o problema que será resolvido. É geralmente usado o ME-R.
- Lógico -> Melhor organização respeitando aspectos físicos da tecnologia de implementação do modelo Conceitual.
- Físico -> Implementação coerente com a realidade de uso do Banco de dados e seus usuários.

É importante haver o Projeto Físico do Banco de Dados do BD da sua empresa. As atividades do Projeto Físico a ser realizado na implementação física do banco de dados não pode consistir em executar instruções provenientes ao modelo lógico. São diversas ferramentas que geral o script bem organizado e coerente com este nível do Projeto do BD (nível lógico sendo mapeado ao nível físico).

Caso não tenha a análise (ou modelagem) para real implementação física do BD o projeto poderá ser totalmente comprometido, assim sendo frágil. Portanto, o DBA deverá:

- Analisar o modelo lógico
- Apurar a realidade de uso das estruturas previstas no modelo
- Constatar características importantes e diferentes entre cada uma dessas estruturas, se for necessário
- Ajustar ou transformar a implementação de possíveis estruturas para atender, adequadamente, aos seus objetivos

- Só serão implementá-lo fisicamente

Portanto, um Projeto de BD é formado pelos seguintes aspectos do:

- Projeto Lógico de Banco de dados
  - Nível conceitual (ME-R e DE-R)
  - Nível Lógico (DLD)
  - Nível Físico (Scripts)
- Projeto Físico de Banco de Dados (Implementação física do BD)
  - Organização Lógica dos Dados
  - Organização Física dos Dados

O objetivo do Projeto Físico de BD é especificar o Modelo Físico de Banco de dados, levando em consideração o Modelo Lógico de dados homologado, assim exigindo informações sobre volumes, acessos e a necessidade de disponibilidade, visando assim garantir uma implementação com desempenho coerente as expectativas, além de assegurar aspectos como padronização, portabilidade, disponibilidade e capacidade de recuperação tempestiva dos dados.

O Modelo Lógico de Dados é proveniente do Modelo Conceitual de Dados no Projeto de BD (processo de mapeamento). A Modelagem de dados que visa representar o Negócio (escopo do projeto), identificando e organizando suas possíveis estruturas (tabelas) que representam o que ocorre no mundo real (conceitual), mas sintonizada com a tecnologia de BD que será adotada para implementação do Projeto de BD.

- É baseado no ME-R, elaborado no nível conceitual, representando os esquemas de suas relações, relacionamentos existentes e restrições que deverão ser implementadas no nível físico do BD.

Existe o instrumento no qual o desenvolvedor especifica como as funções do sistema irão utilizar o Modelo de Dados que será implementado, o seu nome é Mapa de Acesso Lógico (MAL):

- Registra as informações sobre os acessos
- Pode apresentar dados sobre a periodicidade que determinada funcionalidade deverá ser executada.
- Tipo de processamento em lote (batch) ou on-line.

O Mapa de Acesso Lógico (MAL) possui vários modelos com tipos de representações diferentes, seja no nível de funções, transações e programas. A gente a utiliza a melhor representação para a solução que se esteja buscando, independente do modelo adotado.

- Artefato importante como parte integrante da documentação exigida pela metodologia de desenvolvimento de sistemas.
- A prática nos mostra que é melhor trabalhar no nível de macro-especificações de programas, já apresentado as instruções do BD que serão executadas.
- Não é necessário um MAL para todos os programas, mas se recomenda que sejam estabelecidos critérios para selecionar aqueles considerados mais críticos em relação ao desempenho, requisitos do negócio e entre outros aspectos.

Podem haver a necessidade de outras informações além do Modelo Lógico e o MAL. Essas informações são necessárias para que o DBA possa trabalhar no Projeto Físico do BD, e essas informações irão consistir em seu relatório de apoio.

- É necessário que isole as informações para identificar as tabelas foco.
- Verificar os volumes e particularidades de acesso, analisando a modelagem e o MAL para escolher as tabelas que serão consideradas mais críticas e que irão compor este relatório.

Essas informações são fundamentais ao Projeto Físico e deverão ser preenchidas pela equipe de desenvolvimento. Várias delas podem ser derivadas das Entradas anteriores (Modelo Lógico e MAL):

- A necessidade da cópia de segurança (backup) por período.
- "Janelas" para execução de utilitários como reorganização.
- Concorrência entre processamento batch e on line
- Quantidade de usuários concorrentes
- Necessidade de expurgo

E algumas dessas informações são aferidas por estimativa com base nos levantamentos da equipe de análise. Além das entradas esse processo ainda possui:

- Itens regulatórios (normas, padrões, regras, políticas de desenvolvimento, estrutura tecnologia do SGBD)
- Itens de suporte (profissionais, SGBD)
- Saídas (Modelo Físico)
- Atividades a serem realizadas.

E as atividades realizadas pelo DBA deverão analisar o modelo lógico, o MAL e o relatório de apoio. E com isso tudo torna-se possível:

- Definir os índices a serem criados
- Definir os tipos de colunas adequados
- Identificar tabelas com grandes volumes de dados para eventual particionamento

- Identificar hierarquias de generalização/especialização (Entidades super-tipos e subtipos) para decidir o número de tabelas a ser implementado.
- Descrição de Domínios discretos para eventual implementação de Constraints
- Decidir se a Integridade Referencial será garantida pelo SGBD ou pela aplicação
- Definir parâmetros para garantir a disponibilidade dos dados conforme requisitos (Nível de Lock - bloqueios, acessos concorrentes, ...)

# Transações e Concorrência:

Operações que formam uma única unidade lógica de trabalho são chamadas de **Transações**. Se trata de uma unidade de execução que acessa e manipula os dados no Banco de Dados, que no caso é executada geralmente por um programa (instrução) elaborado com linguagem de manipulação de dados em alto nível e linguagem de programação. A transição consiste em todas as operações a serem executadas a partir do começo até o fim da transação.

Essas transações possuem suas propriedades que são conhecidas como A.C.I.D. Essas propriedades são importantes para assegurar a integridade dos dados, portanto vamos falar sobre elas.

- Atomicidade (A): todas as operações da transação são refletidas corretamente no BD ou nenhuma será.
- Consistência (C): a execução de uma transação isolada preserva a consistência do BD (situação inicial e final)
- Isolamento (I): Cada transação não toma conhecimento de outras transações concorrentes.
- Durabilidade (D): Depois da transação completar-se com sucesso, as mudanças que ela faz no BD, persistem até mesmo se houver falhas no sistema.

Bem, as transações possuem também os seus estados:

- ATIVA: permanece neste estado enquanto está sendo executada a transação
- Efetivação Parcial: após a execução da última declaração.
- Falha: descobre-se que a execução não poderá ser efetivada
- Abortada: transação desfeita, restabelecendo o BD ao início (para ser executada novamente deverá ser gerada uma nova transação)
- Efetivada: após a conclusão com sucesso.

A transação é somente efetivada (commited) caso ela entre no estado de Efetivada, e a transação é abortada (rolled back) caso ela entre no estado de Abortada. Portanto, uma transação é concluída se estiver no estado de Efetivada ou Abortada.

Quando a transação está no estado de Falha, que é ocorrida por conta de um erro de lógica, hardware ou de leitura, a mesma não pode prosseguir com sua execução normal, devendo ser desfeita. Dessa forma, ela passa para o estado de Abortada e pode reiniciar a transação (somente possível para erros de hardware ou software) ou encerrar a transação (erro lógico). A operação que reinicia uma transação consiste na criação de uma nova transação para ser processada, nunca é a mesma.

Além disso, temos também as **Transações Concorrentes** que se trata de agilizar a realização da tarefa desejada, porém também traz diversas complicações em relação a consistência dos dados no BD. É muito mais fácil manter as execuções das transações sequencias, entretanto duas possibilidades básicas nos levam a utilizar a concorrência:

- Operação da CPU e as E/S podem ser feitos em paralelo.
- Mistura de transações simultâneas no sistema (curtas, longas).
- As ações executadas podem
    - Acessar diferentes partes do BD
    - Reduzir atrasos imprevisíveis
    - Diminuir o tempo médio de resposta
    - Reduzir a ociosidade da CPU, discos e outros dispositivos

Temos que o processamento concorrente compromete a propriedade de Consistência do BD, portanto para que a concorrência possa acontecer sem comprometer a propriedade C, é analisada a escala de execução (schedules) das transações envolvidas. Execuções de diversas transações em modo concorrente pode prejudicar a consistência do BD, dessa forma é necessário ao sistema controlar a interação entre as transações que são executadas simultaneamente. Dessa forma, é necessário garantir o que chamamos de **Serialização**, que no qual são usados vários esquemas de controles sobre a concorrência, sendo eles Protocolos de Bloqueio, Ordenação por Timestemp, Ténicas de Validação e Esquemas de Multiversão, e iremos falar sobre cada um deles aqui.

## Protocolos de Bloqueio:

É um conjunto de regras que estabelece quando uma transação pode bloquear e desbloquear um item de dados no SGBD. Ele obriga que o acesso a um item de dados seja mutuamente exclusivo, ou seja, enquanto uma transação acessa um item de dados, nenhuma outra transação pode modificá-lo.

Há vamos modos de bloqueios de dados, e os dois mais significativos são:

- Compartilhado (S): se uma transação obteve um bloqueio compartilhado sobre um item de dados, ela só poderá ler o item mas não escrevê-lo.

- Exclusivo (X): se uma transação obteve o bloqueio exclusivo sobre um item de dados, então ela só poderá ler e escrever neste item

Exemplo:

```
Suponha que A possui R$ 400,00 e B R$ 200,00. E que a transferência de 50
reais da conta A para a B serão acessadas pelas respectivas transações (T1 e
T2)

T1: bloqueia-X (A):
    leia(A);
    A = A - 50;
    escreva(A);
    desbloqueia(A);
    bloqueia-X(B);
    leia(B);
    B = B + 50;
    escreva(B);
    desbloqueia(B);

-- T2 responsável para apresentar como resultado a soma do saldo das contas
T2: bloqueia-S (A):
    leia(A);
    desbloqueia(A);
    bloqueia-S(B);
    leia(B);
    desbloqueia(B);
    apresente(A+B);

-- OBS: Os desbloqueios podem ser solicitados ao final da transação, evitando
o acesso e uso de informações momentaneamente inconsistentes
```

## Deadlock:

São problemas inerentes ao bloqueio que garante a consistência do BD. São ditos como impasse. Um exemplo é termos duas transações T3 e T4, e T4 depende que T3 libere B porém T3 também está esperando que T4 libere A; Por conta disso, a situação não chega em lugar nenhum, e as transações não podem processar na sua forma normal, assim ocorrendo um impasse.

O uso de bloqueios pode causar situações indesejadas, dessa forma a melhor opção é utilizar o bloqueio e o desbloqueio tão logo seja possível para que possa evitar essas situações, assim reduz os possíveis deadlocks. E caso ocorre um deadlock, o sistema irá desfazer a transação liberando um item de dado.

O processo para que deadlocks não possam acontecer são chamadas de Inanição, ou seja Concedendo Bloqueio. Para esse caso, verifica se não existe nenhuma outra transação de bloqueio sobre um item de dado, cujo modo de bloqueio seja conflitante. Caso não existe nenhuma outra transação que esteja esperando um bloqueio sobre este item de dado e que tenha feita sua solicitação anteriormente.

## Protocolo de Bloqueio em Duas Fases:

Esse protocolo permite que uma transação bloqueie um item de dado somente após desbloqueá-lo, portanto temos as seguintes fases:

- Fase de Expansão: transação pode obter bloqueios, mas não pode desbloquear nenhum;
- Fase de Encolhimento: transação pode liberar bloqueios, mas não consegue obter nenhum outro bloqueio

O protocolo garante a serialização mas não nos livra do Deadlock. E com a falta de informação a respeito do acesso que é necessário sobre um item de dado, este protocolo será necessário e suficiente para garantir a serialização.

Temos então que a **Conversão de Bloqueios** consiste em um refinamento do protocolo básico de bloqueio de duas fases, assim temos:

- promover um bloqueio compartilhado para exclusivo na fase de expansão. ou
- rebaixar um bloqueio exclusivo para compartilhado no encolhimento.

Lembrar que todos os bloqueios são desbloqueados após uma transação ser concluída.

Mas também temos **Variações do bloqueio em duas fases**. As duas variações de bloqueio são usadas extensivamente em sistemas de BD comerciais:

- Severo: em adição as características deste bloqueio, ele também exige que os bloqueios exclusivos sejam mantidos até a transação ser efetivada (encerrada).
- Rigoroso: exigem que todos os bloqueios sejam mantidos até que a transação seja encerrada (efetivada ou abortada).

E para obter a serialização de conflito, sem precisar utilizar o protocolo de bloqueio, serão necessárias informações adicionais sobre a transação ou a imposição de alguma estrutura ou ordenação sobre o conjunto de itens de dados do SGBD. Existem diversos modelos, que

precisam de quantidades diferentes de informações, de acordo com as características que este modelo irá proporcionar.

## Ordenação por Timestamp:

Aqui utiliza a seleção de uma ordenação entre transações em andamento. Entre alguns métodos, o mais utilizado é este, na qual cada transação recebe a associação de um único timestamp fixo (criado pelo SGBD) antes que esta transação inicie sua execução.

As duas formas para esta implementação seria usar a hora do sistema (relógio) ou usar um contador lógico automático.

O timestamp das transações determinam a ordem da serialização, sendo necessário garantir uma equivalência de escala serial na sua execução. A ordenação é realizada por meio da seleção na ordem de execução baseada no valor do timestamp entre pares de transação.

Temos que:

- Um único timestamp é associado a cada transação
- Execução da transação com menor timestamp
- Reversão da transação, sempre que a ordem for violada
- Uma reversão, feita pelo controle de concorrência, recebe um novo timestamp e é reiniciada
- Protocolo resistente a deadlock, pois uma transação nunca espera.

Exemplo:

```
-- Uma transação recebe um timestamp antes da sua primeira instrução
-- T5 tem timestamp menor que T6

T5: leia(A);

    ...executando T6
    leia(A);
    A = A - 50;
    escreva(A);

    ...executando T5
    leia(B)

    ...executando T6
```

```
    leia(B);

    ...executando T5
    apresente(A+B);

    ...executando T6
    B = B + 50;
    escreva(B);
    apresente(A+B);
```

## Técnicas de Validação:

É adequado em situações que a maioria das transações sejam **somente de leitura**, com baixas taxas de conflito entre elas.

As 3 fases (leitura, validação, escrita) => 3 timestamps (1 transação), logo

- Associação de um único timestamp para cada transação
- A ordem da serialização é determinada pelo timestamp associado
- Necessidade de passar pelo teste de validação para completar-se ou será revertida até seu estado inicial
- Uma transação nunca atrasa neste esquema

## Agregação de Itens de Dados:

Em certos momentos, como em bloqueios, pode ser vantajoso agrupas alguns itens de dados. Estes agrupamentos são tratados como itens de dados agregados que resultam em múltiplos níveis de granularidade.

Podem se estabelecer em tamanhos diferentes para cada item de dados; hierarquia entre eles e os menores são aninhados aos maiores.

Pode ser representado graficamente como uma árvore.

- Bloqueios concedidos no sentido da raiz para as folhas.
- Liberação dos bloqueios no sentido contrário (folhas para a raiz)
- Protocolo garante a serialização
- Possibilidade de ocorrência de deadlock

## Controle de Concorrência Multiversão:

Os métodos vistos acima sempre atrasam ou abordam uma transação, garantindo a serialização e controlando a concorrência. Este método tem por base a criação de uma nova versão do item de dado que será escrito:

- Quando uma leitura é solicitada, o sistema seleciona uma das versões para realizar a leitura
- Cruciar para o desempenho que uma transação possa determinar fácil e rapidamente qual versão de item que será lida.
- O esquema de controle de concorrência garante que a versão a ser lida será serializada por meio do timestamp
- Uma operação de leitura sempre obtém sucesso, porém na multiversão com ordenação por timestamp ela pode resultar em rollback.
- Na multiversão, com bloqueio de duas fases, uma operação escrita pode ter que aguardar para efetivar um bloqueio ou mesmo um deadlock pode ocorrer.

Propriedades indesejáveis:

- Operação de leitura faz dois acessos a disco (própria leitura e a atualização dos R-timestamp)
- conflitos de transações são resolvidos por rollback (não usa tempo de espera)

## Mecanismo de Manipulação de Deadlock:

Vários protocolos de bloqueio ocasionam o deadlock, mas algumas abordagens podem prevenir, como por exemplo ciclos de esperar ou todos os bloqueios serem solicitados juntos.

Um modo de evitar deadlock é usar a preempção e o rollback de transação. Para controlar a preempção:

- marca-se um único timestamp para cada transação
- timestamp auxilia na decisão da transação esperar ou ser revertida (desfeita)
- para transações revertidas o timestamp é mantido, quando for reiniciada

Há dois esquemas de prevenção de deadlock baseados na preempão são:

- Esperar Morrer -> Técnica de não-preempção onde se TA for mais antigo que TB, senão caso contrário TA será revertido (morto).
- Ferir Esperar -> Tem por base uma técnica de preempção, sendo a contrapartida do Esperar-Morrer. A técnica é que TA é mais novo que TB, senão caso contrário TB seria desfeito (ferida).

Uma alternativa seria o uso do método de detecção de deadlock e recuperação, na qual elabora um gráfico de espera, na qual na existência de um ciclo (geralmente lembra um losango) indicaria um deadlock. E assim usaria um algoritmo de detecção de deadlock e o sistema irá se recuperar, revertendo uma ou mais transações para romper o deadlock.

Após a detecção de um deadlock o sistema precisa recuperar-se (revertendo uma ou mais transações). Portanto ele seleciona a transação e checa o seguinte:

- Tempo da transação (realizada e a processar)
- Quantos itens de dados a transação já usou
- Quantos itens ela ainda usará para se completar
- Quantas transações serão envolvidas no rollback (Utilizar reversão total ou semente até a quebra do deadlock)

## Operação de Inserção e Remoção:

Inserção se trata de:

- Necessita do bloqueio exclusivo sobre a nova tupla;
- Possível ocorrência do fenômeno fantasma

Remoção se trata de:

- Necessita de bloqueio exclusivo sobre a tupla a ser excluída

O Fenômeno do fantasma se trata de quando uma inserção entra em conflito com uma consulta, mesmo que cada uma delas não acessem uma tupla em comum. Sua solução seria o bloqueio de índice, que na qual garante que todas as transações conflitantes estejam em conflito por itens de dados reais, e não por fantasmas.

Para estruturas de dados especiais podem ser desenvolvidas técnicas especiais de controle de concorrência:

- Normalmente são aplicadas sobre árvores B+ visando o aumento da concorrência
- Essas técnicas permitem acessos não-seriados sobre as árvores B+
- Estrutura muito adequada, com acesso garantido ao SGBD, de forma seriada.