- **Banco de Dados Distribuído:** coleção de BDs interligados logicamente por meio de uma rede de computadores.
- **SGBDD:** possibilita a gerência de BDDs, tornando esta gerência transparente para o usuário.
- Num SBDD, os dados são armazenados em nós (locais) diversos da rede; esta base não são meros arquivos, nem há apenas um BD centralizado em um dos nós dessa rede.
- Quanto menor a latência da rede, melhor para o BDD.

CARACTERÍSTICAS DOS SBDDS (nossa, que sigla enorme)

- **TRANSPARÊNCIA:** provê independência de dados. Assim, usuários enxergam uma imagem logicamente integrada do BD, embora este seja, na realidade, fisicamente disperso.
  A transparência pode ser de rede, de replicação ou de fragmentação.
  A transparência de fragmentação pode ser horizontal (seleção, as linhas do BD) ou vertical (projeção, as colunas do BD), ou, óbvio, híbrida.
- **CONFIABILIDADE:** os SBDDS evitam pontos únicos de falha porque trabalham com componentes replicados.
- **AUMENTO DE DESEMPENHO:** os dados ficam próximos de seus pontos de uso. No entanto, fragmentação e replicação exigem suporte. Permite paralelismo entre (inter) e dentro (intra) de consultas.
  Para tratar das atualizações de dados replicados, é necessário implementar controle de concorrência e protocolos de finalização (*commit*) destribuídos.

QUESTÕES DE SGBDD

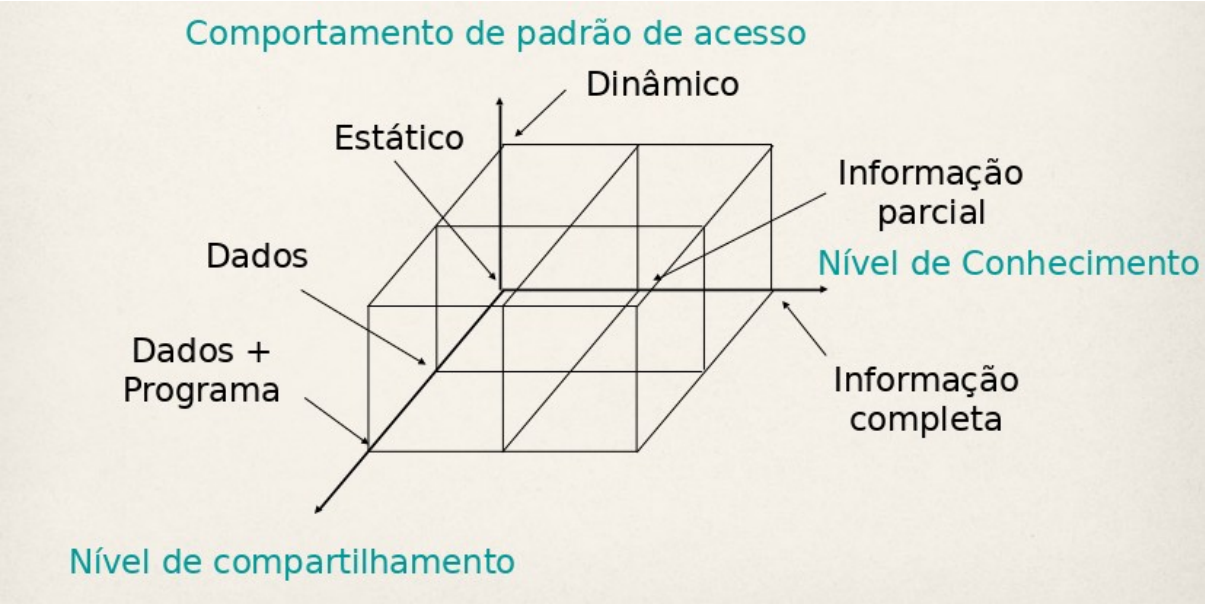
- **Projeto de distribuição:** como distribuir a base? Deve ser replicada ou não? Como acontecerá a gerência de catálogo?
- **Processamento de consulta:** conversão de transações do usuário em instruções de manipulação de dados; problemas de otimização; como saber se é melhor transmitir os dados ou processá-los localmente (problema NP-Difícil)?
  - Transmissão e processamento de dados são os maiores custos de um BDD.
- **Controle de concorrência:** como sincronizar acessos concorrentes, e deixá-los consistentes e isolados? Como gerenciar *deadlocks*?
- **Confiabilidade:** como tornar o BD tolerante a falhas, permitindo atomicidade e durabilidade?

PROCESSAMENTO DISTRIBUÍDO DE CONSULTAS

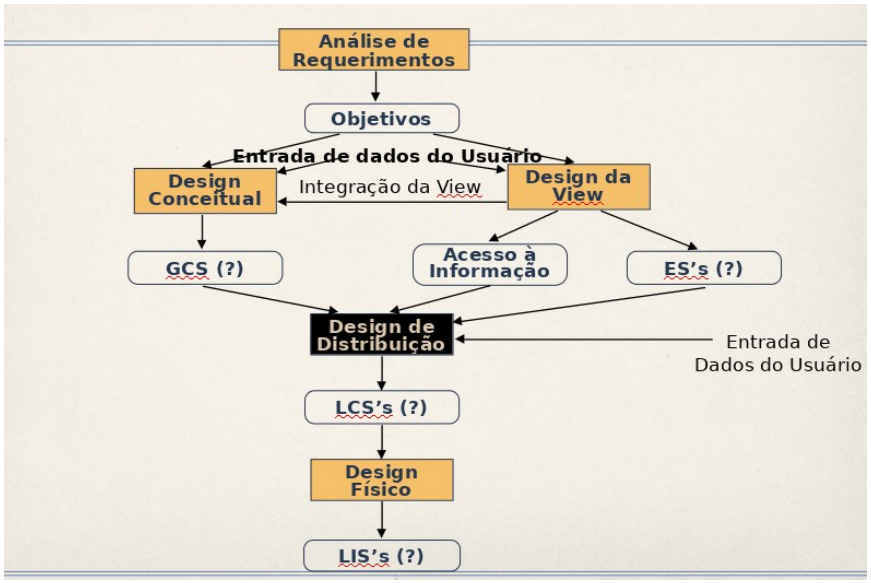
- Traduz uma consulta em linguagem de alto nível sobre uma base distribuída, vista como única pelo usuário. Esta tradução deve ser sempre correta e o plano gerado, ótimo.
- O processamento possui quatro fases:
  - **Decomposição da consulta**
  - **Localização dos dados**
  - **Otimização global**
    - Gera o plano de execução ótimo a partir da consulta fragmentada, usando técnicas heurísticas e sistemáticas de otimização. Este cálculo de custo deve levar em conta a movimentação dos dados pelos nós.
  - **Otimização local**

DESIGN DE S(G)BDDS (de novo essa sigla gigante)

- É importante designar onde serão guardados os dados e os programas ao longo dos pontos da rede, bem como, muitas vezes, fazer o *design* desta própria rede.
- A localização da aplicação envolve tanto a localização do software do SGBD, quanto das aplicações que se utilizam destas bases.



- O design de distribuição pode se dar de duas formas distintas:
  - Bottom-up (baixo para cima): quando o banco de dados já existe em alguns *sites*.
  - Top-down (cima pra baixo): na maioria das vezes, é usado para *designs* feitos do zero, em geral, usado em sistemas homogêneos.



- **GCS:** esquema conceitual global;
- **ES:** esquema externo;
- **LCS:** esquema conceitual local;
- **LIS:** esquema físico (interno) local;
- **LES:** esquema externo lógico;
- **GES:** esquema global externo.

PROBLEMAS DO DESIGN DE DISTRIBUIÇÃO

- Por que sequer fragmentar? Como fazê-lo? O quanto fragmentar?
- Como testar se está correto?
- Como alocar? ~~se fosse em C, bastaria usar malloc, ou chamar o dj alok~~
- Quais os requerimentos da informação?

UNIDADES DE DISTRIBUIÇÃO (FRAGMENTOS) RAZOÁVEIS

- Relações
  - *Views* são apenas sub-conjuntos locais das relações
  - Comunicação extra
- Fragmentos de relações (sub-relações)
  - Execução concorrente de transações que acessam diferentes porções da relação.
  - *Views* que não conseguirem ser definidas em um fragmento requererão processamento extra.
  - Controle semântico de dados mais difícil (forçar integridade, principalmente.)
- O grau de fragmentação é sempre um valor definido entre finitas possibilidades de combinação entre, no mínimo, o número de tuplas/atributos e, no máximo, o número de relações.

EXATIDÃO (CORRECTNESS) DA FRAGMENTAÇÃO

- **Completude (Completeness):** A decomposição de uma relação R em fragmentos R_1, R_2, R_n só é completa se, e somente se, cada item de dado em R puder ser encontrado em R_i.
- **Reconstrução (Reconstruction):** Se a relação R for fragmentada em R_1, R_2, R_n, deve haver um operador relacional qualquer O, de forma que R = (O) R_i, sendo R_i todos os fragmentos.
- **Disjunção (Disjointness):** Se a relação R é fragmentada em R_1, R_2, R_n, e há um item de dado *d* em algum destes fragmentos, então *d* não deve estar em nenhum outro fragmento senão naquele em que originalmente se encontra.

OPÇÕES DE ALOCAÇÃO

- **Não-replicado:** o fragmento é PARTICIONADO, com cada fragmento residindo em um só *site.*
- **Replicado:**
  - **Parcialmente replicado:** cada fragmento em alguns dos *sites*.
  - **Totalmente replicado:** cada fragmento em cada *site*.
- Se (número de *queries* somente-leitura)/(número de *queries* de *update*) < 1, a replicação é vantajosa; em outras situações, pode causar problemas.

	Replicação Total	Replicação Parcial	Particionamento
PROCESSAMENTO DE QUERY	Fácil	← Mesma Dificuldade →	
GERÊNCIA DE DIRETÓRIOS	Fácil ou não-existente	← Mesma Dificuldade →	
CONTROLE DE CONCORRÊNCIA	Moderado	Difícil	Fácil
Confiabilidade	Muito alto	Alto	Baixo
Realidade	Possível Aplicação.	Realístico	Possível Aplicação.

FRAGMENTAÇÃO HORIZONTAL

- Fragmenta por tuplas, isto é, por linhas (e não por colunas.)
- Antes de realizá-la, é importante saber, da aplicação, informações qualitativas (auxiliam no processo de fragmentação) e quantitativas (auxiliam no processo de alocação.)
- A informação qualitativa fundamental consiste nos predicados utilizados em *queries* do usuário.

- Se for impossível determinar este número com exatidão, sabe-se que, em geral, 20% das principais *queries* do usuário são responsáveis por 80% do processamento de dados do BDD, então basta analisar estas principais *queries* (regra 80/20.)

- **PREDICADOS SIMPLES:** é composto por um atributo da relação, um operador (=, <, !=, <=, >=, >), e um valor. NOME = “Venceslau Pietro Pietra”, SALÁRIO < 2000 são predicados simples.
- **PREDICADO MINTERMO:** é um conjunto possível de mintermos a partir de um conjunto de predicados simples. Mintermos são a junção de um ou mais predicados junto ao operador lógico E. O predicado pode estar na sua forma lógica natural, ou pode ser sua versão negada. Por exemplo, pode haver algo como NOME = “Gesonel o mestre dos disfarces” E SALÁRIO < 5, ou algo como NOME = “Gesonel o mestre dos disfarces” E SALÁRIO >= 5 (que é a negação do predicado simples anterior SALÁRIO < 5.)
- A informação quantitativa depende de dois conjuntos de dados.
  - Seletividade do mintermo: o número de tuplas (linhas) da relação que seriam acessadas de acordo com um predicado mintermo.
  - Frequência de acesso: frequência com a qual *queries* acessam os dados em um determinado período de tempo. A frequência de acesso de mintermos pode ser determinada a partir da frequência de suas respectivas *queries*.
- **Fragmentação horizontal primária:** é uma SELEÇÃO aplicada em uma relação R qualquer. A fórmula de seleção pode ser tanto um predicado simples quanto um predicado mintermo.
  - Há tantas fragmentações horizontais possíveis quanto o número de predicados mintermos possíveis da relação.
  - O conjunto de fragmentos horizontais também pode ser chamado de fragmentos mintermos.
  - Os predicados simples devem ter COMPLETUDE.
    - Um conjunto de predicados *P* simples é dito COMPLETO se, e somente se, o acesso às linhas (tuplas) dos fragmentos de mintermos definidos em *P* requer que duas tuplas do mesmo fragmento de mintermo tenham a mesma probabilidade de serem acessadas por qualquer aplicação.
      - Assim, se as *queries* comuns são buscar um país e um certo salário, um conjunto de predicados terá completude se ele atender a estes dois critérios simultaneamente.
    - Um conjunto de predicados *P* simples será mínimo se for relevante à determinação da fragmentação. Se todos os predicados do conjunto *P* forem relevantes, *P* é mínimo.

- **FRAGMENTAÇÃO HORIZONTAL DERIVADA:** se há uma tabela A ligada em alguma outra B, e A é dona da ligação, então a relação B pode ser divida em diferentes fragmentos, sendo estes fragmentos equijoins de B com A_1, A_2, A_n.

FRAGMENTAÇÃO VERTICAL

- Mais difícil que a horizontal, pois existem mais alternativas.
- Se encaixa mais naturalmente em *designs* Top-down.
- É possível fazer ser de:
  - **Agrupamento:** atributos (colunas) viram fragmentos.
    - Pode-se juntar alguns fragmentos até que certo critério tenha sido satisfeito.
    - Produz fragmentos cujos atributos acabam sendo justapostos.
  - **Divisão (Splitting):** relação vira fragmento.
    - Decide-se por partições que sejam benéficas, embasando-se no padrão de acesso das *queries*.
    - Produz fragmentos não justapostos, o que auxilia na disjunção, parte fundamental da exatidão (*correctness*) da fragmentação. Naturalmente, apenas atributos que não sejam chaves primárias não são justapostos.
- As informações requeridas para realizar uma fragmentação vertical são:
  - Afinidades de atributos, que avalia quão próximos os atributos estão. Geralmente, é avaliada vendo o conjunto de atributos mais usados entre as *queries* mais usadas. A regra 80/20 é útil aqui. Por exemplo, se em muitas *queries* os atributos “nome” e “idade” são usados, então eles são afins.
    - Para se conseguir um valor numérico, multiplica-se a quantidade de vezes que os atributos são acessados na *query* pela quantidade de vezes que esta *query* é executada (frequência de execução das *queries*.)
- **Algoritmo de Bond Energy**
  - É utilizado para realizar a fragmentação.
  - Para isso, ele agrupa os atributos. Aqueles que possuírem maior valor de afinidade são colocados juntos em um *cluster*; os que possuírem menores valores de afinidade também são juntos, em outro *cluster*.

- **Exatidão da fragmentação vertical:**
  - **Completude:** Uma relação R, com atributos A que geraram fragmentos verticais R_n, é completo se a união dos atributos de todos estes fragmentos retorno os atributos A originais da relação.
  - **Reconstrução:** A relação R deverá poder ser reconstruída a partir de uma operação JOIN com todos os fragmentos R_i.
  - **Disjunção:** São desimportantes, uma vez que as chaves não são consideradas como justapostas.

FRAGMENTAÇÃO HÍBRIDA

- Aplica-se alguma das duas técnicas anteriores primeiro, e depois outra, e assim sucessivamente, até que sejam satisfeitos os requisitos da aplicação.

ALOCAÇÃO

- Saber a distribuição ótima dos fragmentos dentre os nós da rede é um problema. Há duas formas geralmente pensadas para a estratégia de alocação:
  - **Custo mínimo:** se resume a buscar o menor valor ao somar o custo de manter cada fragmento em um nó, o custo de fazer *queries* de cada fragmento neste nó, o custo de atualizar o fragmento em todos os nós que estiver, e o custo da comunicação.
  - **Performance:** a estratégia de alocação é configurada de forma a proporcionar uma métrica de performance, geralmente buscando minimizar o tempo de resposta e maximizar a produtividade em cada nó.
- Informações requeridas para traçar a melhor estratégia de alocação:
  - **Informação da base de dados:** é preciso conhecer tanto o tamanho em *bytes* da tupla de cada fragmento, quanto o número de tuplas (linhas) do fragmento que deverão ser acessadas para que se possa processar uma determinada *query*.
  - **Informação da aplicação:** é importante conhecer o número de acessos de leitura que uma *query* faz a um fragmento, bem como o número de acessos de atualização (update.)
  - **Informação da rede de comunicação:** deve-se conhecer o custo por unidade para guardar um fragmento em um nó, além do custo por unidade de processamento deste nó.
  - **Informação do sistema computacional:** necessário saber a banda, a latência, e o *overhead* de comunicação.

INTEGRAÇÃO DE DADOS

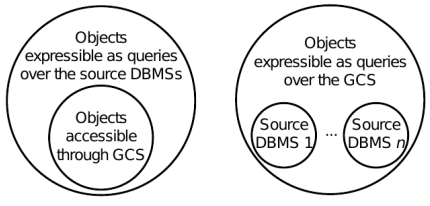
- Lida com a maneira que se integra o LCS (Esquema Conceitual Local) ao GCS (Esquema Conceitual Global.) Neste caso, o processo de *design* é bottom-up.
- Há duas maneiras de se integrar:
  - **Física:** os BDs-fontes são integrados, e o BD integrado final é materializado. São as chamadas *Data Warehouse*.
    - A integração é auxiliada pelas ferramentas de extração-transformação-carregamento (*extract-transform-load*, ETL.) Elas permitem extrair os dados da fonte, transformá-los para que combinem com o GCS, e também carregá-los (materializá-los.)
  - **Lógica:** o Esquema Conceitual Global é virtual, e não materializado. É o que acontece nas Enterprise Information Integration (EII.)
    - A integração é auxiliada pela Enterprise Application Integration (EAI), permitindo a troca de dados entre aplicações, fazendo funções similares de transformação, embora não materialize os dados completamente.

DESIGN BOTTOM-UP

- Se o GCS é definido primeiro, e então os LCSs são mapeados para este esquema, semelhantemente ao que ocorre em *data warehouses*.
- Se o GCS é definido como parte de integração dos LCSs, ele gera o GCS e então mapeia os LCSs para o GCS.

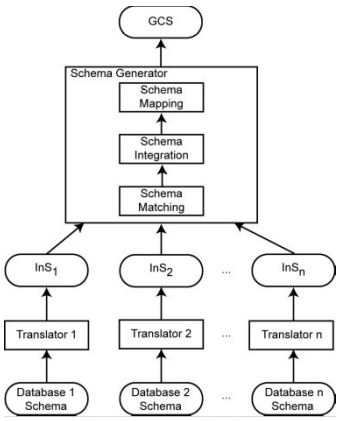
RELAÇÃO ENTRE GCS/LCS

- **Global-as-view:** o GCS é presumido como uma série de *views* sobre os LCSs.
- **Local-as-view:** presume que exista a definição do GCS, e cada LCS é tratada como uma definição de *view* sobre ele.



(a) GAV (b) LAV

PROCESSO DE INTEGRAÇÃO DO BANCO DE DADOS



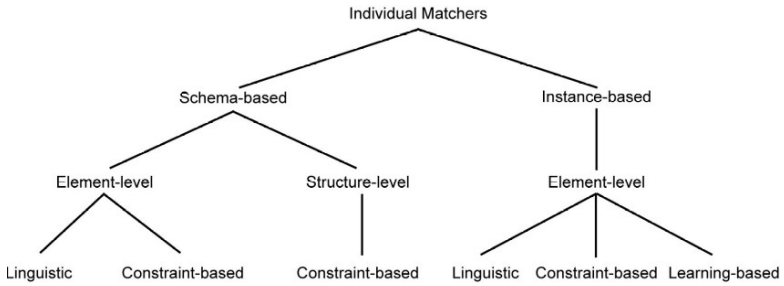
PROBLEMAS NA INTEGRAÇÃO

- **Tradução de esquema:** os esquemas componentes da base de dados são traduzidas para uma representação canônica intermediária comum.
  - O modelo canônico de dados são:
    - Relacionais;
    - Entidade-relacionais;
    - Orientado a objetos;
    - Orientado a grafos.
- **Geração de esquema:** esquemas intermediários são usados para criar um GCS. Passa por alguns processos:
  - **Equiparação de esquemas:** encontrar as correspondências entre múltiplos esquemas.
  - **Integração de esquemas:** criação do GCS usando as correspondências.
  - **Mapeamento de esquemas:** como mapear os dados dos bancos locais para o GCS.

EQUIPARAÇÃO DE ESQUEMAS

- **Heterogeneidade de esquema:** os dados entre as equiparações podem ser diferentes, seja por:
  - **Heterogeneidade estrutural:** pode ser por conta de conflito de tipos, de dependências, de chaves, ou de comportamento.
  - **Heterogeneidade semântica:** mais importante e difícil de lidar; ocorre por conta de diferentes ontologias, do uso impreciso de palavras, ou de saber mapear sinônimos, homônimos e hiperônimos.

- **Outros problemas:** informações insuficientes sobre o esquema ou a instância; indisponibilidade de documentação do esquema; subjetividade de equiparação.
- **Problemas que afetam a equiparação de esquemas:** equiparação de esquema vs equiparação de instância; equiparação de elemento versus equiparação a nível estrutural; cardinalidade de equiparação.



EQUIPARAÇÃO LINGUÍSTICA DE ESQUEMA

- Utiliza-se dos nomes dos elementos, e de outras informações textuais (descrições, anotações.)
- Pode se valer de fontes externas (dicionários.)
- O elemento1 no esquema1 é similar ao elemento2 no esquema2 se o predicado *p* mantém algum valor de similaridade *s*.
- **Em nível de esquema:**
  - Lida com nomes dos elementos dos esquemas.
  - Lida com casos como sinônimos, homônimos, hiperônimos, similaridades de tipos de dados.
- **Em nível de instância:**
  - Foca em técnicas de resgate de informações (frequência de palavras, termos-chaves), deduzindo similaridades destes.

EQUIPARADORES LINGUÍSTICOS

- Utiliza um conjunto de regras linguísticas terminológicas.
- Regras básicas podem ser feitas à mão ou descobertas em fontes externas.
- Assim, o predicado *p* e o valor de similaridade *s* podem ser especificados, se feitos à mão, ou descobertos, se computados ou previamente especificados por um especialista.
- Exemplos de regras:
  - nomesEmMaiusculo ASSEMELHA nomesEmMinúsculo, 100%
  - nomesEmMaiusculo ASSEMELHA nomesCapitais, 100%

- Descobridores automáticos de similaridades:
  - **Afixos:** prefixos e sufixos comuns entre os textos do nome de dois elementos.
  - **N-Grams:** comparar quantas *substrings* de tamanho *N* são comuns entre dois nomes.
  - **Editar distância:** número de modificações de caracteres (adição, exclusão, inserção) que precisam ser feitas para converter uma *string* em outra
  - **Código de Soundex:** similaridade fonética entre nomes baseados nos seus códigos *soundex*.
  - **Observar os tipos de dados:** pode sugerir relações mais fortes do que usando os métodos anteriores, ou ao menos para diferenciar *strings* de mesmo valor.

EQUIPARADORES BASEADOS EM *CONSTRAINTS*

- Pode-se usar *constraints* para se equiparar dados, como a faixa de valores, as informações do tipo de dados etc.

EQUIPARADORES ESTRUTURAIS BASEADOS EM *CONSTRAINTS*

- Se dois elementos do esquema são estruturalmente similares, então há uma grande chance de que eles representem o mesmo conceito.
- Similaridades estruturais: mesmas propriedades (atributos), similaridade de vizinhos (usando grafos, sendo cada nó um conceito.)

EQUIPARADORES BASEADOS EM *MACHINE-LEARNING*

- Usa *machine learning* para determinar equiparações de esquema.
- Classifica conceitos de vários esquemas em classes, de acordo com a similaridade destes conceitos. Os que caírem em classes iguais representam conceitos similares.
- Similaridade é definida de acordo com características da instância dos dados.
- A classificação é “aprendida” a partir de um conjunto de treino.

INTEGRAÇÃO DE ESQUEMAS

- Usa as correspondências encontradas anteriormente para a criação do GCS. Geralmente, é um processo manual, mas algumas regras podem ajudar.
- Pode ser:
  - **Binária:** manipula somente dois esquemas por vez. A complexidade é mais simples, sendo 2^M, *M* o número de esquemas.
    - Em escada: realiza a manipulação e a integração de forma gradual (*stepwise*), gerando uma árvore em formato de escada. Ou seja, são criados esquemas intermediários para manipulação com os esquemas subsequentes.
    - Balanceada: realiza a manipulação de forma binária balanceada, gerando uma árvore binária balanceada. São criados esquemas intermediários entre dois pares de esquemas, e, depois, são estes esquemas intermediários os utilizados para seguir com a manipulação.
  - **N-ária:** manipula mais de um esquema por vez. A complexidade é maior quanto maior for N, sendo N^M, M o número de esquemas.
    - One-shot: ocorre quando todos os esquemas são manipulados de uma só vez, produzindo o GCS após somente uma integração. É difícil de automatizar, e é o método de maior complexidade.
    - Iterativa: é mais geral e oferece maior flexibilidade. Na realidade, integrações binárias são apenas um tipo especial de manipulação iterativa.