FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CARLOS HENRIQUE PAVAM

CRIAÇÃO DE UM BANCO DE DADOS ORIENTADO A OBJETO UTILIZANDO DIAGRAMAS DA UML

MARÍLIA
2007

CARLOS HENRIQUE PAVAM

CRIAÇÃO DE UM BANCO DE DADOS ORIENTADO A OBJETO UTILIZANDO DIAGRAMAS DA UML

Monografia apresentada ao Programa de Conclusão de Curso do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Bacharelado em Ciência da Computação.

Orientador:
Prof. Dr. Edmundo Sergio Spoto

MARÍLIA
2007

PAVAM, Carlos Henrique
Criação de banco de dados orientado a objeto utilizando diagramas da Uml.
/ Carlos Henrique Pavam; orientador: Edmundo Sergio Spoto. Marília, SP: [s.n.], 2007.

43 f.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília - Fundação de Ensino Eurípides Soares da Rocha.

1. Banco de dados 2. Uml

CDD: 005.74

CARLOS HENRIQUE PAVAM

CRIAÇÃO DE UM BANCO DE DADOS ORIENTADO A OBJETO UTILIZANDO DIAGRAMAS DA UML

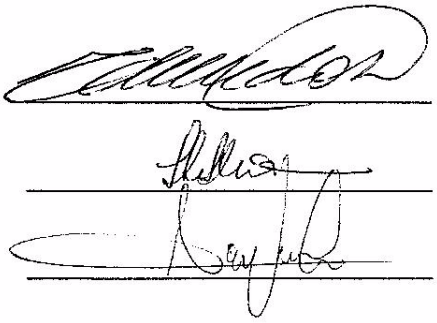
Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 7,5 (Sete e meio)

Orientador: Edmundo Sérgio Spoto

1º. Examinador: Fátima L. S. Nunes Marques

2º. Examinador: Valter Vieira de Camargo



Marília, 12 de Novembro de 2007.

PAVAM, Carlos Henrique, Criação de Um Banco de Dados Orientado a Objeto Utilizando Diagramas da UML. 2007. 48f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação). Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

RESUMO

O paradigma de orientação objetos vem se difundindo no desenvolvimento de banco de dados. Apesar da grande maioria dos sistemas gerenciadores de banco de dados (SGBD) ainda serem baseados no modelo relacional, este vem se tornando insuficiente à demanda cada vez mais complexa das aplicações. O modelo orientado a objetos é baseado no conceito de classes e utilizam encapsulamento, herança e polimorfismo. Uma das vantagens do uso de um SGBDOO consiste em que os objetos possam ser armazenados de forma transparente. Com o crescente uso de linguagens orientadas a objetos, os bancos tradicionais de dados começaram a apresentar limitações. Estas limitações comprometem a integridade dos dados como também podem influenciar no desempenho do sistema.

Palavras-chave: Orientação a Objetos, Banco de Dados, UML.

PAVAM, Carlos Henrique, Criação de Um Banco de Dados Orientado a Objeto Utilizando Diagramas da UML. 2007. 48f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação). Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

ABSTRACT

The paradigm of orientation objects are diffusing in the database development. In spite of the great majority of the systems database managers (SGBD) they are still based on the relacional model, this is turning insufficient to the demand, more and more complex of the applications.
The model guided to objects is based on the concept of classes and they use encapsulament, inheritance and polimorfism. One of the advantages of the use of a SGBDOO consists in that the objects can be stored in a transparent way.
With the crescent use of languages oriented to objects, the traditional banks of data began to present limitations. These limitations put in risk the integrity of the data as well as they can influence in the acting of the system.

Key-Word: Orientation to Objects, database, UML.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Segundo SILBERSCHATZ (1999), os Sistemas Gerenciadores de Banco de Dados (SGBD) mais utilizado é o modelo relacional. Este modelo se baseia em uma única estrutura de dados, a de relação. Uma relação significa uma tabela composta por linhas (tuplas) e colunas (atributos), que contêm um tipo especifico de dados simples.

O modelo relacional é adequado para aplicações convencionais, pois suporta uma pequena quantidade de tipos de dados e devido ao fato das aplicações computacionais terem evoluído juntamente com o poder de processamento das máquinas. Com a necessidade de tratar dados não convencionais e complexos, fez com que surgissem os sistemas gerenciadores de banco de dados orientados a objetos (SGBDOO).

Esta limitação do modelo relacional causa problemas no desenvolvimento de aplicações que utilizam tecnologia de objetos, pois se desperdiça muito tempo de programação para que se consiga adaptar objetos com estrutura complexa antes de armazená-lo em um modelo de banco relacional como também para recuperá-los e adequá-los ao modelo de objeto.

Os conceitos de classe, objeto, tipo, identidade, encapsulamento, herança, agregação, método e polimorfismo, são implementados de forma natural pelos SGBDOOs, isto é, eles mantêm uma correspondência direta entre o mundo real e objetos de banco de dados de forma que objetos não percam sua integridade e identidade e possam ser facilmente identificados e operados.

Esta tecnologia possui mecanismos especiais para controlar transações entre objetos e técnicas de armazenamento que facilitam a recuperação rápida de objetos.

1.1. Objetivo

O objetivo deste trabalho é utilizar os diagramas de classes para geração de um sistema de BDOO ou BDOR (Banco de dado Orientado a Objetos ou Objeto Relacional) extraindo as informações em destaques no diagrama de classe. Para isso será gerada uma notação no diagrama de Classe que facilitará ao desenvolvedor quais métodos foram desenvolvidos na Linguagem de Aplicação (Java ou C++) e quais métodos serão colocados no BDOO (que serão executados dentro do Servidor de BD).

1.2. Organização

Neste capítulo foi apresentada uma breve introdução deste projeto, mostrando as principais motivações e objetivos que nos levaram a desenvolvê-lo. No Capítulo 2 são apresentados os conceitos da Orientação a Objeto, explicando a atuação de cada um deles. No Capítulo 3 são apresentados os conceitos de UML utilizados no desenvolvimento do banco proposto. Será explicada no Capítulo 4, a modelagem do banco baseado na UML. No Capítulo 5 será apresentada a implementação do banco orientado a objetos no Oracle 9i. O Capítulo 6 terá a implementação dos métodos dentro do banco, e por fim no Capítulo 7 a conclusão do trabalho proposto.

2. CONCEITOS DE ORIENTAÇÃO A OBJETO

A Orientação a Objetos (OO) surgiu na tentativa de solucionar problemas complexos existentes no desenvolvimento de Software e resolvê-los de maneira confiável com baixo custo de desenvolvimento e manutenção (DATE, 2000).

Com a utilização deste conceito em banco de dados, o mesmo passou a se tornar um sério concorrente do modelo relacional, pois sistemas clássicos são inadequados em vários aspectos, e vários aspectos novos que são necessários já estão sendo utilizados em diversos bancos.

O banco de dados orientado a objeto não se difere muito de um sistema orientado a objeto, sendo capaz de lidar com objetos e operações sobre esses objetos, permitindo suporte a aplicações altamente dinâmicas que manipulam grandes objetos estruturados e complexos.

2.1. Abstração

Através da abstração, se observa a realidade e dela abstrai entidades, ações, consideradas essenciais para uma aplicação. São consideradas apenas as propriedades comuns de um conjunto de objetos, em que os detalhes são omitidos é utilizada com freqüência na definição de valores similares e na construção de um tipo a partir de outro, em diferentes níveis de abstração. Com a utilização da abstração é possível a geração de tipos baseados em hierarquias de tipos e de relacionamentos (SOUZA, 2005).

Os conceitos de abstração mais utilizados em banco de dados são generalização e agregação. A generalização corresponde (a Especialização) que a partir de duas categorias abstrai-se uma categoria mais genérica. As subcategorias satisfazem todas as propriedades das categorias de que elas constituem sobre as especializações, deve existir pelo menos uma propriedade que distingue duas categorias especializadas. Agregação (Decomposição) é a composição de uma nova categoria como um agregado de categorias pré-existentes instâncias de uma categoria (Estudante) são compostas por instâncias de outras categorias (Nome, Endereço) (FERNANDES, 2002).

2.2. Objeto

Entidades independentes, compostas por um conjunto de elementos que a caracterizam (domínio) e as ações que agem sobre esse domínio (operações). Os objetos são abstrações de dados do mundo real, com uma interface de nomes de operações (FERNANDES, 2002).

As abstrações da representação e das operações são ambas suportadas no modelo de dados orientado a objetos, são incorporadas as noções de estruturas de dados e de comportamento (SOUZA, 2005).

O comportamento de um objeto é representado através dos métodos que consistem em uma assinatura única dentro do objeto, sendo responsável pela execução da operação. O estado de um objeto e baseado no valor de cada uma das suas propriedades, atributos do próprio objeto ou relações com outros objetos. Os atributos podem ter valores muito complexos, podendo inclusive ser outro objeto (BOSCARIOLI *et al,* 2006).

2.3. Objetos Complexos

Objetos complexos são criados por construtores (conjuntos, listas, tuplas, registros, coleções, *arrays*) aplicados a objetos simples (inteiros, *booleanos*, *strings*). Nos modelos orientados a objetos, qualquer construtor pode ser aplicado a qualquer objeto, por isso é chamado de ortogonal. No modelo relacional só é possível aplicar o construtor de conjuntos as tuplas e o construtor de registro a valores atômicos (SOUZA, 2005).

Para a manutenção de objetos complexos, independentes de sua composição é requerida a definição de operadores apropriados para sua manipulação como um todo, e transitivos para seus componentes (SOUZA, 2005).

2.4. Classe

Classes são conjuntos de objetos de mesmo tipo, com comportamentos e propriedades em comum. Novas classes podem ser criadas para que se adéquem a necessidades de cada aplicação, podendo também estarem combinadas com tipos já existentes no sistema (BOSCARIOLI *et al*, 2006).

Ela descreve a estrutura de dados, e o comportamento de objetos similares. Todo objeto é uma instância de uma Classe, e todas as instâncias de uma classe têm valores próprios para os atributos especificados na classe, os objetos representados por determinada classe diferenciam-se entre si pelos valores de seus atributos.

2.5. Métodos

Um método é um código associado (ou “encapsulado”) ao objeto. Sua principal característica é ter acesso ilimitado aos dados do objeto ao qual está ligado. Corresponde ao comportamento dos objetos, implementando uma operação associada a uma ou mais classes.

Cada objeto tem certo número de operações para ele definido. Para cada operação pode-se ter um ou mais métodos de implementação associados. As mensagens são a forma mais usada para se ativar os métodos. Os objetos se comunicam e são ativados através de mensagens enviadas entre eles (FERNANDES, 2002).

2.6. Herança

É um mecanismo que permite definir uma nova classe (subclasse) a partir de uma classe já existente (superclasse). Ao se estabelecer uma especialização (subclasse) de uma classe, a subclasse herda as características comuns da superclasse, isto é, a especificação dos atributos e dos métodos da superclasse passa a fazer parte da especificação dos atributos e dos métodos da subclasse, esta também pode adicionar novos métodos, como também reescrever

métodos herdados. O objeto tem a habilidade de derivar seus atributos (dados) e métodos (funcionalidade) automaticamente de outro objeto (FERNANDES, 2002).

Pelo fato das classes poderem ser construídas a partir de outras já existentes, facilita a extensibilidade, quando uma mensagem é enviada para um objeto, a procura do método correspondente começa pela classe do objeto, se o método não for encontrado, a procura continua na superclasse.

2.6.1. Tipos de Herança

Existem dois tipos de heranças:

i) Herança Simples: quando uma classe é subclasse de somente uma superclasse.

ii)Herança Múltipla: quando uma classe é subclasse de várias superclasses e conseqüentemente herda as características de cada uma delas.

A palavra herança, por si só, dá uma idéia do que significa este conceito. Sua conceituação é simples: é o mecanismo de reutilização de atributos e operações definidos em classes (ou tipos de objetos) gerais por classes (ou tipos de objetos) mais específicas (FERNANDES, 2002).

Segundo FERNANDES (2002), o tipo de objeto mais genérico é chamado de SUPERTIPO (também conhecido como tipo pai ou tipo base) e o tipo de objeto mais especializado é chamado de SUBTIPO (tipo filho ou tipo derivado).

A herança fornece um meio para se construir componentes reutilizáveis. Permite especificar atributos e operações comuns a várias classes com características semelhantes.

2.7. Polimorfismo

Ao receber uma mensagem para efetuar uma operação é o objeto quem determina como a operação deve ser efetuada, pois ele tem o comportamento próprio, como a responsabilidade é do receptor e não do emissor, pode acontecer que uma mesma mensagem ative métodos diferentes, dependendo da classe de objeto para onde é enviada a mensagem (FERNANDES, 2002).

Permite a criação de várias classes com interfaces idênticas, porém objetos e implementações diferentes, tem capacidade de uma mensagem ser executada de acordo com as características do objeto que está recebendo o pedido de execução do serviço.

Polimorfismo então é a capacidade de o software construir, ao tempo de execução, o objeto adequado e, conseqüentemente, acionar o método associado com o tipo de objeto derivado específico do objeto (FERNANDES, 2002).

Isto permite a criação de programas extensíveis, que podem crescer não apenas durante a criação original do projeto, mas ao longo do tempo quando novas características forem adicionadas.

2.8. Encapsulamento

O encapsulamento possibilita a distinção entre a especificação e a implementação das operações de um objeto, além de prover a modularidade que permite uma melhor estruturação das aplicações ditas complexas, bem como a segurança dentro do sistema. Em banco de dados se diz que um objeto está encapsulado quando o estado é oculto ao usuário e o objeto pode ser consultado e modificado exclusivamente por meio das operações a ele associadas (SOUZA, 2005).

A maior vantagem do encapsulamento esta em permitir que a representação interna do objeto seja alterada sem que qualquer das aplicações que o use precise ser reescrita, a não ser que mudanças nas representações internas sejam acompanhadas de mudanças no código que implementa os métodos aplicáveis.

Apenas a interface da classe é conhecida pelo usuário, ficando para o conjunto de métodos definidos, manipularem as instâncias da classe. Um método se torna um programa executável, quando especificado em uma classe que controla parte do comportamento dos objetos da classe (GALANTE *et al,* 2003).

2.9. Identificadores de Objetos

O Identificador de Objetos (OID) permite que entidades do mundo real sejam modeladas como objetos e tenham uma existência própria, sendo independente do seu valor, para cada objeto um identificador próprio que é definido pelo sistema. Este identificador garante a unicidade bem como ele mantém a integridade referencial permitindo assim que sejam manipulados pelos programas aplicativos (BERTINO, 1991).

Um usuário externo não consegue visualizar um OID, pois o mesmo é utilizado pelo sistema para identificação única de cada objeto, e também é base das referências entre

objetos. Uma propriedade importante do OID é ser imutável, seu valor nunca é alterado mesmo quando o objeto é excluído, seu OID não pode ser atribuído a outro objeto (GALANTE *et al,* 2003).

2.10. Banco de Dados Orientado a Objeto Versus Banco de Dados Relacional

Ambos os bancos de dados servem basicamente ao mesmo propósito: armazenamento de informações manutenção, regras de negócio, comparação e tratamento dos dados, a fim de obter resultados tangíveis. Porém, a maioria dos SGBDOO permite a representação de objetos e de valores, pelo fato do OID tornar um objeto imutável, os valores embutidos no objeto não podem ser referenciados unicamente. Sendo assim, o valor de um objeto pode ser atualizado sem perder sua identidade (GALANTE *et al,* 2003).

O conceito de OID se difere do conceito de chaves presentes no modelo relacional. Chaves são definidas por valor de um ou mais atributos, e que podem ser modificados ao longo do tempo, já dois objetos podem ter atributos idênticos que serão diferentes pelos seus OIDs. Quando se utiliza o OID, o sistema apresenta maior desempenho, pois sua implementação é em baixo nível, a seleção de chaves deixa de ser uma preocupação para os programadores (GALANTE *et al,* 2003).

Por exemplo: ao invés de se ter uma tupla DEPTO, e mais uma coleção de tuplas EMP correspondentes, que incluem valores de chaves estrangeiras que referenciam o valor da chave primária nessa tupla DEPTO, o usuário deve pensar diretamente em termos de um objeto departamento que contém um conjunto correspondente de objetos empregados, ao invés de lidar com variáveis de relações que são uma desvantagem em uma consulta no banco de dados relacional, pode se contratar um objeto empregado diretamente para o objeto departamento, ou seja, elevar o nível de abstração, utilizando-se de objetos complexos (DATE, 2000).

2.11. Variável Pública e Privada

A memória privada consiste em variáveis de instância, conhecidas como atributos, cujos valores representam o estado interno do objeto. Porém as variáveis de instância são completamente privadas e ocultas dos usuários em um sistema puro. Já a interface pública

consiste em definições de interface para os métodos que se aplicam a esse objeto. Pode-se dizer que a interface pública é parte do objeto de definição de classe para o objeto em questão, em vez de fazer parte do próprio objeto, ou seja, a interface pública é comum a todos os objetos da classe em questão, ao invés de ser específica para algum objeto individual. Os métodos são invocados através de mensagem, que significa uma invocação de operador, na qual o argumento destino se distingue e recebe um tratamento sintático especial.

Existem dois tipos de variáveis de instância: públicas e privadas. As variáveis públicas não são ocultas do usuário, porém as variáveis privadas estão totalmente ocultas. As variáveis de instância públicas são logicamente desnecessárias, pois se utilizar uma classe com determinadas variáveis de instância públicas, que utilizem uma sintaxe especial para seu acesso, e alterar a sua representação física, os programas que a utilizam tornarão inúteis, e será perdida a independência dos dados. Porém, se houver a definição de métodos o usuário poderá por meio de invocações de métodos apropriados encontrar o que deseja, sem se preocupar com a representação física do objeto, basta somente que os métodos sejam implementados de forma apropriada (DATE, 2000).

3. UML (UNIFIELD MODELING LANGUAGE)

A UML é uma forma de padronizar as metodologias de desenvolvimento de sistemas baseados em orientação a objetos, sendo necessário então um conjunto de termos, nomenclaturas e ferramentas padronizadas formando uma linguagem ou notação que os técnicos entendam de maneira correta e sem ambigüidade e efetivamente utilizem, facilitando assim grupos de desenvolvimento (CESAR, 2001).

A UML foi desenvolvida por Grady Booch, James Rumbaugh, e Ivar Jacobson que são três conhecidos autores de metodologias de desenvolvimento de software seguindo a abordagem orientada a objetos. A UML nasceu da junção do que havia de melhor nas suas respectivas metodologias.

A UML é uma composição das boas características de outras notações de ferramentas direcionadas à orientação a objetos, tornando-se assim a ferramenta ideal para conceber, compreender, testar, validar, arquitetar lógica e fisicamente e ainda identificar todos os possíveis comportamentos do sistema (MATOS, 2002).

Para o exercício de sua função, a UML possui três grandes blocos para modelar um sistema: itens, relacionamentos e diagramas. Itens são as abstrações identificadas como elementos de primeira classe em um modelo, os relacionamentos reúnem esses itens e os diagramas agrupam coleções interessantes de itens (BOOCH, 2000).

É uma linguagem padrão utilizada nos mais variados tipos de sistemas. Abrange qualquer característica do sistema em um de seus diagramas, como também é aplicada em diferentes fases de desenvolvimento do sistema, sendo para visualizar, especificar, construir e documentar os artefatos de um sistema.

3.1. Conteúdos dos Quatro Pontos de Vistas Essenciais

Com relação aos quatro pontos de vista essenciais, a modelagem de software orientado a objetos, serão apresentados os requisitos de conteúdo para que de fato possam constituir uma modelagem completa de um sistema (SILVA, 2007).

Em termos gerais relacionados à modelagem estrutural, observa-se a realização da definição do conjunto de elementos que compõem o sistema sendo modelado e os relacionamentos entre esses elementos.

3.1.1. Modelagem Estrutural do Sistema

A modelagem estrutural do sistema referencia o conjunto de elementos que formam um sistema orientado a objetos e seus relacionamentos. Com base nos elementos de composição como as classes, obtém-se essa visão apresentada pelo diagrama de classes, isto é, o conjunto de classes e seus relacionamentos (herança, associação, composição, agregação).

Existem outros modelos além do diagrama de classes, em que a modelagem estrutural do sistema é explorada em um nível de abstração mais elevado que o da organização de classes. Por exemplo, os diagramas de pacotes e a utilização de UML (SILVA, 2007).

3.1.2. Modelagem Estrutural da Classe

A modelagem estrutural da classe é composta pelos elementos denominados de métodos e atributos. Os tipos de atributo, como parâmetros e retorno, definem relacionamentos entre esses elementos. O diagrama de classes também possibilita a visão da estrutura de classe (SILVA, 2007).

3.1.3. Modelagem Dinâmica do Sistema

A modelagem dinâmica do sistema, realizada pelo diagrama de seqüência, consiste no conjunto de funcionalidades do software e o seu detalhamento, deixando explícito como os objetos (instâncias de classes) interagem em tempo de execução para efetuar cada funcionalidade (SILVA, 2007).

3.1.4. Modelagem Dinâmica da Classe

A execução de um programa, em parte, é observável em uma instância de classe isolada, correspondendo à evolução de estados dos objetos (instâncias das classes) e aos algoritmos dos métodos das classes (SILVA, 2007).

A modelagem dinâmica tem a obrigação de permitir que se possa observar o software em execução (SILVA, 2007).

A utilização dos quatro pontos de vista é uma ótima forma de avaliar a qualidade das notações, bem como as especificações. É também um excelente indicador de conclusão para projeto orientado a objetos em tempo de desenvolvimento, somente com o cumprimento dos quatro pontos de vista o sistema estará realmente concluído.

Na Figura 1 são apresentados os quatro pontos de vista utilizados para a modelagem orientada a objeto.

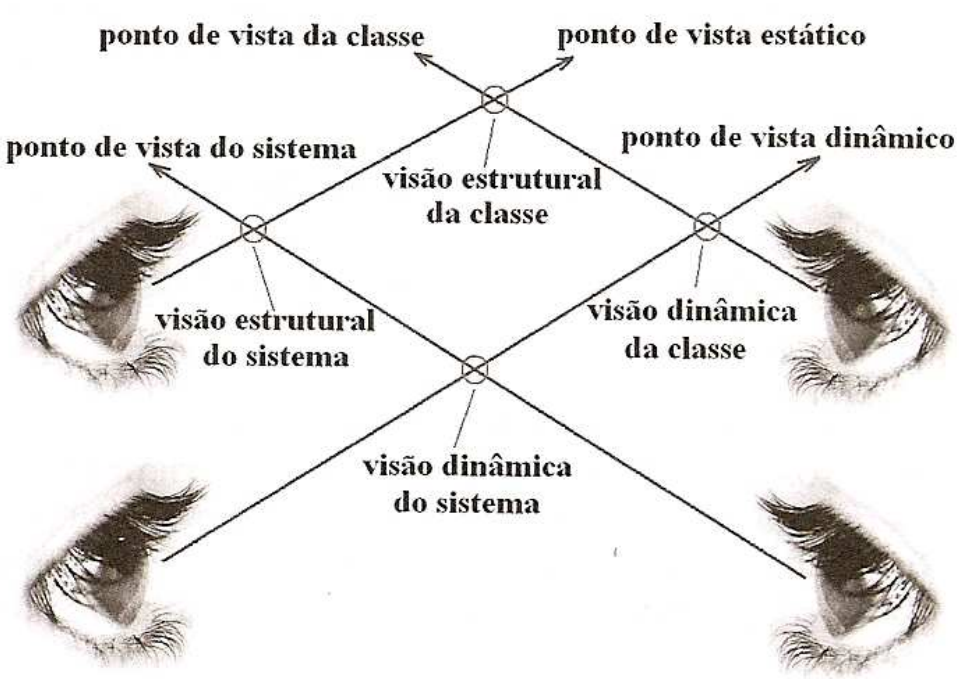


Figura 1 - Os Quatro Pontos de Vista para Modelagem OO (SILVA, 2007).

3.2. Diagramas Propostos

São apresentados sumariamente os diagramas que compõem a Linguagem UML. São também mostrados seus principais elementos sintáticos. Para que cada um possa esclarecer sua finalidade em um processo de modelagem.

3.3. Modelo de Casos de Uso

O diagrama de casos de uso é uma técnica utilizada para descrever as funcionalidades de um sistema. Um ator pode ser uma pessoa ou um sistema que interage com as operações do sistema. O caso de uso descreve o objetivo que um ator externo ao sistema tem em relação ao comportamento do sistema. Os atores se comunicam com o sistema através

dos casos de uso, sendo que o mesmo representa uma seqüência de ocorrências de execuções do sistema (CESAR, 2001).

Os atores e casos de uso são classes, em que um ator pode estar conectado a um ou mais casos de uso por meio de associações e ambos podem possuir relacionamentos de generalização que define um comportamento comum de herança em superclasses especializadas em subclasses.

O diagrama de casos de uso é muito importante na construção do diagrama de seqüência, pois é através da interação de classes e objetos que se pode executar uma atividade especifica no sistema (CESAR, 2001).

Em relação aos quatro pontos de vista o diagrama de casos de uso corresponde ao diagrama que modela a dinâmica do sistema no mais alto nível de abstração propiciado pela UML. Relacionando as funcionalidades do sistema modelado, através do caso de uso, em que os elementos externos se interagem com o sistema modelado através do elemento sintático ator. O diagrama é formado por pares desses elementos formando as ralações (SILVA, 2007). Na Figura 2 é apresentado um exemplo do Diagrama de Casos de Uso em que se utilizam dois atores (Funcionário e Cliente).

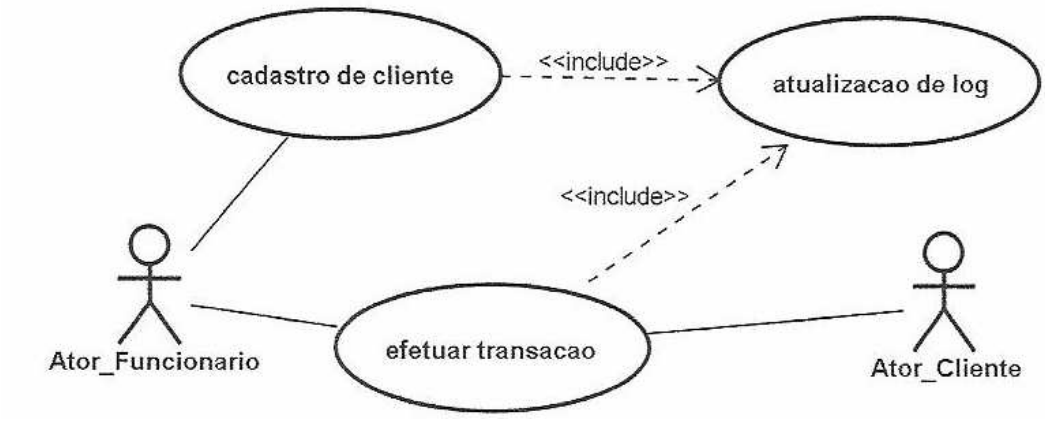


Figura 2 - Exemplo de Diagrama de Casos de Uso (SILVA, 2007).

3.4. Diagrama de Classes

O diagrama de classe descreve a estrutura estática das classes que formam a estrutura do sistema e suas relações. As relações entre as classes podem ser associações, dependência e especialização. As classes possuem além de um nome, os atributos e as operações que

desempenham para o sistema. Uma relação indica um tipo de dependência entre as classes, essa dependência pode ser forte como no caso da herança ou da agregação ou mais fraca como no caso da associação, mas indicam que as classes relacionadas cooperam de alguma forma para cumprir um objetivo para o sistema (CESAR, 2001).

Ao término do processo de modelagem do diagrama de classes, o mesmo pode ser traduzido em uma estrutura de código que servirá de base para a implementação do sistema. Observa-se, no entanto, que não existe no diagrama de classes uma informação sobre os algoritmos que serão utilizados nas operações, e também não se pode distinguir com precisão sobre a dinâmica do sistema porque não há elementos sobre o processo ou a seqüência de processamento neste modelo. Estas informações são representadas em outros diagramas, como os diagramas de seqüência (FURLAN, 1998).

O diagrama de classe produz a descrição mais próxima da estrutura do código de um programa, é o modelo fundamental de uma especificação orientada a objetos, isto é, mostra o conjunto de classes com seus atributos e métodos e os relacionamentos entre classes. Classes e relacionamentos são os elementos básicos de um diagrama.

Considerando a classificação dos quatro pontos de vista (Figura 1), observa-se que o diagrama de classes cobre dois deles: o primeiro é o de *Modelagem estrutural do sistema,* que mostra os elementos que compõem o sistema modelado, sendo as classes e seus relacionamentos, e o segundo a *Modelagem estrutural da classe,* que descreve a estrutura de uma classe, isto é, seus atributos e métodos (SILVA, 2007).

Na Figura 3 é apresentado um exemplo de Diagrama de Classe composto por seis classes que envolvem situações de herança, generalização e associações.

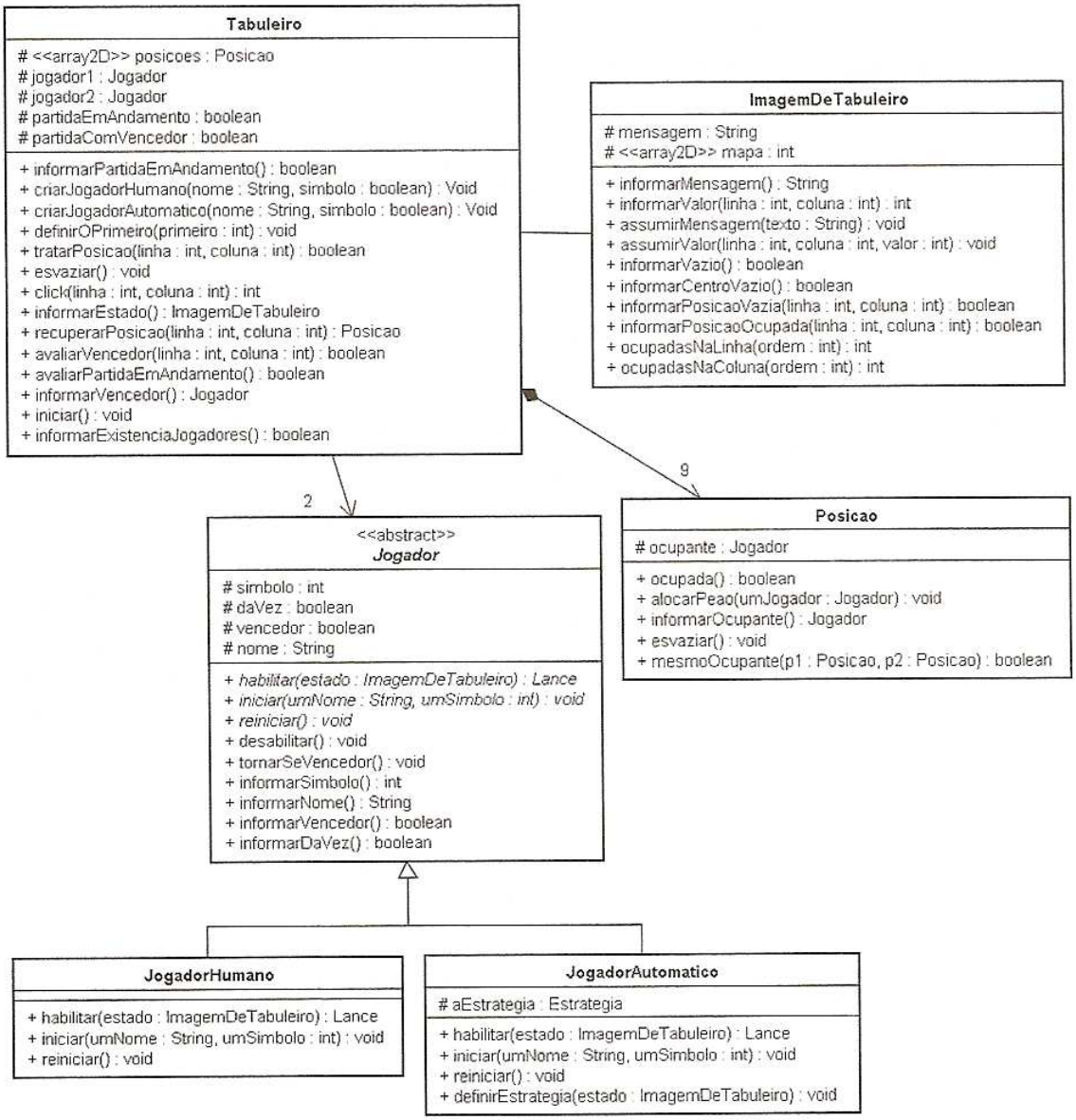


Figura 3 - Exemplo de Diagrama de Classes (SILVA, 2007).

3.5. Diagrama de Seqüência

O diagrama de seqüência exibe a colaboração dinâmica do sistema, permite modelar o processo através da troca de mensagens (eventos) entre os objetos do sistema. Os objetos são representados por linhas verticais e as mensagens como setas que partem do objeto que invoca um outro objeto (ver Figura 4), ou seja, na horizontal estão os objetos envolvidos na seqüência. A seta pode ser cheia para indicar uma mensagem de chamada ou tracejadas para indicar uma mensagem de retorno (FURLAN, 1998).

A funcionalidade do diagrama de seqüência é descrever os objetos que se interagem. Os principais elementos sintáticos são objeto e mensagem (enviada de um objeto a outro). É

usado para a modelagem dinâmica do sistema e tem como principal finalidade a modelagem orientada a objetos é o refinamento de casos de uso. O decorrer do tempo é visualizado observando-se o diagrama no sentido vertical de cima para baixo, a primeira mensagem enviada para invocar um método do objeto destinatário, é a que aparece mais acima do diagrama (SILVA, 2007).

Na Figura 4 é apresentado um exemplo de diagrama de Seqüência extraído de (SILVA, 2007). Observe também na Figura 4 que as mensagens enviadas de um objeto a outro (alocarPeao) faz com que um método denominado alocarPeao() seja construído na Classe de destino (Posição). Assim acontece com todas as mensagens enviadas exceto as mensagens de retornos (pontilhadas), passam a ser métodos nas classes destinos.

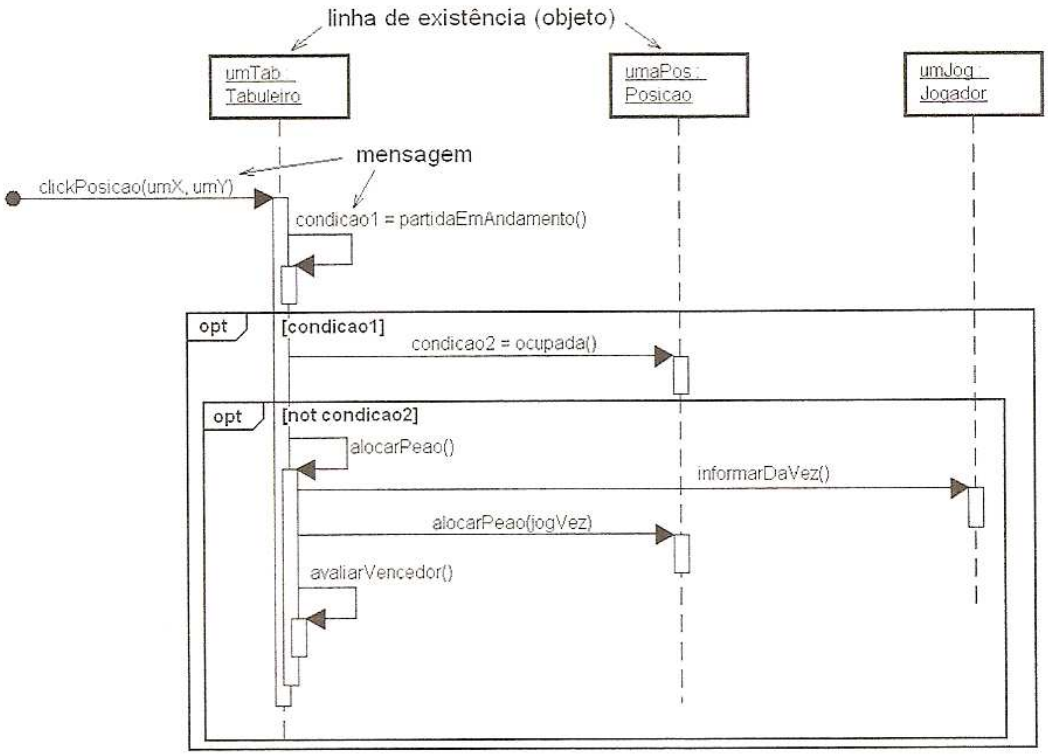


Figura 4 - Exemplo de Diagrama de Seqüência (SILVA, 2007).

4. MODELAGEM DO BANCO DE DADOS ORIENTADO A OBJETO UTILIZANDO UML

O banco de dados proposto é de uma locadora de veículos, onde através dos diagramas da UML, será possível determinar a interação do usuário para com o sistema, assim como determinar as classes necessárias e também seus relacionamentos.

4.1. Construção do Diagrama de Caso de Uso

O diagrama de caso de uso, apresentado na Figura 5, demonstra as funções de um ator externo a um sistema de controle de locadora de veículos, o diagrama especifica que funções o proprietário os funcionários e os usuários da locadora poderão desempenhar.

O diagrama demonstra que o proprietário é o único com direito de cadastrar um funcionário, ele também pode interagir com o restante do sistema, após o funcionário ter sido cadastrado, este pode cadastrar alterar e excluir qualquer dos dados, sendo estes o Cliente, Categoria, Carro e os Serviços Adicionais, pode realizar reserva do carro e também o aluguel

O cliente pode tanto reservar um carro, quanto alugá-lo de imediato. Tanto na reserva quanto no aluguel será verificada a existência do cliente assim como sua situação cadastral, somente após sua confirmação é que o cliente poderá retirar o veiculo. No ato da devolução será realizada uma verificação do veículo para saber se o estado da entrega confere com o da retirada e se houve algum dano no veículo.

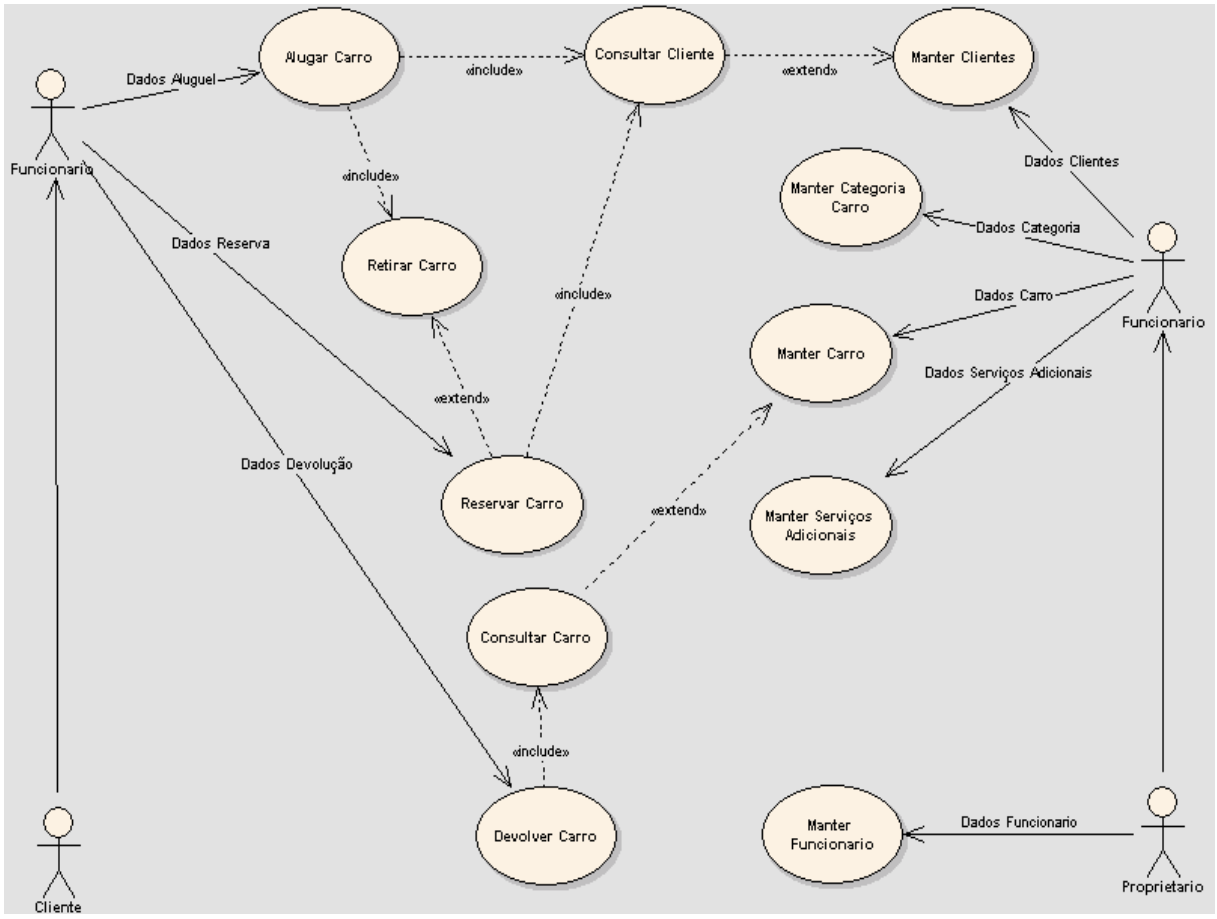


Figura 5 – Modelagem do Diagrama de Caso de Uso.

4.2. Construção do Diagrama de Classe

Após ter modelado o diagrama de use-cases, começa-se então a modelagem do diagrama de Classes que é baseado no diagrama apresentado na Figura 5.

O diagrama de classe demonstra a estrutura estática das classes de um sistema onde esta representa a estrutura que será gerenciada pela aplicação modelada. Na Figura 6 são demonstradas as classes que serão utilizadas no banco desta aplicação, bem como os seus respectivos relacionamentos, sendo que nestas notam se as associações, heranças e composições, bem como de algumas das formas suportadas pela ferramenta Oracle 9i na construção de um banco orientado a objeto.

Sobre os métodos que serão implementados, somente os métodos envoltos por um retângulo azul serão implementados dentro do Banco de Dados os demais serão implementados pelo próprio aplicativo. No Capítulo 6, está exemplificada a criação dos respectivos métodos dentro do Banco de Dados.

Na Figura 6 é demonstrado como será o armazenamento das informações em um banco de dados orientado a objeto.

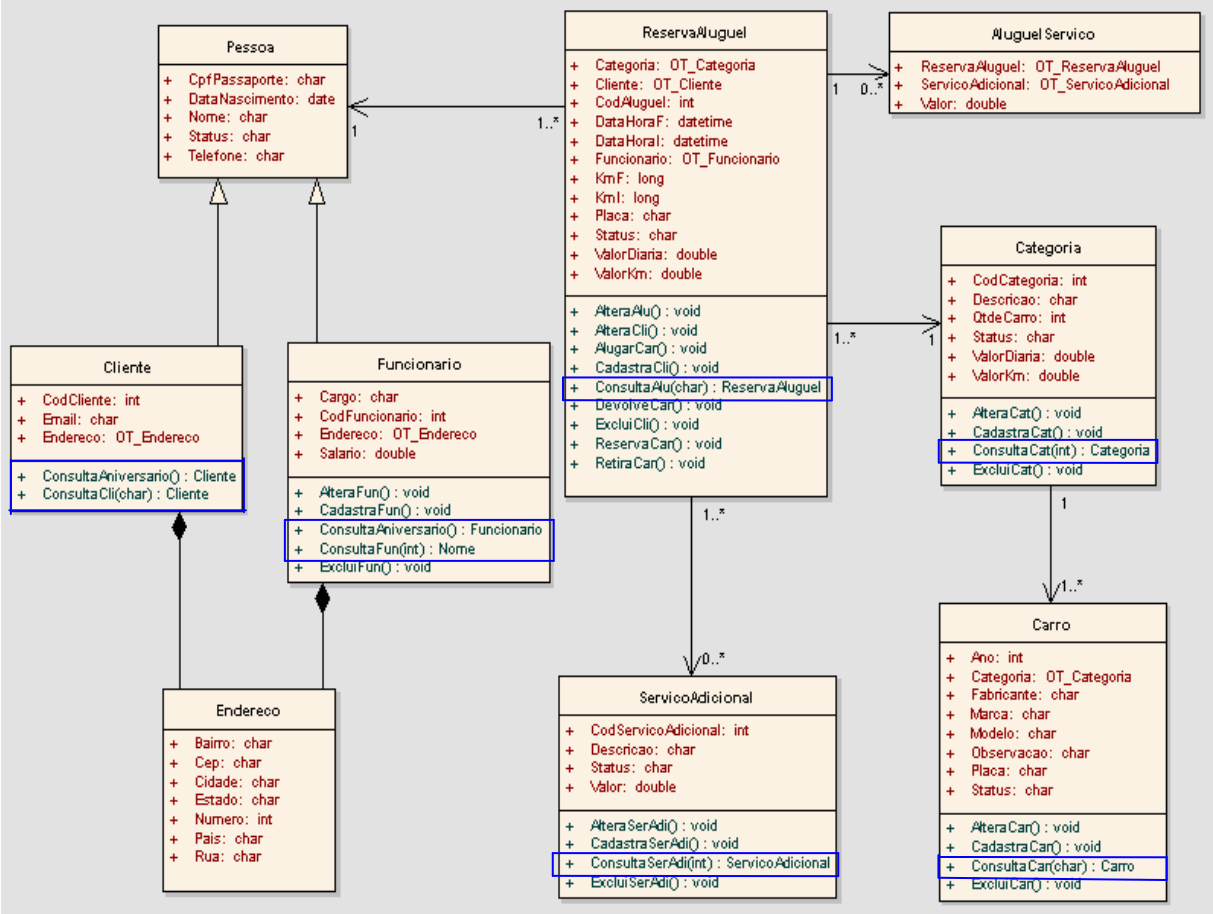


Figura 6 – Modelagem do Diagrama de Classes.

A classe ReservaAluguel permite realizar tanto a reserva quanto o aluguel e ela armazena todas as informações necessárias para ambas as situações. Esta classe ainda permite saber se foi realizado uma reserva ou um aluguel, e se ambas as operações foram concretizados ou não. A classe ReservaAluguel tem uma dependência de quase todas as outras classes, pois necessita de informações de cada uma delas, este tipo de dependência se chama associação pois indica que um atributo de um objeto é um objeto associado ou que a implementação de um método de objeto conta com o objeto associado.

As classes Categoria, Carro, Pessoa, simplesmente armazenam as informações a serem utilizadas em uma reserva ou aluguel. As classes ServicoAdicional e AluguelServico são eventualmente utilizadas, dependendo da necessidade do usuário, a classe ServicoAdicional sempre existe, porem nem sempre é utilizada, já a classe AluguelServico,

ela só passa a existir quando a classe de ServicoAdicional foi utilizada, do contrario, ela só existirá como um objeto vazio.

A classe Pessoa é um exemplo de hierarquia de onde se podem derivar vários tipos de pessoas, no diagrama acima vemos que ela se divide em Cliente e Funcionário, tanto cliente quanto funcionário tem acesso aos atributos do objeto Pessoa, como também pode realizar operações baseadas nestes atributos, a este tipo de relacionamento da se o nome de herança, onde Pessoa que é a classe Pai é chamada de classe abstrata, pois não é instanciada, e a classe Cliente e Funcionário são chamados de classe Filha, por herdar tudo o que é da classe pai.

Por fim, a classe Endereco, está vinculada às classes Cliente e Funcionário através de um relacionamento de composição. Este relacionamento permite que se gere um atributo do tipo de um objeto, fazendo com que ao armazenar o Cliente ou o Funcionário o Endereço também seja armazenado juntamente.

4.3. Construção do Diagrama de Seqüência

Para a construção dos métodos que existirão nas classes, e necessário que seja implementado o diagrama de seqüência, pois será este que verificará a necessidade dos métodos à implementação do sistema.

Nas próximas figuras estão relacionados os diagramas de seqüência que demonstram as principais funcionalidades do sistema.

Na Figura 7 são demonstrados os passos para a reserva de um veículo, que mostra a interação do usuário com o sistema, e os métodos utilizados para a realização da reserva.

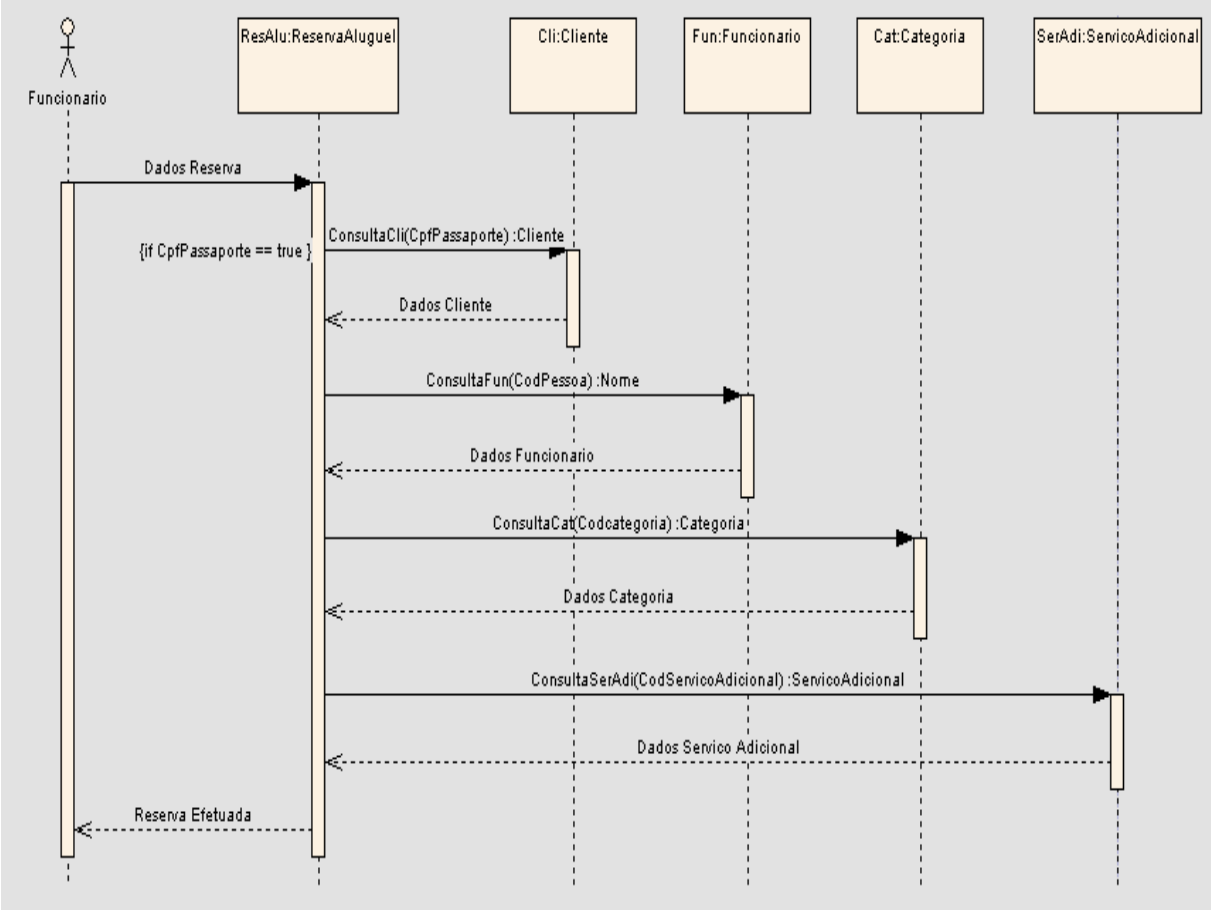


Figura 7– Modelagem do Diagrama de Seqüência (Reservar Carro).

O objeto AluRes que é uma instancia do ReservaAluguel recebe um pedido para realização de uma reserva, esse objeto envia uma mensagem ao objeto Cli que é uma instância de Cliente para validação do cliente (usuário), onde o objeto AluRes aguarda uma resposta.

Sendo verdadeiro o retorno, o objeto AluRes envia uma mensagem para o objeto Fun que é uma instancia de Funcionário e fica aguardando o retorno, que no caso é o funcionário responsável pela reserva. Selecionado o funcionário, o objeto AluRes envia uma mensagem para o objeto Cat que é uma instancia de Categoria e fica aguardando o retorno, neste o retorno será o tipo de categoria de carro que o cliente deseja reservar, e por fim o objeto AluRes envia uma mensagem para o objeto SerAdi que é uma instancia de ServicosAdicionais e fica aguardando um retorno, este retorno pode ser verdadeiro ou falso, pois serviços adicionais são opcionais que o cliente (usuário) pode ou não optar em comprar, e por se tratar de uma reserva não se seleciona o carro a ser reservado, somente a categoria de carros.

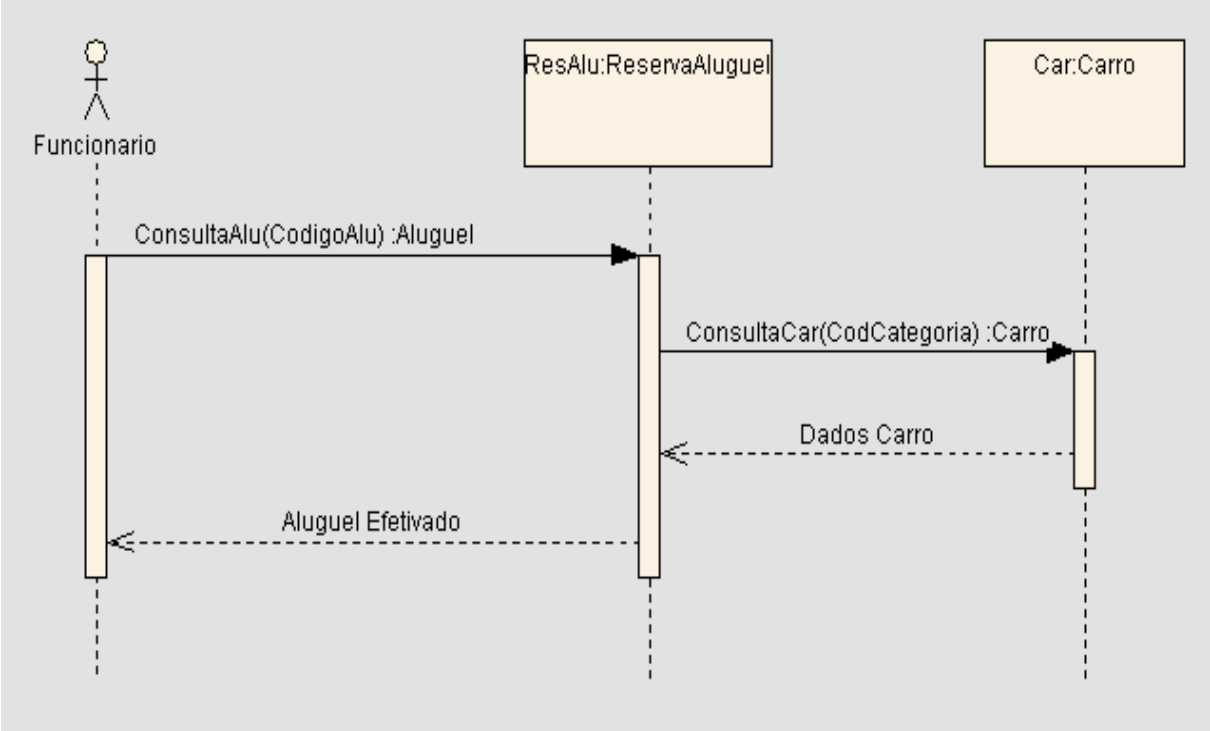


Figura 8– Modelagem do Diagrama de Seqüência (Alugar Carro).

Na Figura 8 é mostrado o objeto AluRes que é uma instância do ReservaAluguel que recebe um pedido de um funcionário para confirmação da reserva, se verdadeiro o objeto AluRes envia uma mensagem para o objeto Car que é uma instância de Carro onde o objeto AluRes aguarda uma resposta, onde esta é o carro a ser escolhido pelo cliente (usuário), o objeto AluRes retorna uma mensagem para o funcionário informando a efetivação do aluguel.

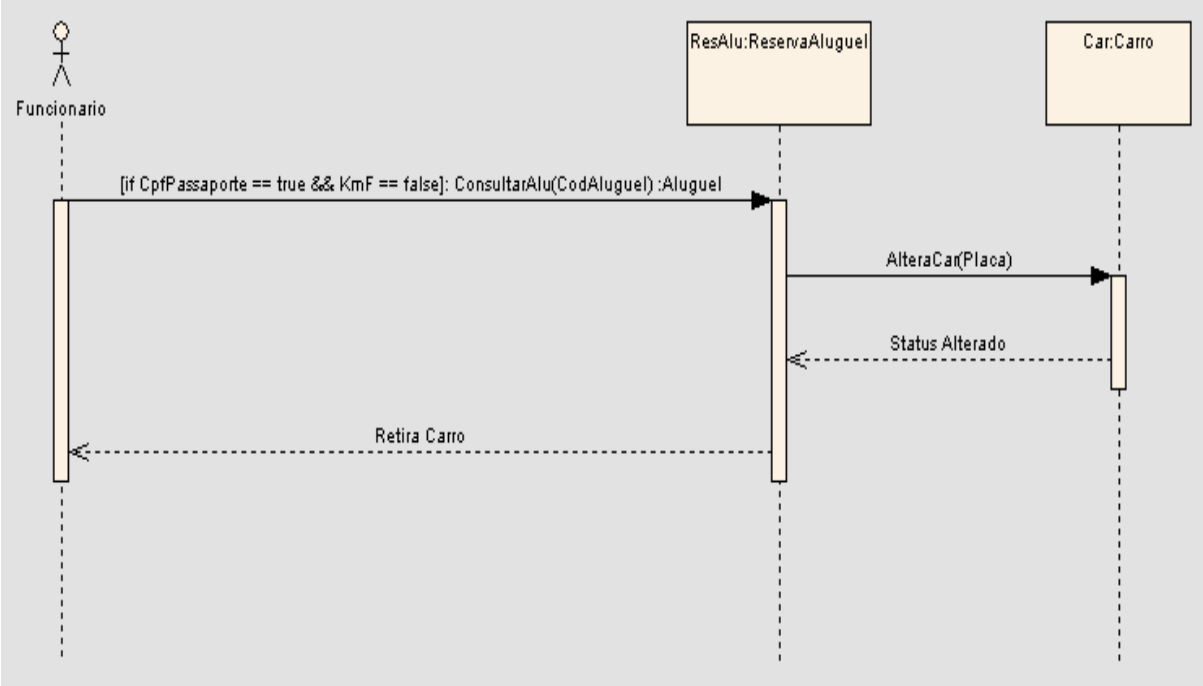


Figura 9– Modelagem do Diagrama de Seqüência (Retirar Carro).

Na Figura 9 é mostrado o objeto AluRes que é uma instância do ReservaAluguel que recebe um pedido de um funcionário para confirmação do aluguel, se verdadeiro o objeto AluRes envia uma mensagem para o objeto Car que é uma instância de Carro onde o objeto AluRes aguarda uma resposta, onde esta é o carro que foi escolhido pelo cliente (usuário) e também a alteração do status do carro, onde na retirada a placa do veiculo escolhido ficara marcada como “Alugado”, o objeto AluRes retorna uma mensagem para o funcionário informando a retirada do carro.

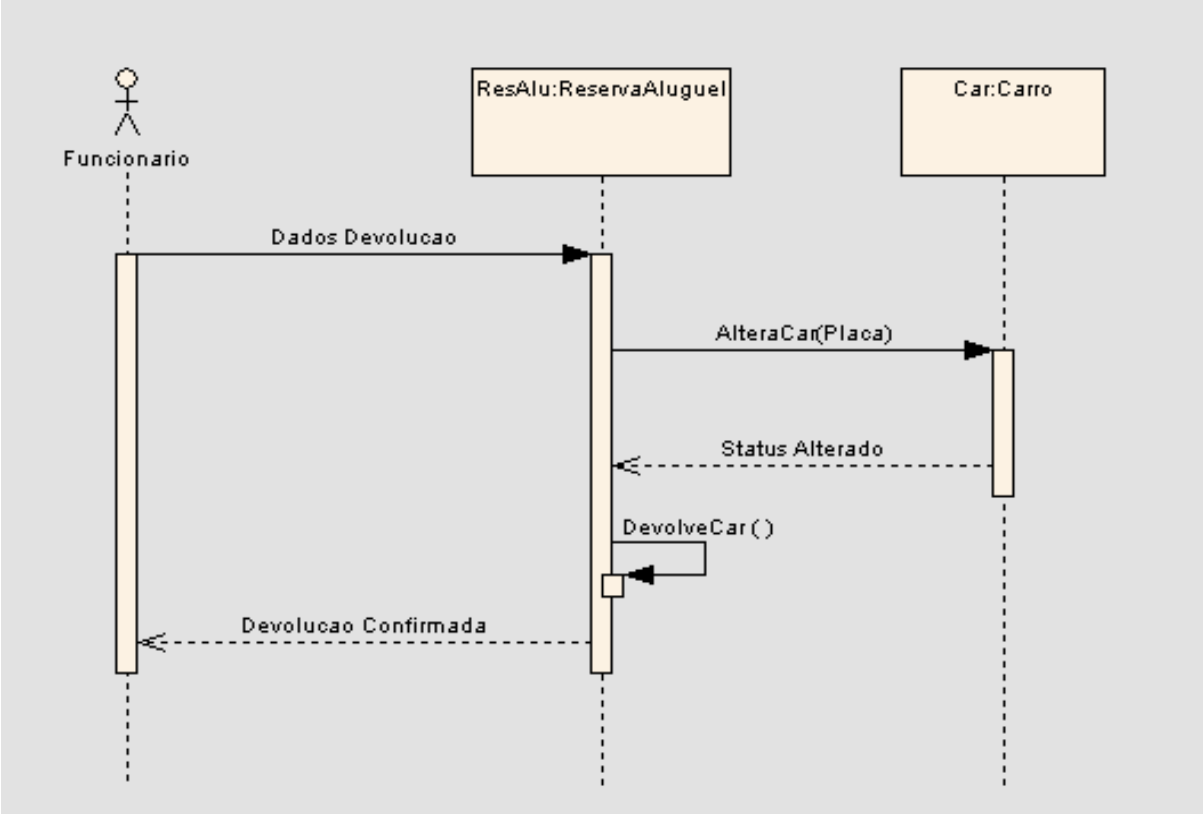


Figura 10– Modelagem do Diagrama de Seqüência (Devolver Carro).

Na Figura 10 é mostrado o objeto AluRes que é uma instância do ReservaAluguel que recebe um pedido de um funcionário para devolução do carro, se verdadeiro o objeto AluRes envia uma mensagem para o objeto Car que é uma instância de Carro onde o objeto AluRes aguarda uma resposta, e modificado o status do carro, onde a placa do veiculo alugado ficara marcada como “Disponível”, é também verificado o estado do carro, verifica-se neste momento se o carro contem as mesmas características do momento que foi liberado para o cliente, do contrario será cobrado pelos danos, o objeto AluRes recebendo a mensagem do objeto CAR, verifica o fechamento do aluguel computando todos os dados pertinentes a devolução, somente após o termino desta verificação que será retornada uma mensagem para o funcionário informando a devolução do carro.

5. IMPLEMENTAÇÃO DO BANCO DE DADOS ORIENTADO A OBJETO

Será abordada agora a implementação dos conceitos de banco de dados Oracle. As figuras a seguir ilustram a implementação do diagrama de classes (Figura 6) que foi realizada na ferramenta Oracle 9i.

Na Figura 11 é ilustrada a criação do Objeto OT_Aluguel no Oracle, um tipo objeto contém três informações: nome, atributo(s) e método(s).

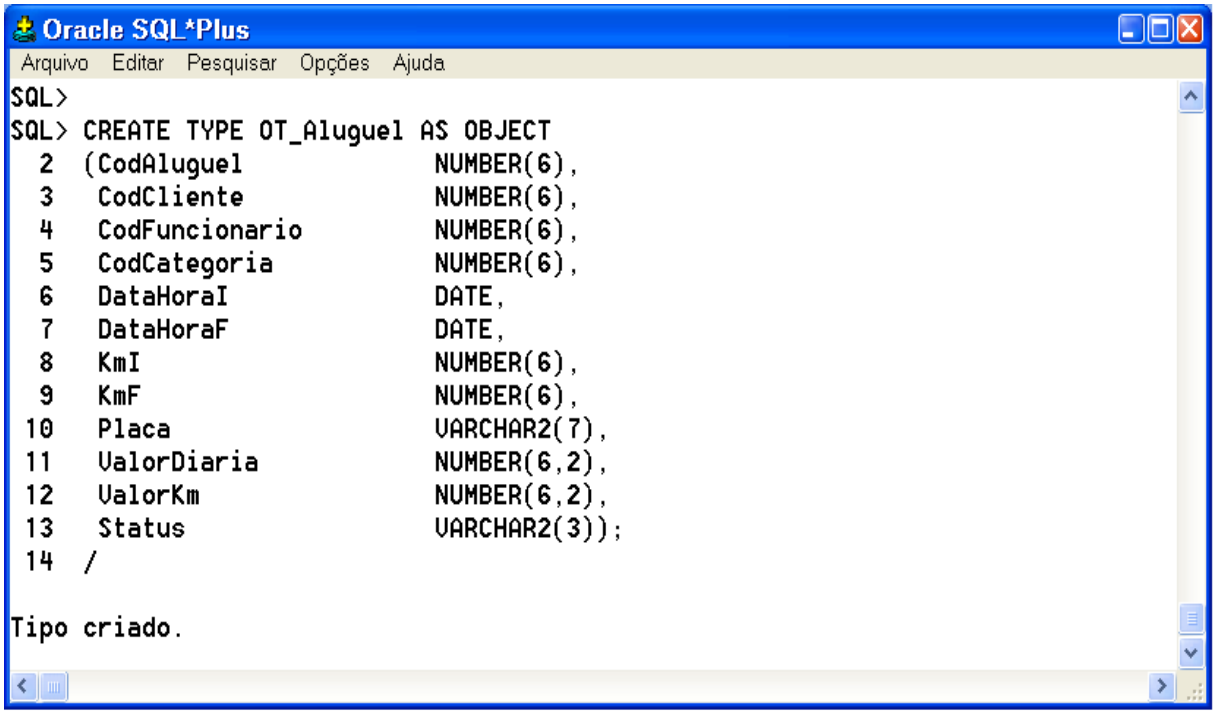


Figura 11– Criação do Objeto Aluguel.

A classe ou tipo de objeto corresponde a um molde e não ao objeto em si. No banco de dados este conceito está perfeitamente adequado, isto é, não podemos incluir dados em um tipo de objeto.

Quando se trata deste assunto baseado em um programa, a criação e armazenamento do objeto com as características definidas pelo tipo OT_Pessoa é feita em memória, o que é extremamente volátil. Em um banco de dados, no entanto, é necessário armazenar os objetos. Sendo assim, precisamos de um “repositório” para armazenamento de objetos.

5.1. Método Construtor em Banco Oracle

Quando se trata de um tipo OT_Pessoa, não se pode, simplesmente, dar um valor diretamente a ele, pois ele é composto de diversos atributos. O método construtor é um método especial que é capaz de construir objeto. Quando um tipo de objeto é criado, implicitamente é criando simultaneamente um método capaz de construir objetos deste tipo. Esta ação tanto é feita nas linguagens OO quanto no banco de dados. Para que o banco de dados identifique um dos métodos da classe como construtor, existem duas regras:

O nome do método construtor deve ser exatamente, igual ao nome da classe. Pode haver mais de um método construtor para a mesma classe.

Conforme a Figura 12, para resolver a necessidade de armazenamento do objeto, o Oracle criou um novo tipo de tabela no banco de dados. Uma *Object Table* é uma tabela para armazenamento das instâncias de uma determinada classe. Uma instância de uma classe é o resultado da construção de um elemento com características definidas pela classe, ou seja, da construção de um objeto. Uma Object Table é uma tabela para armazenamento de objetos.

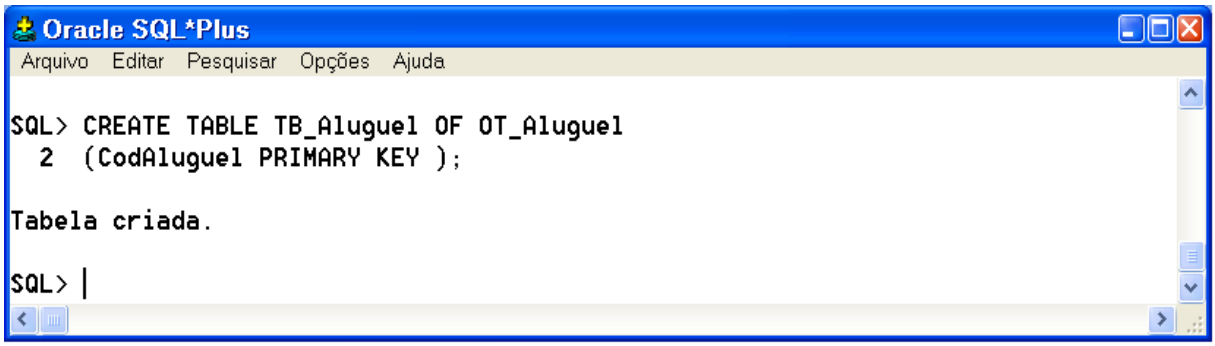


Figura 12– Criação do Objeto Tabela Aluguel.

5.2. Inclusão - Método Construtor em Banco Oracle

Na Figura 13, é acionado o método construtor OT_Aluguel (mesmo nome do objeto) a fim de construir o objeto. Para este método são informados os valores referentes a todos os componentes individuais do objeto.

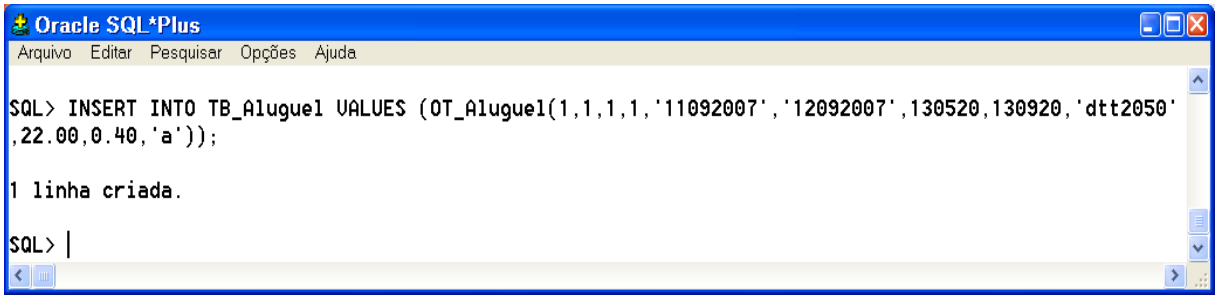


Figura 13– Inclusão do Objeto na Object Table.

5.3. Implementação de Herança em Banco Oracle

Na Figura 14, o tipo OT_Pessoa é um supertipo. Suas características serão herdadas pelos subtipos criados (OT_Cliente, OT_Funcionario).

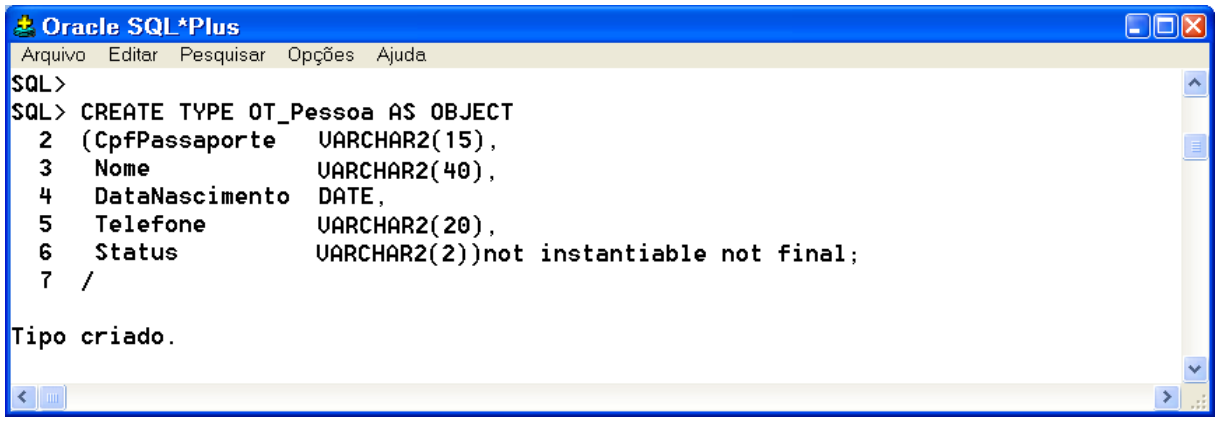


Figura 14– Criação da Classe Pai.

A expressão NOT FINAL anexada à descrição do tipo OT_Pessoa indica que poderemos criar subtipos a partir deste supertipo. Quando não for indicado nada, significa que este tipo não poderá ser herdado por nenhum outro tipo criado. O default é FINAL.

Na Figura 15 é mostrada que tanto OT_Funcionario quanto OT_Cliente, vão herdar todos os atributos do Objeto OT_Pessoa.

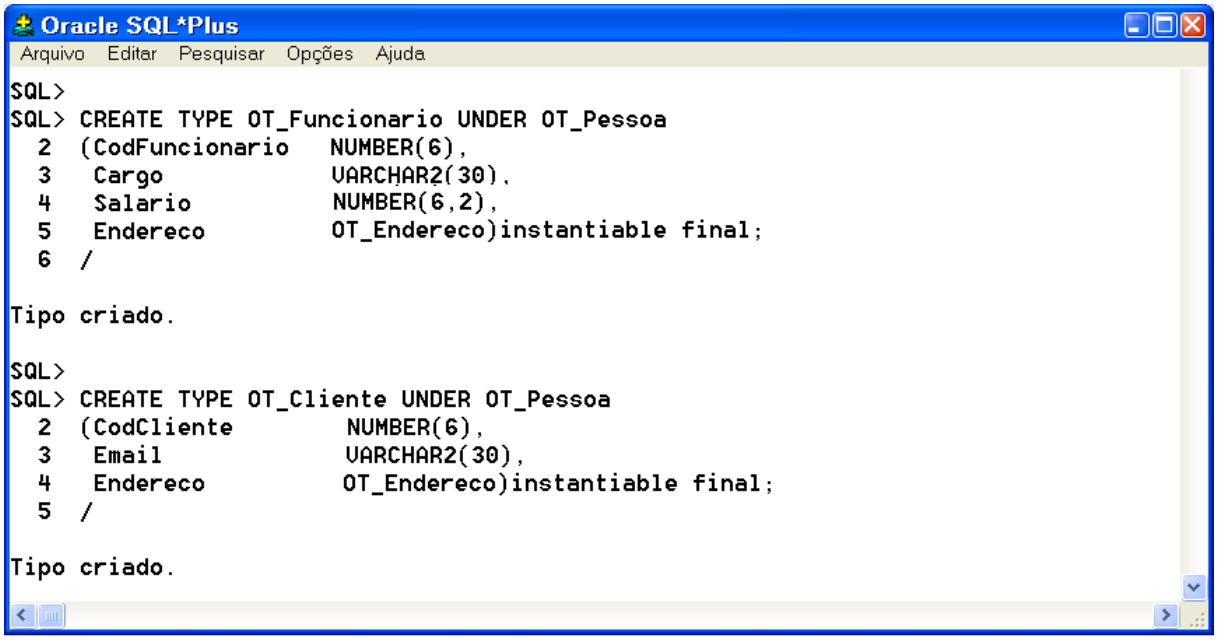


Figura 15– Criação das Classes Filhas.

A expressão Not Instantiable deve ser usada no supertipo, pois indica que o método não foi implementado, ou seja, é uma abstração. A implementação deverá ser feita nos subtipos. Esta expressão deve e somente pode ser usada para tipos de objeto Not Instantiable.

5.4. Implementação de Composição em Banco Oracle

Na Figura 16, tanto OT_Funcionario quanto OT_Cliente, implementou um atributo do tipo OT_Endereco, isto permite o reuso através da composição, permite que se crie tipos simples e que estes sejam usados para a montagem de tipos complexos.

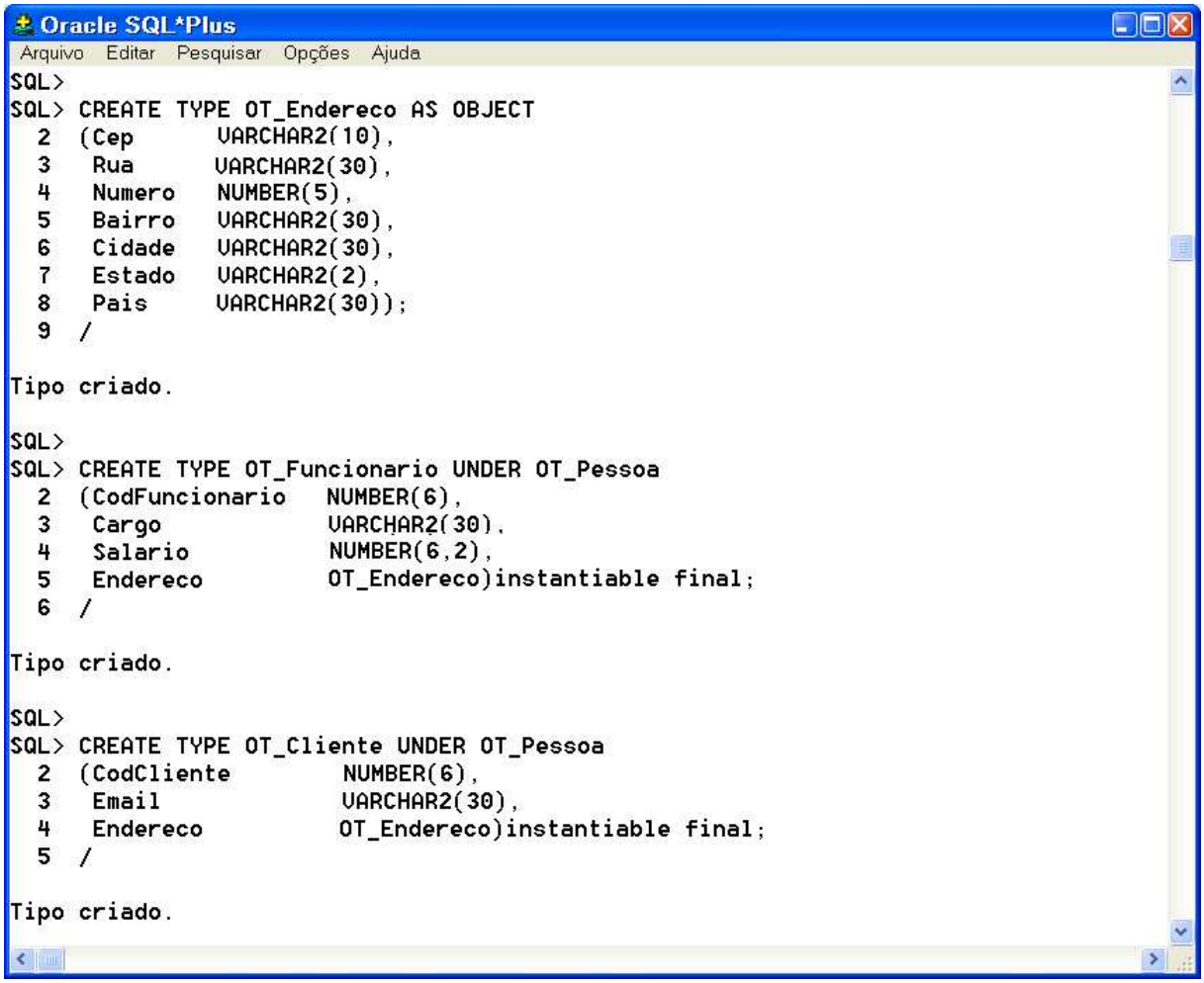


Figura 16– Utilizando Composição no Banco.

Como exemplo de inserção de dados nos objetos a tabela abaixo exemplifica como que são inserido os dados, esta inserção contém tanto Herança quanto a composição apresentada na Figura 15 e 16.

Nota-se na Figura 17, que mesmo sendo uma herança, a inserção é feita da mesma forma que na Figura 13, aciona-se o método construtor OT_Fucionario (mesmo nome do objeto) a fim de construir o objeto, onde são informados todos os valores referentes aos componentes individuais do objeto, após ter preenchido o objeto OT_Funcionario, é acionado o método construtos OT_Endereco, onde serão inseridos todos os valores referentes deste objeto, fazendo com isso uma composição, onde OT_Endereco só existe se OT_Funcionario ou OT_Cliente existirem. O mesmo procedimento para inclusão no objeto se repete para OT_Cliente.

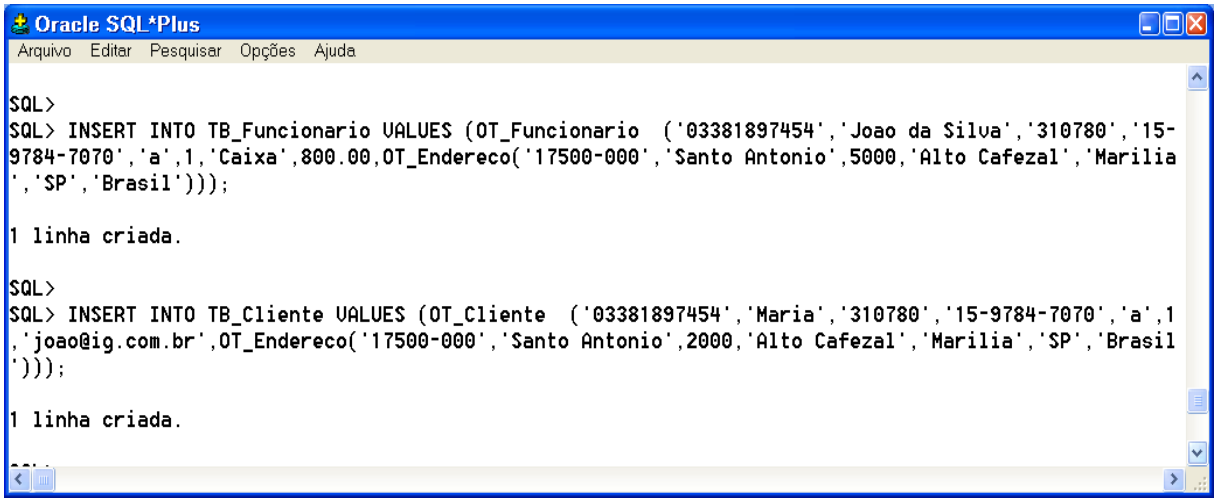


Figura 17– Inserção de Dados nos Objetos.

Na Figura 18 é exibida uma busca simples no banco, simplesmente manda listar todos os clientes cadastrados, e a resposta é todos os atributos do objeto OT_Pessoa referentes a cliente, todos do OT_Cliente e todos do OT_Endereco, porém estes respectivos ao cliente.

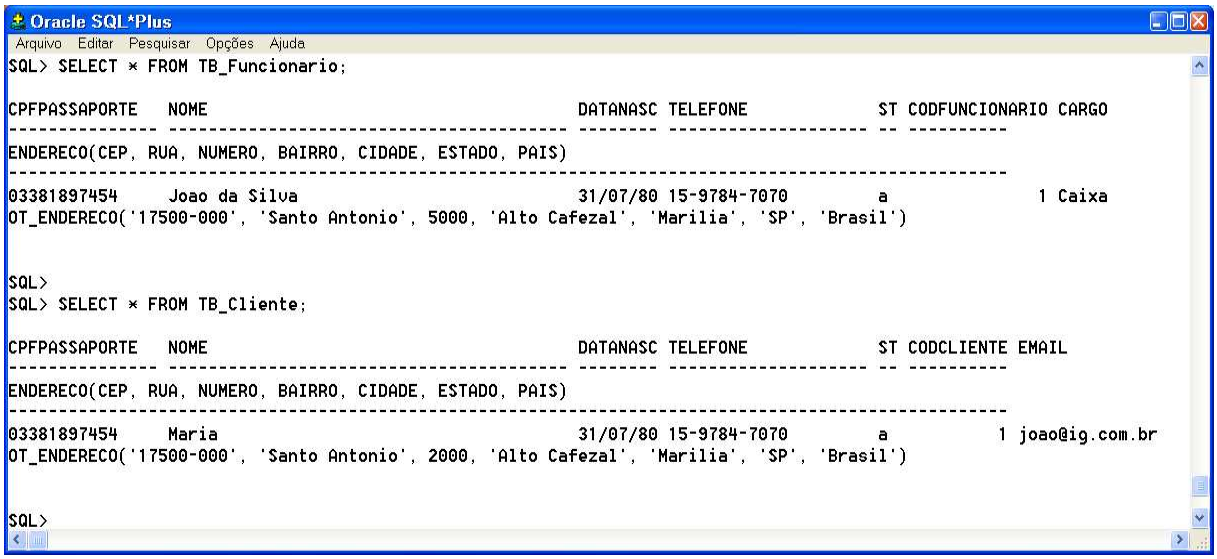


Figura 18– Consulta simples dos Dados nos Objetos.

6. IMPLEMENTAÇÃO DOS MÉTODOS NO BANCO DE DADOS ORACLE 9I.

Estão listados a seguir os métodos que foram implementados no próprio banco de dados, sendo que um futuro aplicativo, poderá apenas invocar a chamada destes métodos no banco, e terá como retorno o resultado esperado.

ConsultaAniversario(CodCliente):Cliente

ConsultaCli(CodCliente):Cliente

6.1. Criação da Função no Objeto.

Na Figura 19 são criadas duas funções, ConsultaCli e ConsultaAniversario dentro do banco de dados a fim de retornar para o aplicativo o resultado já pronto, O aplicativo não terá que fazer nada a mais do que invocar o método ConsultaCli, ou ConsultaAniversario e terá como resposta o resultado desejado.

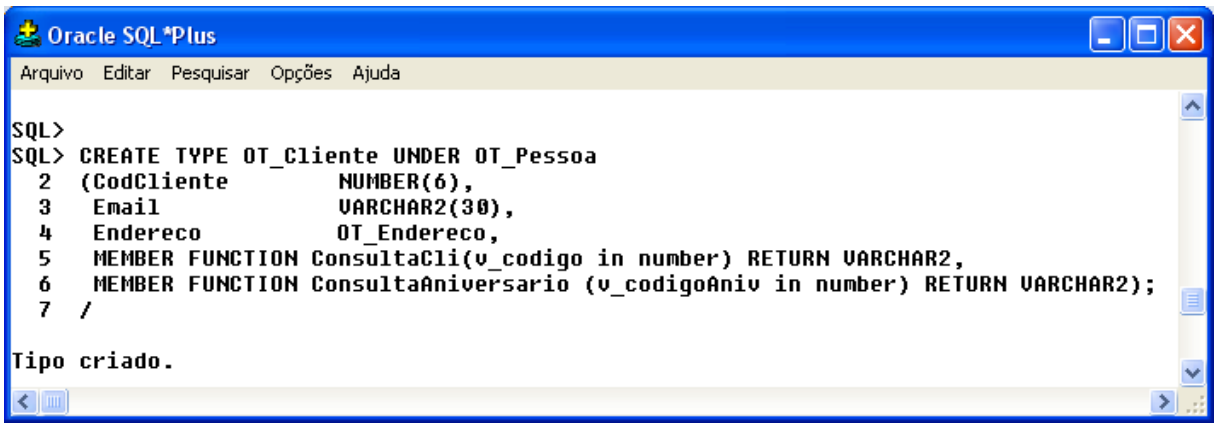


Figura 19– Declarando as Funções Dentro do Objeto Cliente.

Na Figura 20 esta descrito o código das funções declaradas na criação do objeto OT_Cliente, que realizarão a consulta dentro do objeto OT_Cliente.

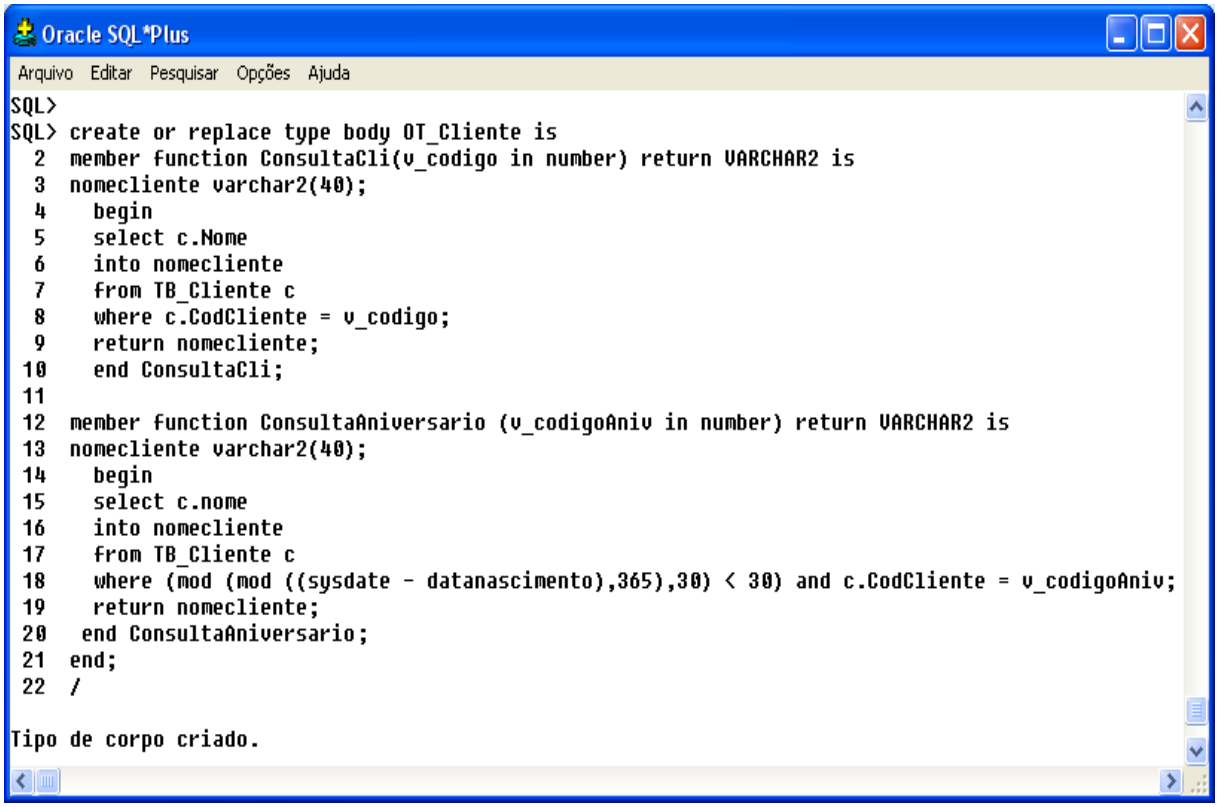


Figura 20– Criação das funções dentro do Banco de Dados.

O Método ConsultaCli varre o objeto TB_Cliente que é uma tabela para armazenamento de objetos até encontrar o cliente cujo o código é passado por parâmetro, esta é uma função utilizada quando se necessita consultar um cliente no aplicativo e que este retorne o nome (como no exemplo) ou outro dado qualquer que se deseja saber, esta função pode ser implementada para as mais diversas consultas de dados dentro do Banco.

O Método ConsultaAniversario retorna o cliente que fará aniversario dentro de 30 dias, ele varre o objeto TB_Cliente até achar o cliente, cujo o código é passado por parâmetro, do contrario caso não exista cliente com data de aniversário próximo a 30 dias, retornara falso.

Na Figura 21 é apresentado o resultado da chamada de cada função criada na Figura 20, como já falado, tanto a consulta de cliente como a de aniversariante necessita de um parâmetro para busca (utilizado na consulta o Código 1).

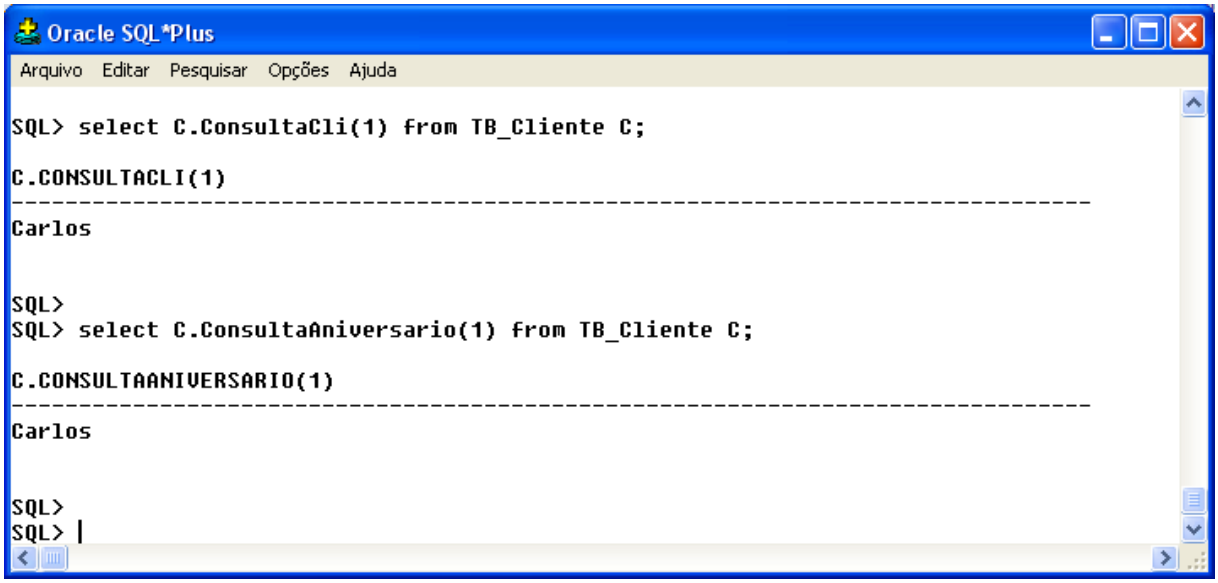


Figura 21– Consulta de Dados Através da Chamada das Funções Criadas.

Estes foram somente dois exemplos de funções que podem ser implementadas dentro de um banco de dados, pode haver quantas funções forem necessário, para busca de informações no banco, estas funções podem ser utilizadas não somente para busca de dados já armazenados como também para busca de valores que dependam de vários cálculos entre vários atributos do banco, retornando o resultado esperado, e como são implementados diretamente no banco diminui significativamente o tempo de busca e calculo de dados e simplifica o código do aplicativo.

7. CONCLUSÃO

Os serviços gerenciadores de banco de dados orientado objeto têm sido desenvolvidos, principalmente, para modelos e aplicações altamente dinâmicas que manuseiam grandes e complexos objetos estruturados e que apresentam, freqüentemente, modificações tanto no seu valor quanto em sua estrutura.

Buscam integrar software desenvolvido em linguagem de programação orientada a objetos, não precisando realizar adaptações para que ambos se comuniquem, pois os bancos de dados tradicionais apresentam algumas limitações quando embutidos em aplicações de software que foram desenvolvidas em uma linguagem de programação orientada a objetos.

A UML é muito útil para uma ótima modelagem do banco, pois é possível verificar todas as funcionalidades do sistema, estudar a integração do usuário, a composição das classes bem como seus relacionamentos e principalmente as funções que o sistema devera desempenhar. Esta modelagem nos permite uma documentação do sistema, maior transparência e facilidade na manutenção.

Pelo fato do modelo procedural ser altamente dependente do modelo relacional em que os dados existem, é difícil de manter e evoluir funções, pois a consistência e a coesão dos programas são fracas. Baseado nesta deficiência do banco relacional o SGBDOO possibilitam uma grande dinâmica nos aplicativos por dar suporte a funções criadas dentro do próprio banco, estas funções são executadas em baixo nível tendo um tempo de execução bem menor que outra criada no aplicativo. O aplicativo simplesmente invoca o método existente no banco e este fará todo o processo de busca no qual é possível “N” operações dentro de uma função, retornando para o aplicativo somente o resultado desejado. Esta técnica facilita o trabalho do programador, pois permite redução significativa do código no aplicativo, maior velocidade nas informações e também maior facilidade na manutenção e transparência das funcionalidades. Uma propriedade importante do objeto, é que todo objeto tem identidade (OID) um endereço na memória que permite ser único, como na criação do objeto, herança e na composição, ao contrário do modelo relacional que depende da comparação de chaves para se relacionar.

Pela inúmera variedade de ferramentas existentes hoje no mercado a tecnologia de banco de dados orientado a objeto ainda é pequena perto do modelo relacional, pois a cultura empresarial e a falta de técnicos capacitados contribuem bastante para sua não aquisição.

Apesar das inúmeras vantagens há ainda grandes obstáculos a serem superados pelo modelo orientado a objeto. Uma característica de um objeto de negócio é manter o seu estado para uma posterior recuperação, que muitas vezes decorre de uma necessidade do fluxo de negócio onde os dados devem ser armazenados para uma posterior analise de valor (estatística).

Sobre este fato o modelo OO é relativamente pobre, uma das maneiras de se alcançar esse objetivo é utilizar o modelo Objeto Relacional e fazer com que este utilize todas as técnicas estudadas sobre SGBDOO além das próprias técnicas do modelo Objeto Relacional.

REFERÊNCIAS BIBLIOGRAFIAS

BERTINO, E., Martino, L. Object-oriented database management systems: concepts and issues. IEEE Computer 1991.

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. UML: Guia do Usuário. Rio de Janeiro: Campus, 2000.

BOSCARIOLI, C.; BEZERRA, A.; BENEDICTO, M.; DELMIRO, G. Uma reflexão sobre Banco de Dados Orientados a Objetos. IV CONGED(Congresso de Tecnologias para Gestão de Dados e Metadados do Cone Sul). Paraná, 2006.
Disponível em: <HTTP://angel.deinfo.uepg.br/artigo4.pdf>. Acesso em 04 set. 2007.

CESAR, Luis. UML – Linguagem de Modelagem Unificada. Universidade Ibirapuera: São Paulo, 2001.
Disponível em:<HTTP://br.geocities.comanaliset43Material_UML.pdf>. Acesso em 04 set. 2007.

DATE, C.J. Introdução a Sistemas de Banco de Dados. Rio de Janeiro: Campus, Editora, 2000. Tradução da 7º Edição Americana. Rubens de Carvalho Sousa, Flávio Cachê, Banco de Dados Pós-Relacional, 2005

FERNANDES, Lúcia. Oracle 9i para Desenvolvedores ORACLE DEVELOPER 6i Curso Completo. Rio de Janeiro: Axcel Books do Brasil Editora, 2002.

FURLAN, José Davi. Modelagem de Objetos através da UML- the Unified Modeling Language. São Paulo: Makron Books, 1998.

GALANTE, Renata de Matos; EDELWEISS, Nina. Bancos de Dados Orientado a Objetos e Relacional-Objeto, 2003.
Disponível em: <http://www.cultura.ufpa.br/clima/bdoo/galante.pdf>. Acesso em 04 set. 2007.

MATOS, Alexandre Veloso de. UML Prático e Descomplicado. São Paulo: Érica, 2002.

SILBERSCHATZ, Abraham. Sistema de Banco de Dados. 3ª ed. São Paulo: Makron Books, 1999.

SILVA, Ricardo Pereira. UML2 em Modelagem Orientada a Objetos. Visual Books, Editora, 2007.

SOUZA, Juliana Pereira. Análise de Requisitos e Projeto Orientado a Objeto usando UML: Um Estudo de Caso. 2005. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Fundação de Ensino “Eurípides Soares da Rocha” Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2005.