Ambientes de Desenvolvimento Avançados

http://www.dei.isep.ipp.pt/~jtavares/ADAV/ADAV.htm

Aula 7

Engenharia Informática

2005/2006

José António Tavares
jrt@isep.ipp.pt

Microsoft®

Designing Data Tier Components and Passing Data Through Tiers

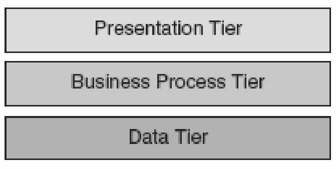
Projecto de Componentes da Camada de Acesso a Dados e Passagem de Dados entre Camadas

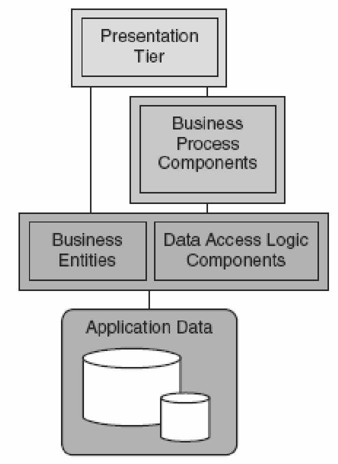
patterns & practices
proven practices for predictable results

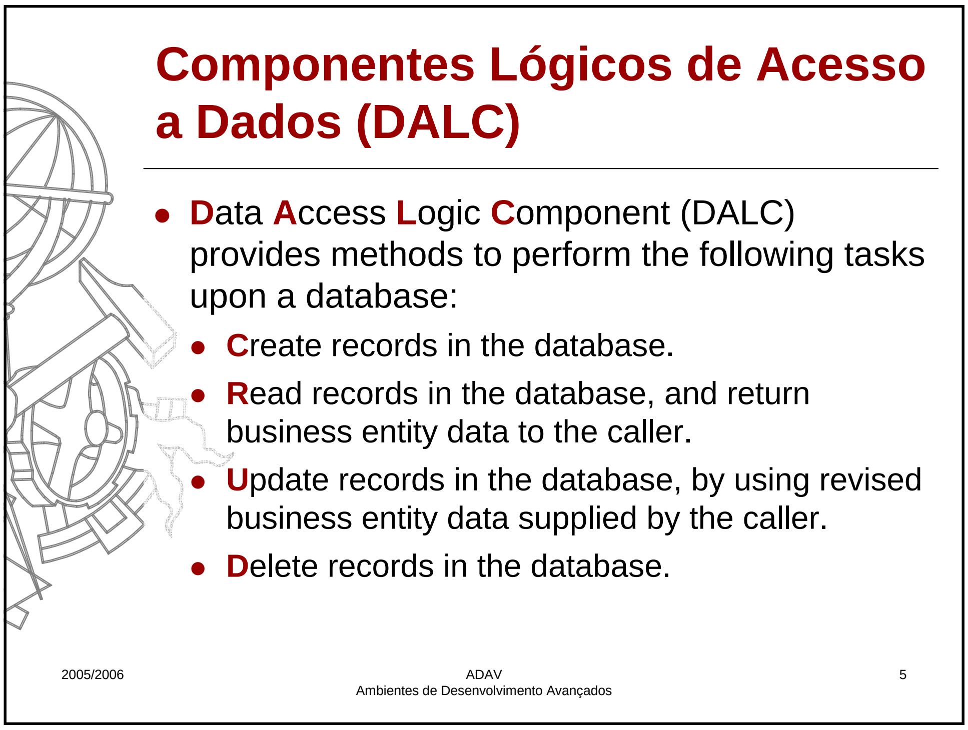
Conteúdo

Intodução

Camadas comuns numa aplicação distribuída

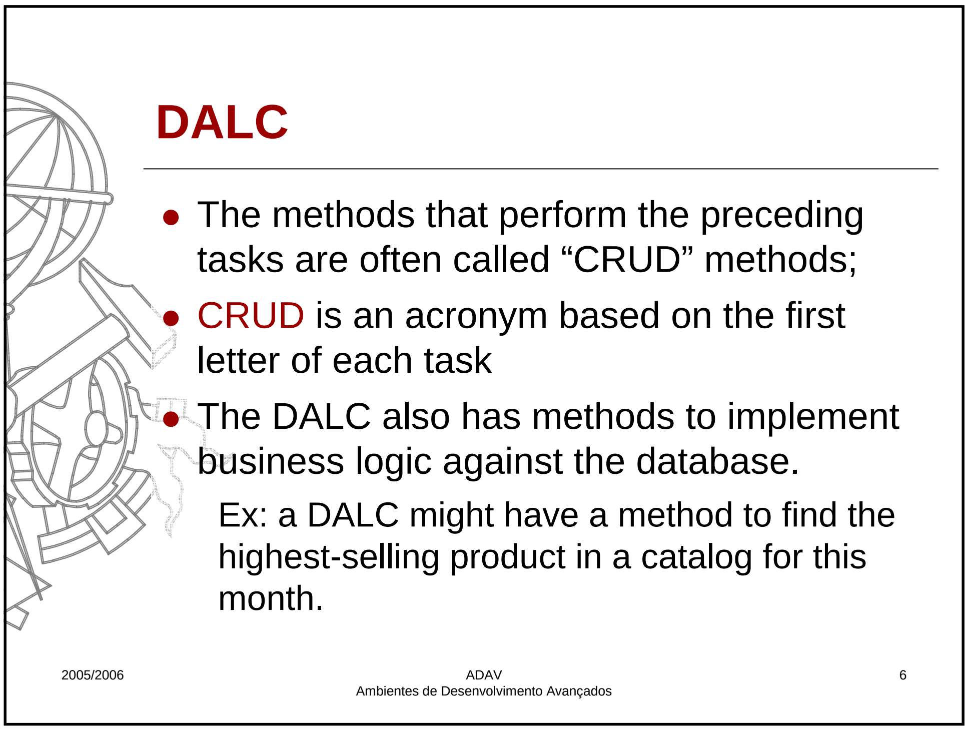






Componentes Lógicos de Acesso a Dados (DALC)

- **D**ata **A**ccess **L**ogic **C**omponent (DALC) provides methods to perform the following tasks upon a database:
  - **C**reate records in the database.
  - **R**ead records in the database, and return business entity data to the caller.
  - **U**pdate records in the database, by using revised business entity data supplied by the caller.
  - **D**elete records in the database.



DALC

- The methods that perform the preceding tasks are often called “CRUD” methods;
- CRUD is an acronym based on the first letter of each task
- The DALC also has methods to implement business logic against the database.

  Ex: a DALC might have a method to find the highest-selling product in a catalog for this month.

DALC

- Typically, a DALC accesses a single database and encapsulates the data-related operations for a single table or a group of related tables in the database.
  - Ex: you might define one DALC to deal with the `Customer` and `Address` tables in a database, and another DALC to deal with the `Orders` and `OrderDetails` tables.

Representação de Entidades de Negócio

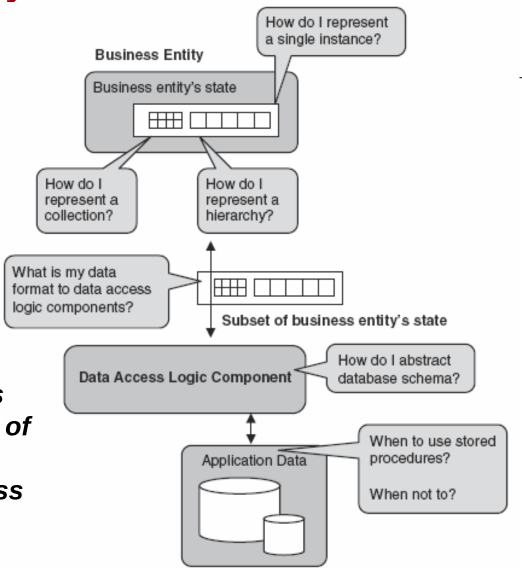
- Each DALC deals with a specific type of Business Entity (BE).
- For example, the Customer DALC deals with Customer BE.
- There are many different ways to represent BE, depending on different factors:

Representação de Entidades de Negócio

Such factors are as the following:

- need to bind BE data to controls in a WinForm or on a ASP.NET page?
- need to sort or search operations on the BE data?
- application deals with BE one at a time, or does it typically deal with sets of BE?
- deploy your application locally or remotely?
- will the BE be used by XML Web services?
- how important are nonfunctional requirements, such as performance, scalability, maintainability, and programming convenience?

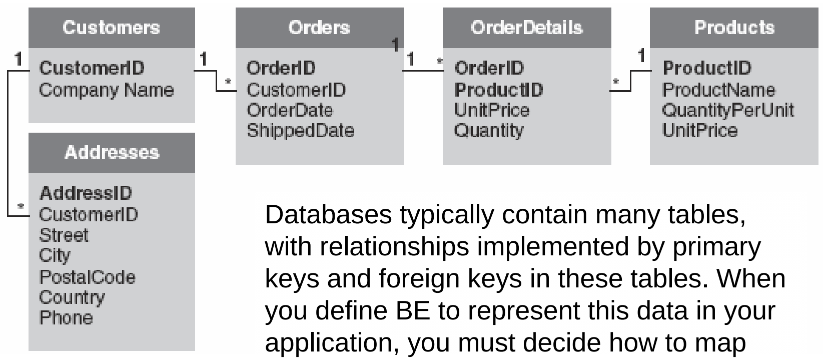
Representação de Entidades de Negócio


How do I represent a single instance?
Business Entity
Business entity's state
How do I represent a collection?
How do I represent a hierarchy?
What is my data format to data access logic components?
Subset of business entity's state
Data Access Logic Component
How do I abstract database schema?
Application Data
When to use stored procedures?
When not to?

Technical considerations that influence the design of data access logic components and Business Entities

Mapeamento de Dados Relacionais a Entidades de Negócio

An hypothetical retailer’s database



Databases typically contain many tables, with relationships implemented by primary keys and foreign keys in these tables. When you define BE to represent this data in your application, you must decide how to map these tables to BE.

Mapeamento de Dados Relacionais a Entidades de Negócio

Typical operations in the hypothetical retailer’s application are as follows:

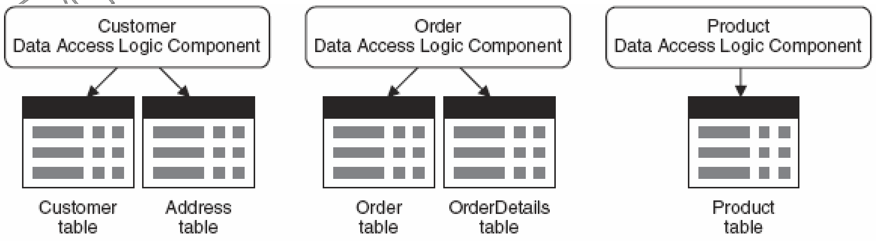
- Get (or update) information about a customer, including his or her addresses.
- Get a list of orders for a customer.
- Get a list of order items for a particular order.
- Place a new order.
- Get (or update) information about a product or a collection of products.

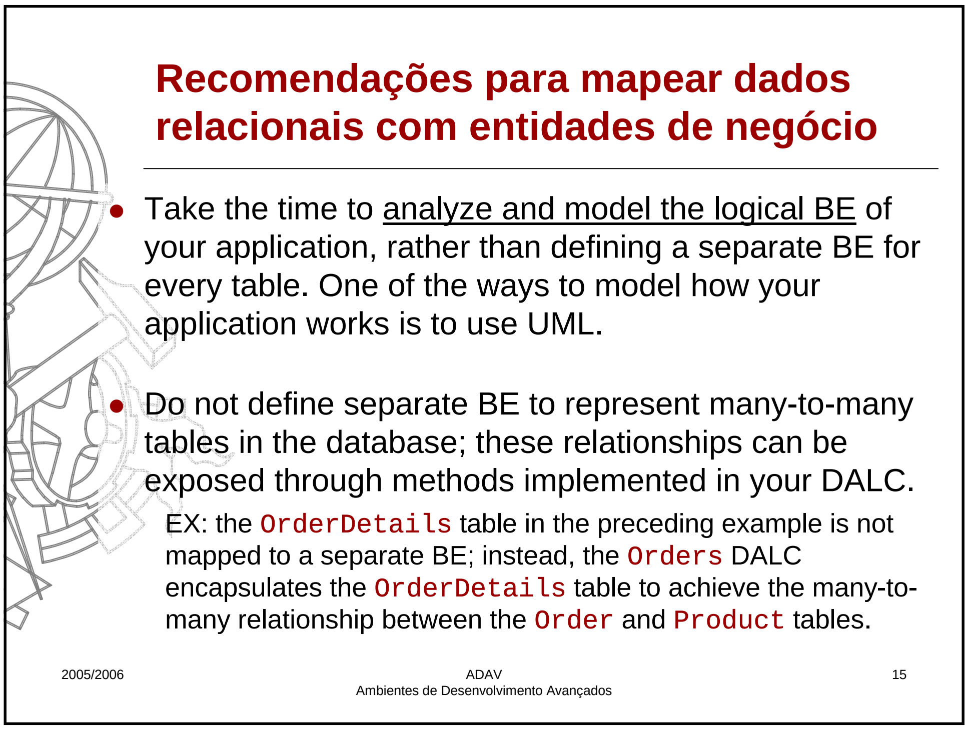
Mapeamento de Dados Relacionais a Entidades de Negócio

- There are three logical BE that the application will handle: a Customer, an Order, and a Product.
- For each BE, a separate DALC will be defined, as follows:
  - Customer DALC. This class will provide services to retrieve and modify data in the `Customer` and `Address` tables.
  - Order DALC. This class will provide services to retrieve and modify data in the `Order` and `OrderDetails` tables.
  - Product DALC. This class will provide services to retrieve and modify data in the `Product` table.

Mapeamento de Dados Relacionais a Entidades de Negócio

The relationships between the data access logic components and the tables that they represent in the database.


Customer Data Access Logic Component
Order Data Access Logic Component
Product Data Access Logic Component
Customer table
Address table
Order table
OrderDetails table
Product table

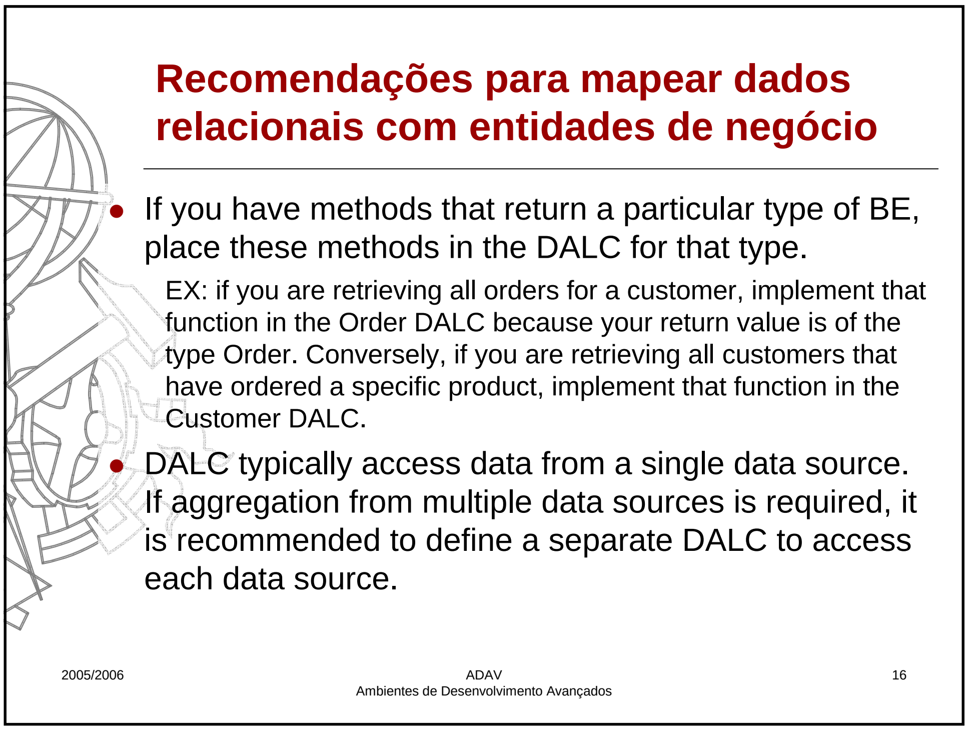


Recomendações para mapear dados relacionais com entidades de negócio

- Take the time to <u>analyze and model the logical BE</u> of your application, rather than defining a separate BE for every table. One of the ways to model how your application works is to use UML.
- Do not define separate BE to represent many-to-many tables in the database; these relationships can be exposed through methods implemented in your DALC.

  EX: the `OrderDetails` table in the preceding example is not mapped to a separate BE; instead, the `Orders` DALC encapsulates the `OrderDetails` table to achieve the many-to-many relationship between the `Order` and `Product` tables.

2005/2006 ADAV Ambientes de Desenvolvimento Avançados 15



Recomendações para mapear dados relacionais com entidades de negócio

- If you have methods that return a particular type of BE, place these methods in the DALC for that type.

  EX: if you are retrieving all orders for a customer, implement that function in the Order DALC because your return value is of the type Order. Conversely, if you are retrieving all customers that have ordered a specific product, implement that function in the Customer DALC.
- DALC typically access data from a single data source. If aggregation from multiple data sources is required, it is recommended to define a separate DALC to access each data source.

2005/2006 ADAV Ambientes de Desenvolvimento Avançados 16

Recomendações para mapear dados relacionais com entidades de negócio

- There are two reasons for separate DALC:
  - **Transaction management is centralized to the business process component** and does not need to be controlled explicitly by the DALC. If you access multiple data sources from one DALC, you will need the DALC to be the root of transactions, which will introduce additional overhead on functions where you are only reading data.
  - **Aggregation is usually not a requirement** in all areas of the application, and by separating the access to the data, you can let the type stand alone as well as be part of an aggregation when needed.

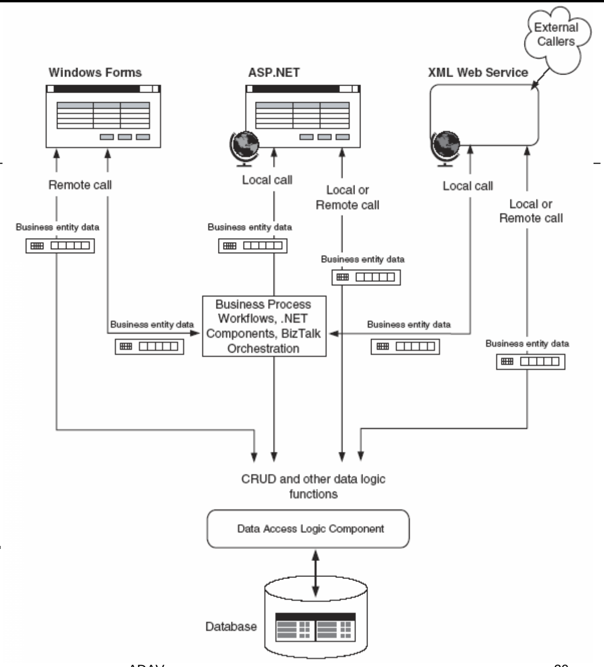
Implementação dos Componentes Lógicos de Acesso a Dados

- A DALC is a **stateless** class, meaning that all messages exchanged can be interpreted independently.
- DALC provides methods for accessing one or more related tables in a single database, or in some instances, multiple databases as in the case of horizontal database partitioning.
- Typically, the methods in a DALC invoke stored procedures to perform their operations.

Implementação dos Componentes Lógicos de Acesso a Dados

- One of the key goals of DALC is to hide the invocation and format idiosyncrasies of the database from the calling application.
- Specifically, a DALC handle the following implementation details:
  - Manage and encapsulate locking schemes
  - Handle security and authorization issues appropriately
  - Handle transaction issues appropriately
  - Perform data paging
  - Perform data-dependent routing if required
  - Implement a caching strategy if appropriate, for queries of nontransactional data
  - Perform data streaming and data serialization

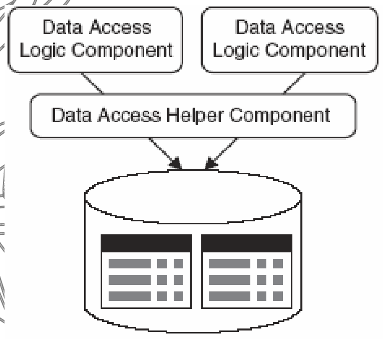
Cenários para os DALC



DALC can be called from a variety of application types:
- Windows Forms applications,
- ASP.NET applications,
- XML Web services
- Business processes.

These calls might be local or remote, depending on how you deploy your applications.

Implementação das classes de um DALC


Data Access Logic Component
Data Access Logic Component
Data Access Helper Component

- DALC use ADO.NET to execute SQL statements or call stored procedures.
- If an application contains multiple DALC, you can simplify the implementation of DALC classes by using a **data access helper component**.
- Design your DALC classes to provide a consistent interface for different types of clients.

Microsoft provides Data Access Application Block

Implementação das classes de um DALC

- To support a diverse range of business processes and applications, consider the following techniques to pass data to and from DALC methods:
  - Passing BE data into methods in the DALC – you can pass the data in several different formats: as a series of scalar values, as an XML string, as a DataSet, or as a custom BE Component.
  - Returning BE data from methods in the DALC – you can return the data in several different formats: as output-parameter scalar values, as an XML string, as a DataSet, as a custom BE Component, or as a data reader.

Pág. 12-14 – Advantages/Disadvantages

Utilização de *Stored Procedures* com DALC

Vantagens

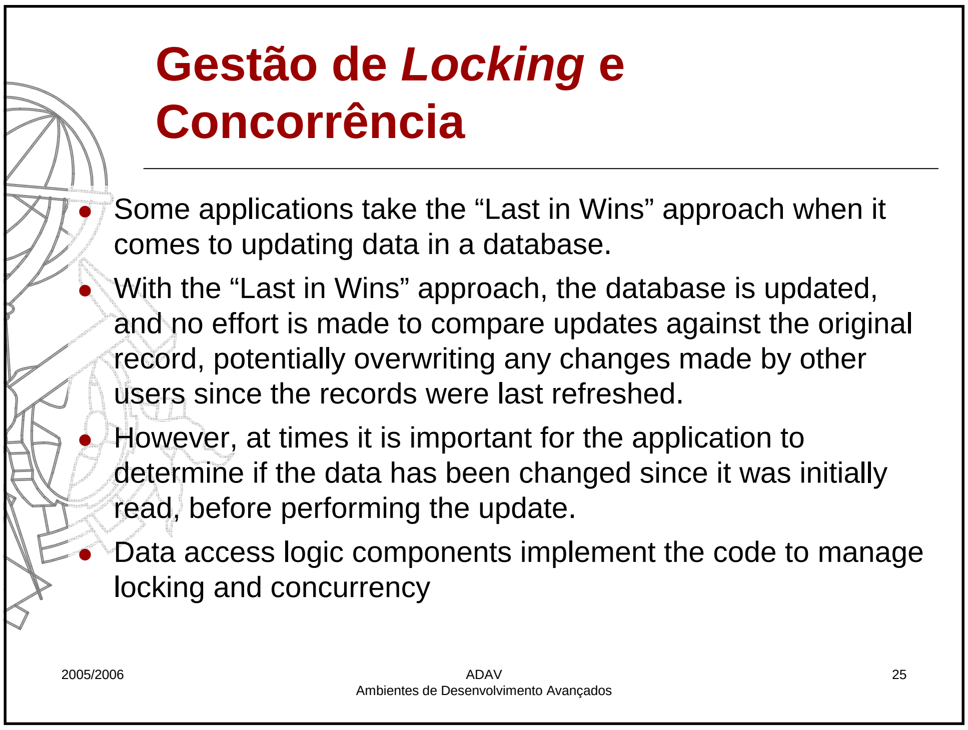
- Improved performance
- Can individually secured within the database
- Easier maintenance
- Add an extra level of abstraction from the underlying database schema
- Can reduce network traffic

Desvantagens

- Excessive load on the server if the logic implemented entirely in stored procedures
- Maintenance and the agility becomes an issue when you must modify business logic in T-SQL
- Stored procedures is most often a specialized skill

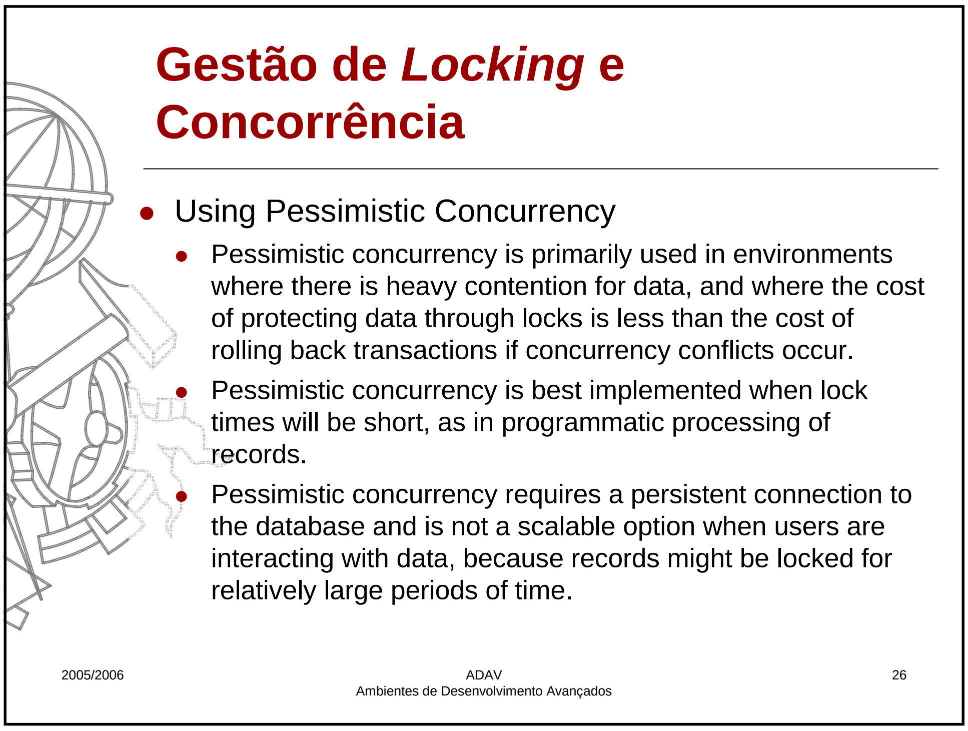
Utilização de *Stored Procedures* com DALC

- Recommendations for Using Store Procedures (SP) with DALC – p15-16:
  - Exposing SP – Only DALC
  - Each SP should be called by only one DALC and associated with the DALC that owns the action
  - Naming SP – choose SP names that emphasize the DALC to witch they pertain
  - Addressing security issues – ex: do not create a string by concatenating values without parameters



Gestão de *Locking* e Concorrência

- Some applications take the “Last in Wins” approach when it comes to updating data in a database.
- With the “Last in Wins” approach, the database is updated, and no effort is made to compare updates against the original record, potentially overwriting any changes made by other users since the records were last refreshed.
- However, at times it is important for the application to determine if the data has been changed since it was initially read, before performing the update.
- Data access logic components implement the code to manage locking and concurrency



Gestão de *Locking* e Concorrência

- Using Pessimistic Concurrency
  - Pessimistic concurrency is primarily used in environments where there is heavy contention for data, and where the cost of protecting data through locks is less than the cost of rolling back transactions if concurrency conflicts occur.
  - Pessimistic concurrency is best implemented when lock times will be short, as in programmatic processing of records.
  - Pessimistic concurrency requires a persistent connection to the database and is not a scalable option when users are interacting with data, because records might be locked for relatively large periods of time.

Gestão de *Locking* e Concorrência

- Using Optimistic Concurrency (p17-21)
  - Optimistic concurrency is appropriate in environments where there is low contention for data, or where read-only access to data is required. Optimistic concurrency improves database performance by reducing the amount of locking required, thereby reducing the load on the database server.
  - Optimistic concurrency is used extensively in .NET to address the needs of mobile and disconnected applications, where locking data rows for prolonged periods of time would be infeasible. Also, maintaining record locks requires a persistent connection to the database server, which is not possible in disconnected

Interoperabilidade - COM

- If you want your DALC class to be callable from COM clients, the recommended approach is to define your DALC by using the preceding guidelines and provide a wrapper component.

Interoperabilidade - COM

- However, if you want COM clients to be able to access your DALC, consider the following recommendations:
  - Define the class and its members as public.
  - Avoid using static members.
  - Define event-source interfaces in managed code.
  - Provide a constructor that does not use parameters.
  - Do not use overloaded methods. Use methods with different names instead.
  - Use interfaces to expose common operations.
  - Use attributes to provide extra COM information for your class and members.
  - Include HRESULT values in any exceptions thrown from .NET code.
  - Use automation-compatible data types in method signatures.

Questões

?