Opções de Persistência

- Tratamos aqui apenas dos aspectos arquiteturais da persistência
  - Não tratamos do mapeamento de projetos OO para esquemas de BD relacionais
- Três grandes pontos devem ser tratados com respeito à persistência de objetos:
  - Escolha da forma básica de persistência (na memória, em arquivos, usando um SGBD)
  - Escolha do paradigma de SGBD (se usar um SGBD)
  - Determinação da estratégia de interação entre a aplicação e os dados

Escolha da forma básica de persistência

- As alternativas básicas são:
  - Dados na memória
  - Dados em arquivos
  - Dados sob controle de um BD
- Muitas aplicações requerem necessariamente os serviços de um BD, mas algumas podem funcionar bem com uma alternativa mais simples
- Os seguintes pontos devem ser avaliados para tomar a decisão sobre a forma de persistência:

Persistência de dados

- Arquivos e SGBDs são fortes
- Ficar com os dados na memória requer um suporte especial de memória e fontes de alimentação para prevenir a perda de informação
- Frequente para software embarcado (em celulares, ...)

Custo de aquisição

- Um SGBD pode ser muito caro
  - Se houver um site license já em uso, o custo

incremental pode ser pequeno

- O hardware para manter dados na memória pode ser caro
- Arquivos nada custam pois são inerentemente suportados pelo sistema operacional

Custo total de posse (Total Cost of Ownership)

- Custos incluem a aquisição, treinamento, desenvolvimento, implantação, operação e manutenção
- Qualquer uma das 3 alternativas pode ganhar aqui
  - Exemplo: SGBD pode reduzir custos de desenvolvimento mas aumenta custos de operação (incluindo recursos humanos)
  - Exemplo: arquivos não têm custo de aquisição mas têm um alto custo de manutenção

Quantidade de dados

- SGBDs podem armazenar gigantescas quantidades de informação
- Recursos de hardware limitam a quantidade de dados que pode ser armazenada na memória
- Arquivos são limitados em tamanho pelo sistema operacional
  - Cuidado com sistemas que limitam arquivos a 32 bits (4 GBytes)

Desempenho

- Dados na memória são acessados mais rapidamente
- SGBDs têm algoritmos e estruturas de dados especiais para lidar com grandes quantidades de dados
- Arquivos podem ser rápidos se couberem na memória ou se forem eficientemente acessados seqüencial ou randomicamente

Extensibilidade

- SGBDs oferecem independência de dados de forma que o desenvolvedor pode se concentrar nos aspectos

lógicos dos dados, sem preocupação imediata com detalhes de implementação física

Acesso concorrente

- A granularidade de travamento pode afetar o desempenho
- Tipicamente, arquivos são travados por completo
  - Travamento de registros de arquivos requer muita programação
- A granularidade é melhor com dados na memória
- SGBDs podem ter boa granularidade (record locking) ou pior (page locking)
  - Além do mais, SGBDs podem escalonar travamentos e resolver conflitos automaticamente sem programação adicional

Recuperação de crash

- Arquivos: Arquivos de backup são necessários
- SGBDs: têm lógica sofisticada para recuperação de crash
- Memória: requer o uso de *shadow memory*

Integridade

- Apenas SGBDs oferecem o suporte a regras de integridade (definidas pelo programador)

Suporte a transações

- Apenas SGBDs oferecem suporte a transações

Distribuição

- Apenas (alguns) SGBDs oferecem suporte à distribuição de dados e vários sites

Linguagem de consulta

- SGBDs oferecem linguagens de consulta para facilitar o manuseio de dados
- Alguns produtos do mercado oferecem linguagens de

consulta simples para dados em arquivos

Segurança

- Arquivos podem ser protegidos pelo sistema operacional (mas sem sofisticação)
- SGBDs podem implementar segurança com passwords, views, etc.

Metadados

- SGBDs permitem a manipulação da estrutura dos dados (metadados) em tempo de execução

Um sumário segue:

	Dados na memória	Arquivos	SGBDs
Persistência de dados	Requer hardware especial	Bom suporte	Bom suporte
Custo de aquisição	Custo do hardware especial	Não há	Pode ser caro
Custo total de posse	Variável	Variável	Variável
Quantidade de dados	Limitado pelo hardware	Limitado pelo sistema operacional; A memória limita arquivos em cache	Essencialmente sem limite
Desempenho	Muito rápido	Rápido para acesso sequencial, certos acessos randômicos e para arquivos em cache	Rápido
Estensibilidade	Limitada	Limitada	Excelente
Acesso concorrente	Locking de objetos	Locking de arquivos	Locking de objetos ou de registros; Alguns SGBDs só têm locking de páginas
Recuperação de crash	Shadow memory	Arquivos de backup	Bom suporte
Integridade	Não há	Não há	Projetista pode especificar regras
Suporte a transações	Não há	Não há	Transações curtas
Distribuição	Não há	Não há	Às vezes
Linguagem de			

consulta	Não há	Parcial	Poderosa
Segurança	Não há	Proteção simples do sistema operacional	Pode ser simples ou sofisticado
Metadados	Não há	Não há	Sim

- Em suma, algumas aplicações naturalmente se beneficiam de cada alternativa de persistência:
  - Dados na memória
    - Dados para dispositivos eletrônicos (celulares, PDAs, ...)
    - Dados para smart cards, cartuchos de jogos
    - Dados para aplicações que não podem tolerar o custo ou falta de confiabilidade de um disco
  - Arquivos
    - Dados de grande volume, com pouca densidade de informação (archival files, registros históricos detalhados, logs, ...)
    - Dados crus que devem ser sumarizados num BD (exe.: aquisição de dados, telemetria, ...)
    - Quantidades modestas de dados com estrutura simples (quando não tem SGBD ou este seria complexo demais para a empresa manter)
    - Dados acessados sequencialmente
    - Dados que são lidos por inteiro na memória
  - SGBDs
    - Dados que requerem atualização com níveis finos de granularidade por múltiplos usuários
    - Dados que devem ser acessados por várias aplicações
    - Grandes quantidades de dados que devem ser eficientemente manipulados
    - Dados de longa duração e muito valiosos a uma organização
    - Dados que devem ser acessados com sofisticado controle de segurança
    - Dados sujeitos a análise sofisticada para o apoio à decisão
- Finalmente, observe que é comum utilizar mais do que uma única alternativa para dados diferentes

Escolha do paradigma de SGBD

- Se usar um SGBD, ainda deve-se escolher entre SGBDs relacionais, objeto-relacionais e orientados a objeto
- Os detalhes fogem do escopo desta disciplina
- Resumindo:
  - SGBDRs são maduros mas sofrem de um problema sério: impedance mismatch entre objetos e relações
    - As grandes desvantagens são:
      - Navegação lenta (usando joins de tabelas)
      - Protocolos de travamento inflexíveis (por serem automáticos)
      - Poucos tipos de dados
      - Tudo *tem* que ser uma tabela
  - SGBDOOs não são tão maduros mas não sofrem de impedance mismatch e apresentam recursos avançados (evolução de esquema, controle de versões, long transactions, notificações de mudanças, ...)
    - Oferecem navegação rápida mas ainda carecem até de uma teoria completa!
- Há ainda a alternativa de SGBDs Objeto-Relacionais que mantêm o paradigma relacional mas resolvem algumas questões (tais como melhor suporte a tipos de dados)

Determinação da estratégia de interação entre a aplicação e os dados

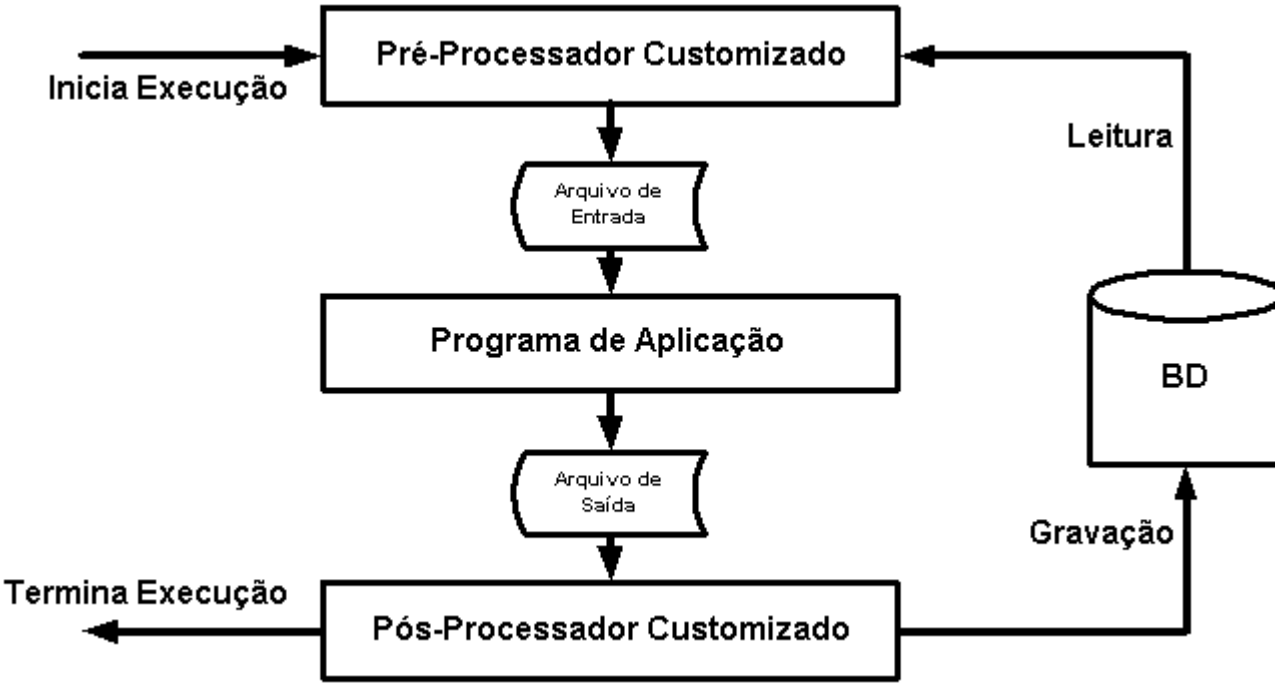
- Quais são as estratégias para conectar sua aplicação a um SGBD?

Pré-processador e pós-processador batch

- Em alguns casos, é melhor que a aplicação não acesse o BD
- Aplicações batch existentes podem fazer uso de um

BD de forma indireta

- Pode ser útil em alguns (raros?) casos em que se usa software legado ou sem código fonte dentro de uma aplicação (como subsistema)



Arquivos de script

- Para certas aplicações simples, basta executar um ou mais arquivos contendo comandos de uma linguagem de consulta (SQL, OQL, ...)
- Bom para certas situações tais como criar esquemas, popular um BD, prototipagem

Comandos embutidos da linguagem de manipulação de dados de um SGBDOO

- Neste caso, a aplicação acessa o BD diretamente
- Parece natural devido ao casamento entre a linguagem hospedeira (digamos Java) e um SGBDOO
- Cuidado! Ao proceder assim, o acoplamento com o SGBDOO fica muito forte e pode ser difícil mudar de SGBD posteriormente

Comandos SQL estáticos embutidos

- Código SQL estático é conhecido em tempo de compilação

- É relativamente simples de usar, mas:
  - Sem muitos cuidados (batching, etc.), pode resultar numa execução lenta
  - Acopla muito o código da aplicação ao esquema do BD
  - Resulta em código difícil de se ler
- Em suma, tenta-se desta forma resolver o impedance mismatch com código estático

Implementar uma API customizada de acesso aos dados

- Os acessos ao BD são encapsulados em poucos métodos da aplicação
- Ajuda a manter a consistência dos dados
- Ajuda a manter o escopo de transações
- Ajuda a não contaminar a aplicação com um monte de código de acesso ao BD
- Técnica muito usada

Métodos armazenados no BD

- Stored procedures
- Boa forma de reuso para várias aplicações
- Cuidado! Stored Procedures não são implementados de forma transportável pelos vários SGBDs do mercado
  - O padrão SQL não é obedecido pelos fabricantes

Linguagem de quarta geração

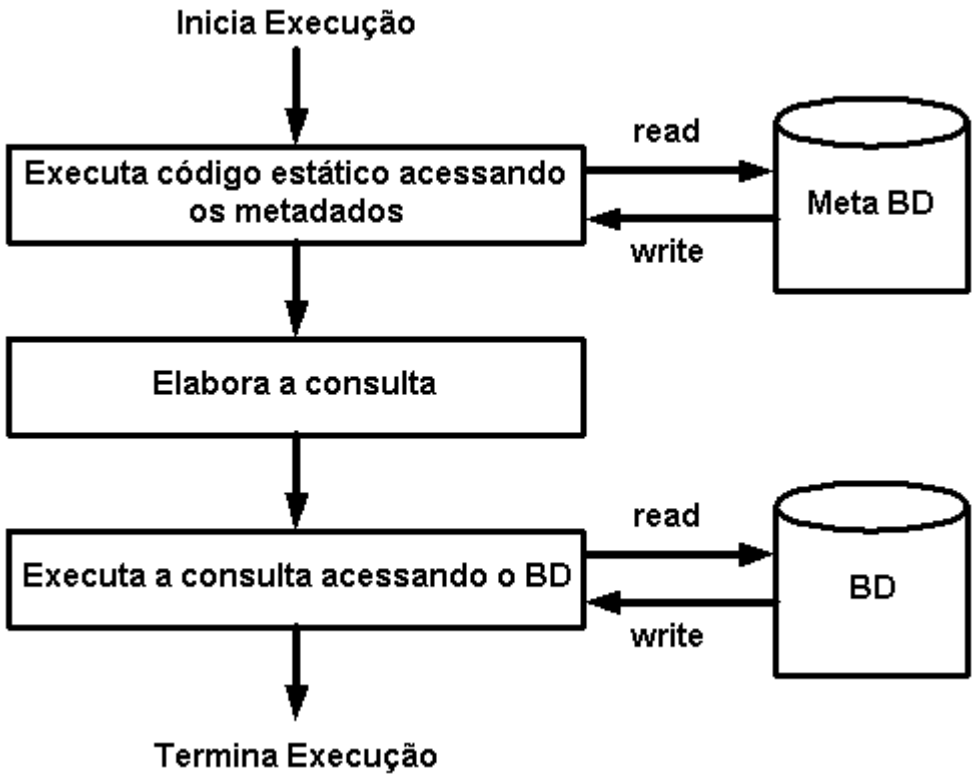
- SGBDRs normalmente vêm com uma linguagem 4GL
- Podem reduzir o esforço de desenvolvimento, mas não são transportáveis

Camada Genérica Orientada a Objeto

- Uma camada OO é escrita para completamente esconder o BD
- Ela pode ser genérica (para qualquer objeto, qualquer aplicação) com bastante esforço
- Geralmente sofre com desempenho

Interação baseada em Metamodelo

- A camada de acesso ao BD acessa o metamodelo e constrói a consulta dinamicamente
- A consulta resultante acessa o BD
- Solução freqüentemente usada com frameworks
- Cuidado! Essa técnica requer muito esforço!
  - É melhor usar uma solução pronta, tal como Toplink ou CocoBase



Uso de Middleware

- Os serviços de persistência podem ser obtidos automaticamente usando Middleware
  - J2EE/Enterprise JavaBeans
  - .NET (ex-MTS, ex-COM+)
- Vantagem: muitos serviços são obtidos além da persistência:
  - Gerência de transações distribuídas
  - Directory/Naming
  - Segurança
  - Tolerância a falha com Failover
  - Balanceamento de carga

- Resource pooling
- Monitoring, logging, ...

programa