UNIVERISIDADE FEDERAL DE UBERLÂNDIA

TRABALHO DA DISCIPLINA BANCO DE DADOS II

BANCO DE DADOS ORIENTADO A OBJETO
BANCO DE DADOS MULTIMIDIA
BANCO DE DADOS PARA DISPOSITIVOS MOVEIS

Leonardo de sousa menezes junior
Ingrid Iplinsky de Oliveira
Bianca Messias Ataide

LISTA DE SIGLAS

SQL	**Structured Query Language**
BDOO	**Banco de dados Orientado a Objeto**
BDO	**Banco de dados de Objeto**
SGOO	**Sistema de Gerenciamento de Dados Orientado a Objeto**
ODMG	**Object Data Management Group**
ODL	**Object Definition Language**
OQL	**Object Query Language**
OO	**Orientado a Objeto**

SUMÁRIO

1. Introdução

O Presente trabalho visa elucidar e trazer alguns conhecimentos sobre outros tipos de bancos de dados além do relacional. Serão abordados três tipos de banco de dados, passando por uma visão sobre temáticas pertinentes a cada um deles e exemplificando algumas questões: Bando de dados Orientado à Objeto, Banco de Dados para Dispositivos Móveis e Banco de Dados Multimídia

2. BANCO DE DADOS ORIENTADO À OBJETO.

Banco de dados orientado a objeto é um tipo de modelagem para construção de banco de dados, neste capítulo apresentaremos como este conceito surgiu e para qual finalidade ele foi criado. Também serão abordados os conceitos básicos de orientação a objeto para que possamos compreender como o sistema funciona. Apresentaremos os modelos de Sistema de Gerenciamento de Dados Orientados a Objetos disponíveis e para finalizar construiremos um modelo de BDOO.

2.1 Como surgiram os Banco de Dados Orientado a Objeto

Os bancos de dados orientados a objeto (BDOO) ou comumente chamado de banco de dados de objetos começam a surgir na década de 80, à medida que o conceito de programação orientada a objeto ganha espaço na computação, o surgimento de estruturas de dados mais complexas que precisam ser armazenados, novos tipos de dados, transações de maiores durações e as necessidades de operações especificam para a aplicação. Se faz necessária um novo modelo para a modelagem de banco de dados e a criação de Sistemas de Gerenciamento de Dados de Objeto (SGDO) que suporte as características da aplicação e da linguagem orientada a objeto. [4, 5]

Algumas áreas que utilizam os bancos de dados de objetos são a medicina, multimídia, física elevada, projeto de engenharia, experimentos científicos, telecomunicações e sistemas de informações geográficas.

As novas aplicações possuem requisitos e características que diferem das aplicações de negócios tradicionais. Os bancos de dados de objeto foram desenvolvidos para atender a algumas dessas necessidades das aplicações mais complexas. Um recurso importante dos bancos de dados de objeto é o poder que eles dão ao projetista para especificar tanto a estrutura dos objetos quanto as operações que podem ser aplicadas a esses objetos. O banco de dados de objetos pode ser integrado diretamente a sistemas desenvolvidos usando a linguagens de programação orientadas a objetos, assim pode se evitar problemas de divergência de impedância. [1]

Atualmente temos dois tipos de desenvolvimentos de banco de dados de objetos, os “puros” e o objeto-relacional. Os bancos de dados de objetos “puros” segue o padrão desenvolvido pelo ODMG (Object Data Management Group) usam a linguagem ODL (Object Definition Language) e OQL (Object Query Language). E os bancos de dados objeto-relacional é uma derivação do modelo relacional, as empresas que gerenciam Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR) viu a necessidade de incorporar recursos que foram propostos para bancos de dados de objeto, e versões mais novas dos SGBD utilizam estes recursos, conhecida como SQL/Foundation.

2.2 Conceitos e características do Banco de Dados Orientado a Objeto

Os bancos de dados orientados a objeto incorporaram muitos dos conceitos que foram criados originalmente para as linguagens de programação orientadas a objeto, como identidade de objeto; estrutura de objeto e construtores de tipo; encapsulamento de operações; definição de métodos como parte das declarações de classe; mecanismos para armazenar objetos em um banco de dados, tornando-os persistentes, e hierarquias e herança de tipo e classe. [1, 2]

O banco de dados de objetos possui três pilares principais: herança, polimorfismo e encapsulamento. Os BDOO apresentam maior flexibilide na manipulação de seu conteúdo e por meio de identificadores de objetos manipula os dados de forma consistente. Os mecanismos como herança e polimorfismo permitem acrescentar novas operações e atributos a classes já existentes, possibilitando a escrita de código reutilizável e extensível.

O paradigma da orientação a objetos possui uma maneira própria de representar um problema. Essa maneira difere muito da forma tradicional de modelagem de dados utilizada pelos bancos de dados relacionais, apesar de guardar algumas semelhanças, sobretudo, relativas à cardinalidade das relações entre as entidades. A modelagem de um BDOO está diretamente relacionada com os diagramas de classes criados para a aplicação.

2.2.1 Objetos

Em um sistema orientado a objetos, cada objeto representa cada entidade do mundo real. Cada objeto possui um estado e comportamentos e é identificado por um indicador único (OID ou object identifier). Ultimamente tem utilizado inteiros longos como OIDs e depois se usa tabela de hash para mapear o valor do OID ao endereço físico atual do objeto no local de armazenamento.

O estado do objeto depende do valor de cada uma de suas propriedades e o comportamento é especificado pelas operações que podem ser executadas pelo objeto, que permite a alteração de estado do mesmo. As propriedades podem ser atributos ou relações entre esse objeto e outros objetos que façam parte do domínio do problema. Uma das propriedades mais importantes do objeto é justamente sua identidade o OID. Sem ele, não é possível saber com qual objeto se comunicar, seja para obter ou atualizar dados. Normalmente, um objeto deve assumir uma identidade no momento de sua criação e permanece com ela durante toda a sua existência.[3, 4]

O objeto é composto por três informações (i, c, v), onde o i é o OID do objeto, o c é um construtor, ou seja, que tipo de valor ele pertence, como, atom, tuple, set, list, bag, array e v é o valor corrente. Logo o objeto passa a suportar aquilo que foi definido para ele. Se ele vai receber um valor atômico ele só aceitará valores atômicos.

Nos bancos de dados orientados a objeto há três tipos de construtores básicos são átomo, struct (ou tupla) e coleção. [1]

1. Construtor do tipo átomo correspondem aos tipos de dados básicos como, inteiros, cadeias de caracteres, números de ponto flutuante, tipos enumerados, booleanos, e assim por diante. Eles são denominados tipos de valor único ou atômicos, pois cada valor do tipo é considerado um único valor atômico (indivisível).
2. Construtor do tipo struct correspondem aos valores do construtor de tipo tupla.
3. Construtor de coleção especificam valores que são coleções de objetos ou valores, como set, bag, list, array e dictionary.
    a) O construtor de coleção do tipo set criará objetos que são um conjunto de elementos distintos.
    b) O construtor de coleção do tipo bag é semelhante ao set, porém os elementos do conjunto não precisam ser distintos.

c) O construtor de coleção do tipo list criará uma lista ordenada.
d) O construtor de coleção do tipo array criará um vetor unidimensional de elementos do mesmo tipo. A principal diferença entre array e list é que uma lista pode ter um número arbitrário de elementos, enquanto um array normalmente tem um tamanho máximo.
e) O construtor de coleção do tipo dictionary cria uma coleção de duas tuplas (k, v), onde o valor de uma chave k pode ser usado para recuperar o valor correspondente v.

2.2.2 Encapsulamento de Operações e Classe de Objetos

O conceito de encapsulamento é uma das principais características da linguagem orientada a objeto. Ele está relacionado aos conceitos de tipos de dados abstrato e ocultação de informação. Logo o desenvolvedor pode decidir quais informações é relevante o usuário externo ter conhecimento através das interfaces ou também chamadas de assinaturas onde especifica o seu nome e seus parâmetros e os detalhes da implementação, que são os métodos que ficam ocultos. A estrutura de um objeto é dividida em atributos visíveis e ocultos.[1, 5]

O termo classe é frequentemente utilizado para se referir a uma definição de tipo, junto com as definições das operações para esse tipo. Em que a partir de uma classe podemos criar subclasses e encapsular operações.

Algumas operações típicas criadas na classe são:

- Construtor de Objeto.
- Construtor de Destruição.
- Operações de Modificação.
- Operações para Recuperar Informações sobre um Objeto.

2.2.3 Herança e Hierarquias de Tipo

Outro conceito importante da programação orientada a objeto é o conceito de herança em que permite a definição de novos tipos com base em outros tipos predefinidos. Logo, a herança de tipos nos pode levar a uma hierarquia de tipos.

Um tipo (ou classe) possui um nome, uma lista de atributos (variáveis de instancia) e operações (métodos). Os subtipos herdam todos os atributos e operações do tipo predefinido.

Há dois tipos de herança, elas são:[1, 5]

- Herança Múltipla ocorre quando um subtipo T é um subtipo de dois (ou mais) tipos. T herda as funções dos dois subtipos.
- Herança Seletiva ocorre quando um subtipo herda apenas algumas das funções de um supertipo.

2.2.4 Persistência de Objetos

Há dois tipos de objetos os transientes e os persistentes. Objeto transiente não necessita de ser armazenado, ele existe apenas no momento de execução do programa. Objeto persistente é armazenado e pode retornar ao seu estado quando o programa é reiniciado. Tem dois tipos de mecanismo para conseguir realizar a persistência de objetos no SGDO por nomeação e acessibilidade.[1]

- O mecanismo de nomeação dá a um objeto um nome persistente único dentro de determinado banco de dados. Esse nome de objeto persistente pode receber uma instrução ou operação específica no programa.
- O mecanismo de acessibilidade funciona tornando o objeto alcançável a partir de algum outro objeto persistente. Um objeto B é considerado alcançável com base no objeto A se uma sequência de referências no banco de dados levar do objeto A até o objeto B.

2.2.5 Polimorfismo de Operações

O polimorfismo de operações é a capacidade de uma operação ser aplicada a diferentes tipos de objetos. Um nome de operação pode se referir a várias implementações distintas. [2]

2.3 Exemplos de Orientação a Objeto usando a linguagem ODL e SQL

Nesta seção apresentaremos a representação de dados usando a linguagem ODL no modelo mais simples, na seção 2.5 apresentaremos a linguagem ODL seguindo o padrão ODMG. Também será apresentado o modelo Objeto-Relacional usando o SQL, para que possamos fazer uma comparação das duas abordagem.[1]

2.3.1 Linguagem ODL

O script C2.1 representa a criação dos Objetos Funcionário, Data e Departamento.

C2.1:

```
define type FUNCIONARIO tuple (
        Pnome: string;
        Minicial: char;
        Unome: string;
        Cpf: string;
        Data_nascimento: DATA;
        Endereco: string;
        Sexo: char;
        Salario: float;
        Supervisor: FUNCIONARIO;
        Dep: DEPARTAMENTO;
);

define type DATA tuple (
        Ano: integer;
         Mes: integer;
         Dia: integer;
);

define type DEPARTAMENTO tuple (
        Dnome: string;
        Dnumero: integer;
        Ger: tuple (
                Gerente: FUNCIONARIO;
                 Data_inicio: DATE;
```

```
        );
        Localizacoes: set(string);
        Funcionarios: set(FUNCIONARIO);
        Projetos: set(PROJETO);
);
```

No script C2.2 é apresentado a criação de Operações básicas relacionadas aos Objetos Funcionário e Departamento.

C2.2:

```
operations
        idade: integer;
        criar_Funcionario: FUNCIONARIO;
        destroir_Funcionario: boolean;
end FUNCIONARIO;

operations
        nr_Funcionario: integer;
        criar_Departamento: DEPARTAMENTO;
        destroir_Departamento: boolean;
        alocar_Funcionario (e: FUNCIONARIO): boolean;
        remover_Funcionario(e: FUNCIONARIO): boolean;
end DEPARTAMENTO;
```

No script C2.3 apresentamos a criação da classe Set_Departamento e a persistência de objetos.

C2.3:

```
define class SET_DEPARTAMENTO
        type set (DEPARTAMENTO);
        operations
                adicionar_Departamento(d: DEPARTAMENTO): boolean;
                remover_Departamento(d: DEPARTAMENTO): boolean;
                criar_Set_Departamento: SET_DEPARTAMENTO;
                destroi_set_dep: boolean;
        end SET_DEPARTAMENTO;

        persistent name TODOS_DEPARTAMENTOS: SET_DEPARTAMENTO;

        d:= criar_Departamento;

        b:= TODOS_DEPARTAMENTOS.adicionar_Departamento(d);
```

No script C2.4 são criadas 3 classes Pessoa, Funcionário e Aluno, podemos observar que está 3 classes possuem atributos em comum, no script C2.5 apresentarmos como é realizado a representação de herança na ODL.

C2.4:

PESSOA: Nome, Endereco, Data_nascimento, Idade, Cpf

FUNCIONARIO: Nome, Endereco, Data_nascimento, Idade, Cpf, Salario, Data_contratacao

ALUNO: Nome, Endereco, Data_nascimento, Idade, Cpf, Curso, Coeficiente

C2.5:
PESSOA: Nome, Endereco, Data_nascimento, Idade, Cpf

FUNCIONARIO **subtype-of** PESSOA: Salario, Data_contratacao
ALUNO **subtype-of** PESSOA: Curso, Coeficiente

2.3.2 Usando o conceito de Objeto-Relacional no SQL

Nesta subseção apresentamos o conceito de objeto-relacional usando SQL. No script C2.6 demonstra a criação de tipos.

C2.6:

```
CREATE TYPE  END_RUA AS (
        numero          VARCHAR (5),
        nome_rua        VARCHAR (25),
        nr_apto         VARCHAR (5),
        nr_bloco        VARCHAR (5)
);

CREATE TYPE   END_BRASIL AS (
         end_rua        TIPO_END_RUA,
        cidade          VARCHAR (25),
        cep             VARCHAR (10)
);
CREATE TYPE   TELEFONE_BRASIL AS (
        tipo_telefone   VARCHAR (5),
        codigo_area     CHAR (3),
        num_telefone    CHAR (7)
);
```

No script C2.7 é apresentada uma alternativa para a criação do tipo END_RUA usando a função ROW permite criar um atributo como uma estrutura.

C2.7:

```
CREATE TYPE  END_BRASIL AS (
        end_rua         ROW (
                                numero       VARCHAR (5),
                                nome_rua     VARCHAR (25),
                                nr_apto      VARCHAR (5),
                                nr_bloco     VARCHAR (5)
                        ),
        cidade      VARCHAR (25),
        cep           VARCHAR (10)
);
```

No script C2.8 é feito a criação de tipo PESSOA que será uma super classe . Para permitir que um tipo possa possuir subtipos este deve ser definido como NOT FINAL, por padrão um tipo de objeto é FINAL. INSTANTIABLE permite que os métodos sejam instanciáveis, caso contraria deve ser definido NON INSTANTIABLE.

REF IS SYSTEM GENERATED indica que o identificador do objeto será gerado pelo sistema. É criado o método Idade() o qual tem retorno do tipo integer.

C2.8:

```
CREATE TYPE PESSOA AS (
        nome            VARCHAR (35),
        sexo            CHAR,
        end             END_BRASIL,
        data_nasc       DATE,
        telefones       TELEFONE_BRASIL ARRAY [4],
INSTANTIABLE
NOT FINAL
REF IS SYSTEM GENERATED
INSTANCE METHOD IDADE( ) RETURNS INTEGER;

CREATE INSTANCE METHOD IDADE( ) RETURNS INTEGER
        FOR TIPO_PESSOA
        BEGIN
                RETURN
                ...
         END;
 );
```

As funções podem ser definidas de duas maneiras no SQL internamente e externamente. Funções internas são escritas na linguagem PSM estendida do SQL. Funções externas são escritas em uma linguagem hospedeira (host), com apenas sua assinatura (interface) aparecendo na definição. Uma definição de função externa pode ser declarada da seguinte maneira:

C2.9:

```
DECLARE EXTERNAL <Nome_Função>
<Assinatura> LANGUAGE <Nome_Linguagem>;
```

No script C2.8 é feito a criação de tipo FUNCIONARIO é usado a função UNDER para declarar herança do tipo PESSOA e GERENTE herda o tipo FUNCIONARIO.

C2.10:

```
CREATE TYPE  FUNCIONARIO UNDER   PESSOA AS (
         CODIGO_EMPREGO        CHAR (4),
        SALARIO                FLOAT,
        CPF                    CHAR (11)
 INSTANTIABLE
NOT FINAL
);
CREATE TYPE GERENTE UNDER  FUNCIONARIO AS (
        DEP_GERENCIADO CHAR (20)
 INSTANTIABLE
);
```

No script C2.11 é apresentado a criação de três tabelas TABLE_PESSOA será do tipo PESSO isto é definido através do OF.

C2.11:

```
CREATE TABLE TABLE_PESSOA OF PESSOA
        REF IS ID_PESSOA SYSTEM GENERATED;

CREATE TABLE TABLE_FUNCIONARIO OF FUNCIONARIO
        UNDER TABLE_PESSOA;

CREATE TABLE TABLE_GERENTE OF GERENTE
        UNDER TABLE_FUNCIONARIO;
```

2.4 Características dos SGOO

Há diversos sistemas de gerenciamento de dados orientados a objeto, nesta seção apresentaremos alguns deles, pois para iniciar um projeto é essencial conhecermos as ferramentas existentes para que se possa escolher a melhor que enquadrará ao projeto. A seguir estão alguns exemplos: [2, 3]

1. Caché: permite utilizar as seguintes linguagens: Java, .Net, C++, XML e outras. Pode ser utilizado nos sistemas operacionais Windows, Linux e Unix, é um banco de dados comercial fabricante Intersystems Software.
2. Versant: permite utilizar as seguintes linguagens: Java e C++. É bastante utilizado nos sistemas telecomunicações, redes de transporte, áreas médicas e financeiras. Pode ser utilizado nos sistemas operacionais Windows, Linux e Unix, é um banco de dados comercial fabricante Versant Corporação.
3. DB4Objects: permite utilizar as seguintes linguagens: Java e .Net. Sua linguagem de Consulta é a Object Query Language (OQL) e é um banco de dados distribuído em duas licenças, a GPL (licença pública Geral) e uma licença comercial.
4. O2: permite utilizar as seguintes linguagens: C, C++ e o ambiente O2. Sua linguagem de Consulta: O2Query, OQL. Seu gerenciador do Banco de Dados é o O2Engine, e é um banco de dados comercial O2 Technology..
5. GemStone: permite utilizar as seguintes linguagens: Java, C++, C#, XML e outras. Sua linguagem de Consulta é o DML. Pode ser utilizado nos sistemas operacionais Windows, Linux e Unix, é um banco de dados comercial fabricante GemStone System Incorporação.
6. Jasmine: Possui alta conectividade com Web, suporte à linguagem Java. Pode-se ainda desenvolver aplicações em Visual Basic usando Active/X, em HTML (HyperText Markup Language) usando as ferramentas de conectividade para Web disponíveis no Jasmine, em C e C++ usando APIs e em Java usando interfaces de middleware embutidas no Jasmine. Pode ser utilizado nos sistemas operacionais Windows, Linux e Unix, é um banco de dados comercial fabricante Computer Associates.
7. Matisse: permite utilizar as seguintes linguagens: Java, C#, C++, VB, Delphi, Perl, PHP, Eiffel, SmallTalk. É um banco de dados comercial.
8. Objectivity/DB: permite utilizar as seguintes linguagens: C#; C++; Java; Python, Smalltalk; SQL++ (SQL com objeto - extensões orientadas) e XML (para a importação e a exportação somente). Pode ser utilizado nos sistemas operacionais Windows, Linux e Unix, é um banco de dados comercial fabricante Objectivity Database Systems.
9. EyeDB: pode ser utilizado nos sistemas operacionais Linux e Unix, é um banco de dados open source fabricante Sysra Informatique.
10. ObjectStore: pode ser utilizado nos sistemas operacionais Windows, Linux e Unix, é um banco de dados comercial fabricante Progress Software.

11. Ozone: permite utilizar as seguintes linguagens: Java e XML. É um banco de dados opensource.
12. EnterpriseDB: permite utilizar as seguintes linguagens: C/C++. É um banco de dados opensource.
13. ORION: pode ser utilizado nos sistemas operacionais Linux e Unix, é um banco de dados open source fabricante Orion Group (Purdue University).

2.5 Desenvolvendo um modelo de DBOO usando ODL e OQL nos padrões da ODMG.

Nesta seção apresentaremos um modelo de caso usando a linguagens ODL e OQL seguindo dos padrões ODMG. ODMG (Object Database Management Group) grupo formado pelos principais fabricantes de banco de dados OO além de um grande número de empresas interessadas num padrão para SGOO. O ODMG define padrões para a modelagem de objetos, linguagem de definição de objetos – ODL, linguagem de consulta – OQL, ligações com LPOO, metadados, controle de concorrência e modelo de transações.

Na figura 2.12 é apresentado o diagrama de caso de uso de uma Universidade e o relacionamento dos Objetos. Há três relacionamento de herança, Aluno e Professor herda a classe Pessoa e TurmaCorrente herda a classe Turma.

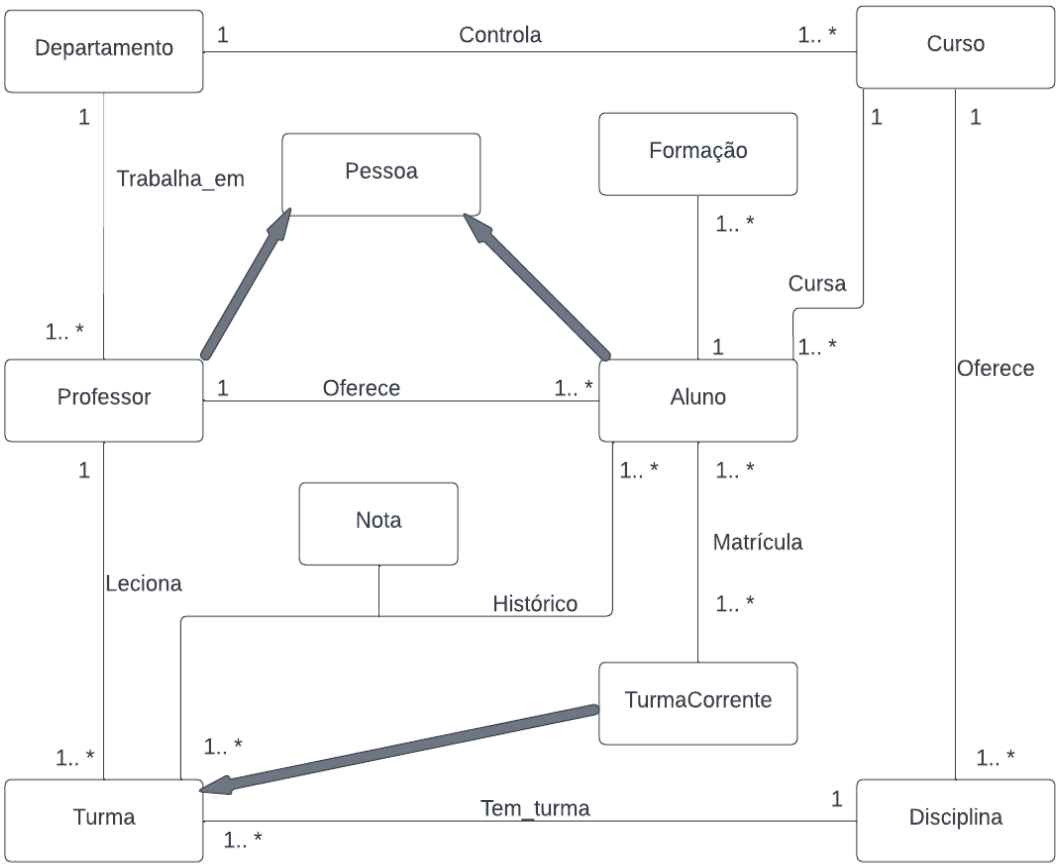


Figura 2.1 - Representação do diagrama de caso de uso de uma Universidade e o relacionamento dos Objetos

2.5.1 Sintaxe da linguagem ODL no padrão ODMG

Nesta subseção é apresentado a criação de todas as classes e métodos. O termo **class** define uma classe, **extent** define que a classe pode ser estendida e **key** cria um atributo de ligação, **extends** define a classe que foi extendida (herdada), **attribute** define os atributos da classe, **relationship** define um relacionamentoe o **inverse** representa o relacionamento inverso **::**, a definir uma função precisa indicar o tipo de retorno e caso tenha exceção pode-se usar a função **raise.**

C2.12:

```
class PESSOA
        ( extent   PESSOAS      key   Cpf ) {
        attribute       struct  Nome {
                                        string Pnome,
                                        string Mnome,
                                        string Unome
                        } Nome;
        attribute       string   Cpf;
        attribute       date    Data_nascimento;
        attribute       enum    Genero{M, F} Sexo;
        attribute       struct  Endereco {
                                        short Nr,
                                        string Rua,
                                        short Nr_apto,
                                        string Cidade,
                                        string Estado,
                                        short Cep
                        } Endereco;
        Short     Idade( );
};

class PROFESSOR
        extends PESSOA ( extent PROFESSORES ) {
                attribute       string  Cargo;
                attribute       float   Salario;
                attribute       string  Escritorio;
                attribute       string  Telefone;
                relationship    DEPARTAMENTO Trabalha_em inverse
                                                        DEPARTAMENTO::Tem_professor;
                relationship    set<ALUNO> orienta  inverse  ESTUDANTE::Orientador;
                relationship    set<TURMA> leciona  inverse  TURMA::Professor;
                void    aumentoSalario(in float aumento);
                void    promocao(in string novo_cargo);
};

 class ALUNO
        extends PESSOA ( extent ALUNOS ) {
                        attribute       string  Matricula;
                        relationship    set<FORMAÇÃO> Formação;
                        relationship    CURSO Cursa   inverse  CURSO::Tem_aluno;
                        relationship    PROFESSOR Orientador  inverse  PROFESSOR::Orienta;
                        relationship    set<NOTA> concluido  inverse  NOTA::Estudante;
                        relationship    set<TURMACORRENTE>  inverse
                                                        TURMACORRENTE::Aluno_matriculado;
                        float   media( );
                        void    matricula( in  short  nr_turma ) raises ( turma_invalida );
```

```
                    void    aloca_nota( in short nr_turma, in double nota)
                                                        raises ( turma_invalida, nota_invalida );
};

class FORMAÇÃO{
        attribute       string  Faculdade;
        attribute       string  Titulo;
        attribute       string  Ano;
};

class CURSO ( extent CURSOS     key  CursoNome ) {
        attribute       string  CursoNome;
        relationship    DEPARTAMENTO  Pertence  inverse    DEPARTAMENTO::Controla;
        relationship    set <ESTUDANTE> Tem_estudante  inverse  ESTUDANTE::Cursa;
        relationship    set < DISCIPLINA> Oferece  inverse  DISCIPLINA::Oferecida;
};

class DEPARTAMENTO ( extent DEPARTAMENTOS    key CodDep ) {
        attribute       string  CodDep;
        attribute       string  Dnome;
        attribute       string  Dtelefone;
        attribute       string  Descritorio;
        attribute       string   Faculdade;
        attribute       PROFESSOR     Diretor;
        relationship    set<PROFESSOR> Tem_Professor  inverse  PROFESSOR::Trabalha_em;
        relationship    set<CURSO> Controla    inverse    CURSO::Pertence_a;
};

class DISCIPLINA ( extent DISCIPLINAS  key CodDisc ) {
        attribute       string  DiscNome;
        attribute       string  CodDisc;
        attribute       string  Descrição;
        attribute       string  CargaHoraria;
        relationship    set<TURMA> Tem_turma  inverse   TURMA::Da_disciplina;
        relationship    CURSO Oferecido_por inverse CURSO::Oferece;
};

class TURMA ( extent TURMAS ) {
         attribute     short    codTurma
        attribute     string    Ano;
        attribute     enum    Periodo {Primeiro, Segundo} Semestre;
        relationship    set <Nota> Estudantes inverse   Nota::Turma;
        relationship    DISCIPLINA Da_disciplina    inverse    DISCIPLINA::Tem_turma;
        relationship    PROFESSOR Professor         inverse    PROFESSOR::Leciona;

};

class TURMACORRENTE extends TURMA ( extent TURMASCORRENTES ) {
        relationship set <ALUNO> Aluno_matriculado    inverse     ALUNO::Matricula_em;
        void    matricular_aluno ( in string Cpf ) raises ( aluno_invalido, turma_cheia );
};

class NOTA
        ( extent NOTAS ) {
```

```
                Attribute       double      Nota;
                relationship    TURMA   Turma   inverse     TURMA::Alunos;
                relationship    ALUNO   Aluno   inverse     ALUNO::Turma_completada;
};
```

2.5.2 Sintaxe de consultas em OQL

Consulta Simples.

C2.13:

```
select D.CodDep
        from D in DEPARTAMENTOS
        where D.Faculdade = ‘Computação;
```

Consulta usando dados do tipo STRUCT e consulta embutida.

C2.14:

```
select struct (
        nome: struct (
                ultimo_nome: A.nome.Unome,
                primeiro_nome: A.nome.Pnome),
                formação:(
                        select struct (tit: F.Titulo, ano: F.Ano, faculdade: F.faculdade)
                                from F in A.Formação
                )
        )
        from D in DEPARTAMENTO.Diretor.Orientados;
```

Consulta usando ORDER BY.

C2.15:

```
select N.Turma.Disciplina.DiscNome,  N.Nota
         from  N in CC_NOTAS
        where  N.Turma.Ano = 2022 and N.Turma.Semestre.Periodo = ‘Primeiro’ and
                                                                N.Estudante.Nome = ‘João’
        order by  N.Nota desc;
```

Criação de Visão.

C2.16:

```
define cursoSI as
        select C
                from C in CURSOS
               where C.NomeCurso = ‘Sistema de Informação’;
```

Usando o LOOP FOR.

C2.17:

```
for all A in ( select A
                from A in ALUNOS
                where A.Cursa.CursoNome = 'Sistema de Informação'
        ) : A.Orientador   in   cursoSI.Tem_professor;
```

Usando a função MAX.

C2.18:

```
max ( select N.Nota
                from N in NOTAS
                where N.Turma.Ano = 2020 and N.Turma.Semestre.Periodo = 'Segundo');
```

Usando a função AVG, GROUP BY, HAVING e COUNT.

C2.19:

```
select nomeCurso, mediaGeral: avg ( select P.A.media() from P in partion )
        from A in ALUNOS
        group by  nomeCurso: A.Cursa.CursoNome
                having  count ( partion ) > 700;
```

3. BANCO DE DADOS PARA DISPOSITIVOS MÓVEIS.

Neste capítulo será apresentado uma breve visão da computação móvel, que deu origem aos Banco de dados para dispositivos móveis, quais os conceitos que se caracterizam como novos desafios dentro dos BD móveis e alguns exemplos de scripts feitos em Oracle Database Lite e SQLite.

3.1 Visão Geral da Computação Móvel

Com os avanços nas relações sociais e econômicas, a transmissão, atualização e acesso aos dados em qualquer lugar e em qualquer horário se tornou não somente um diferencial, mas também uma necessidade.
A transmissão e compartilhamento de informações de maneira rápida, segura e sem comprometer os dados é um dos desafios da computação móvel. Proveniente da união da computação tradicional com tecnologias de comunicação sem fio, a computação móvel, também referida como Computação Ubíqua, pode ser descrita pelo uso de dispositivos móveis e portáteis, sendo os mais comuns notebooks e celulares.

3.1.1 Desafios da Computação Móvel

Desenvolvida a partir de uma necessidade emergente de mobilidade, seus principais empecilhos têm origem em 3 propriedades essenciais: portabilidade, comunicação e mobilidade.

3.1.1.1 Portabilidade

A portabilidade caracteriza o equipamento que apresenta completamente o conjunto de ferramentas necessárias ao usuário de forma compacta e que permita o seu transporte de um lugar para o outro. Sendo assim, ele precisa ser pequeno, leve, durável, operacional em diferentes condições e consumir o mínimo possível de energia de forma a assegurar longa autonomia de uso à bateria. Computadores comuns não foram projetados para serem transportados, por isso ao ser adaptado sofreu alterações quanto a capacidade de armazenamento, restrições de memória, consumo de energia e interface do usuário. Dispositivos portáteis podem ainda serem considerados menos

confiáveis e seguros, já que a portabilidade aumenta os riscos de danos físicos e acessos não autorizados, bem como o risco de roubo ou perda.

3.1.1.2 Comunicação

A comunicação por redes sem fio é dada através de sinas de ondas de rádio por intermédio de satélites, micro-ondas ou pulsos de luz infravermelha, variando sua área de alcance. Por isso, acaba por enfrentar mais obstáculos que a comunicação por fios, já que interage com os sinais, podendo potencialmente bloquear a transmissão ou inserir ruídos e ecos.

3.1.1.3 Mobilidade

A mobilidade é entendida na computação como o acesso à informação em qualquer localização. Leva em consideração as necessidades dos usuários conectados à rede de localizações arbitrárias e que não ficam permanentemente conectados.
A mobilidade exige adaptação, que pode ser entendida como a condição de produzir resultados em condições adversas e, por isso, deve existir um equilíbrio entre a necessidade e a disponibilidade de recursos para a execução de aplicações.
Também aumenta a volatilidade da informação. Dados antes estáticos em computadores fixos agora podem se transformar em dados dinâmicos em dispositivos móveis. Sendo assim, cria a necessidade de gerência de localização e seu custo proveniente passa a ser considerado.

3.2 Arquitetura Para Aplicações Com Banco de Dados Móveis

Foi necessário encontrar uma alternativa para permitir que as aplicações móveis conseguissem executar suas funções remotamente. Assim foram desenvolvidos os Agentes Móveis, cuja função é levar as instruções dados e um estado de execução.
Agentes Móveis são descritos como programas enviados de uma máquina para executar remotamente uma tarefa. Dois fatores principais motivam sua utilização: agentes móveis dispõem de um método ágil e eficiente para obter informações ou serviços desejados; agentes móveis suportam as interrupções das conexões, as redes lentas e os equipamentos com poucos recursos, tornando sua utilização imprescindível para a computação móvel.
Os BD móveis utilizam parte da arquitetura para trafegar a informação, a fim de não permitir que seus dados sejam perdidos por intervenção do espaço geográfico. Um Host Móvel acessa um banco de dados em um Host fixo (caso o banco seja embutido na aplicação) ou em um Host Móvel. O Host Móvel pode ser o Cliente ou o Servidor da requisição com base na aplicação cujas informações se deseja persistir.

3.3 Banco de Dados Móveis

A computação móvel permitiu o desenvolvimento de novas aplicações em banco de dados, ocasionando mudanças no gerenciamento e nos mecanismos de garantia de consistências dos dados. Essas mudanças são originárias das restrições impostas pelos ambientes de comunicação sem fio: a limitação da largura de banda dos canais de comunicação sem fio, a mobilidade, as frequentes desconexões dos dispositivos móveis e a mobilidade dos dados.
Com o constante uso dos dispositivos móveis, se passou a ter também a necessidade de armazenamento de informações para posterior acesso. Sendo assim são desenvolvidos os SGBD Móveis, que são adaptações dos SGBD tradicionais, porém permitindo que eles sejam executados em um dispositivo móvel agindo com um papel de servidor de dados local e global. Banco de Dados móveis podem ser entendidos como uma adaptação dos SBD-D (Sistemas de Banco de Dados Distribuídos), adicionando o conceito da mobilidade com as restrições impostas pelo ambiente.
Bancos de dados móveis geralmente funcionam em redes wireless (sem fio), podendo ser distribuídos em dois possíveis cenários:

1. Toda base de dados está distribuída principalmente entre os componentes ligados por fios, possivelmente com replicação total ou parcial dos dados. Uma estação base

gerencia sua própria base de dados com um SGBD com funcionalidades para localizar unidades móveis e para gerenciar consultas e transações do ambiente móvel
2. A base de dados é distribuída pelos componentes com e sem fio, sendo compartilhada entre as unidades móveis e as estações base a responsabilidade do gerenciamento desses dados.

Com isso, muitas funções de gerenciamento de dados distribuídos também podem ser aplicadas aos bancos de dados móveis levando em consideração a distribuição de dados e replicação (os dados têm a opção de não estar totalmente distribuídos entre as estações base e as unidades móveis, sendo assim as restrições de consistência precisam adequar a unidade móvel aqueles dados mais repetidamente atualizados e acessados) e os modelos de transação (a tolerâncias a falhas e a ocorrência das transações são mais difíceis de serem manipuladas no ambiente móvel, por conta disso as propriedades ACID das transações podem precisar de ajustes e novos modelos de transação devem ser redefinidos para atender as exigências da mobilidade).
Para garantir a disponibilidade da informação e o desempenho, os dispositivos levam seus dados replicados. Após os dados serem distribuídos são mantidos atualizados de forma a garantir ao usuário a certeza de que está disponível a ele à versão mais recente da aplicação.
Em sistemas distribuídos, a recuperação de falhas é baseada em checkpoints.

3.3.1 Processo de Transações com BD Móveis

Em transação móvel, a computação é dividida e executada um período no host móvel e outro no host fixo, não podendo ser considerada estruturada como uma operação atômica.

3.3.1.1 Estados de Operação do Host Móvel

Em um sistema distribuído tradicional, as operações são totalmente conexas ou sem conexão. No caso do meio móvel o momento desconexo pode ser dividido em várias etapas que vão desde falta de conexão por completo até conexão bem fraca, o que ocasiona a possibilidade de uma unidade móvel operar em estados diferentes.
Outro estado é o cochilo, ou adormecido, que é utilizado para preservar a energia. O estado de cochilo diminui a velocidade do clock do processador e não executa qualquer tipo de operação do cliente, aguardando de modo passivo o recebimento de mensagens por parte dos integrantes da rede para voltar ao estado normal de execução. Antes de desconectar da rede o host móvel pode realizar o protocolo de execução, que tem como objetivo dar garantia da realização da transferência de dados para que ele continue a operar o mínimo necessário.
Uma unidade móvel pode trocar do estado desconexo para o estado parcialmente conexo a fim de permitir a sua execução. Enquanto estiver nesse estado o contato com a rede será reduzido em apenas uma célula, transferindo todos os dados pertencentes ao seu estado para à estação base da célula por meio do protocolo handoff.
A desconexão pode ocorrer em três estados distintos: hoarding, operações desconectadas e reintegração.
O Hoarding tem como saída as informações transferidas para a unidade móvel e ocorre quando existiam dados sendo carregados no host antes da perda de conexão. Nesse estado os dados mínimos necessários para realizar a operação são carregados no host móvel e podem ser simplesmente locados novamente ou levados para a unidade móvel pelo host fixo.
Operações desconectadas acontecem quando há quebra na conexão com a rede convencial, permitindo ao host móvel utilizar apenas os dados locais, se tentar solicitar dados não locais, não será atendido e retornará um erro. Nesse caso estes itens pendentes ficarão organizados em uma fila para serem atendidos quando voltar a estar conectado. As aplicações com solicitações que retornarem erro poderão suspender a execução ou trabalhar em outros processos que não dependam de dados locais.

Quando uma nova conexão entre o host móvel e a rede convencional acontece é chamado de estado de reintegração. Nesse processo atualizações da unidade móvel reintegradas através de sites realizam novamente sua entrada no host fixo.

3.3.2 Caching

Uma maneira de aumentar a disponibilidade de dados ao usuário é através do caching, que consistem em armazenar os dados em disco local (cache) para a melhor eficiência das operações. São usadas duas regiões para coordenar a classificação dos dados quanto ao seu uso na cache. A região quente contém o conjunto de dados que são utilizados com mais frequência e a região fria contém os que são usados menos frequência.

3.3.3 Difusão de Dados ou Disseminação

Quando uma informação não se encontra na cache é preciso fazer uma requisição ao servidor. Na computação wireless os servidores são munidos de canais com largura suficiente para fazer a entrega de informação por difusão aos usuários em suas células, meio que deu origem ao denominado push-based. Push-based é uma forma de entrega de informações que consiste no servidor insistentemente jogar informações aos usuários sem a necessidade de qualquer tipo de solicitação, permitindo adquirir os dados que necessitam.
A forma broadcast pushbased entrega as informações que são extremamente importantes a uma gama enorme de aplicações que precisam difundir essas informações para uma quantidade imensa de usuários, como por exemplo avisos de emergência e informações climáticas.
Na forma pullbased o servidor retorna ao usuário um dado solicitado através de um pedido do usuário, o que limita a entrega a capacidade do servidor.

3.3.4 Recuperação e tolerância a falhas

Assim como BD tradicionais, bancos de dados móveis podem apresentar falhas, sendo as mais comuns classificadas como falhas locais (que são ocasionadas pela queda da unidade móvel, sendo a bateria fraca o motivo mais frequente), falhas de mídia, de transação e de comunicação. Falhas de transação são mais frequentes durante operações de handoff. Falhas na unidade móvel afetam os algoritmos de roteamento.A suspensão da conexão voluntariamente causam particionamento da rede, mas não deve ser encarada como falha se os dados e métodos necessários para que se complete uma tarefa ainda estiverem no dispositivo móvel.

3.3.5 Replicação de dados e sincronização

Já citada anteriormente, a replicação é o processo pelo qual um arquivo ou grupo de arquivos é copiado de seu local original para outras máquinas dentro de um sistema distribuído. Com o intuito de aumentar a disponibilidade de dados e a performance os dispositivos carregam dados replicados, ou seja, cópias redundantes das informações contidas no banco de dados distribuído. Permite a otimização dos recursos de hardware e rede, maximizando a flexibilidade e escalabilidade em ambientes heterogêneos.
Sincronização é o processo que mantém os dados distribuídos atualizados. Quando os dados em uma base ou em um servidor e suas réplicas são sincronizados, é usado o log de transação de cada volume para verificar se foram atualizados, inseridos ou removidos da base central, posteriormente replicando as cópias mais recentes para todos os outros locais que acessam os dados.
A base de dados que contém os dados a serem replicados é chamada de consolidada. As bases remotas são aquelas que contêm um subconjunto das informações da base consolidada e geralmente se localizam fisicamente distantes delas, possuindo réplicas dos dados da base consolidada.
A replicação de dados é dividida em: baseada em sessão, baseada em mensagens ou baseada em conexão.

3.3.5.1 Replicação baseada em sessão

A replicação ocorre através uma linha de comunicação direta entre a base consolidada e a base remotas. Os intervalos entre duas destas sessões são variáveis, podendo ser de minutos, horas, dias ou semanas, dependendo da configuração inicial. A base remota cliente abre a comunicação enviando uma lista completa das modificações feitas naquela base desde a última sincronização ao servidor de sincronização que atualiza a base consolidada e envia de volta as modificações relevantes. O host remoto incorpora o conjunto de mudanças e fecha a sessão logo após enviar uma confirmação de sucesso da operação junto com uma tabela que mapeia o contendo das informações que foram atualizadas.

3.3.5.2 Replicação baseada em mensagens

Um agente de mensagens vinculado a cada base envia mensagens (normalmente arquivos armazenados em algum diretório ou correio eletrônico com formato especial) informando as mudanças recentes em sua base e recebe mensagens de um ou mais outros agentes, modificando seus dados de acordo com o conteúdo delas, permitindo replicação entre bancos de dados que não estão diretamente conectados. Para que nenhuma conexão seja necessária entre as aplicações cada uma das mensagens carrega informações de controle como o seu endereço de destino.

3.3.5.3 Replicação baseada em conexão

Caracterizada por ser rápida, confiável e de cara manutenção por necessitar de muitos recursos, dependem de uma conexão contínua, ou praticamente contínua, entre as bases remotas e a base consolidada. Devido a isso é mais apropriado para replicações entre bases consolidadas.

3.3.6 Consistência de dados em conectividade intermitente

Para reduzir custos, consumo de energia e solucionar problemas de disponibilidade e latência os clientes móveis usam a rede alternando entre os modos de operação conectado e desconectado, sendo também desejável uma operação que faça uso da conectividade fraca. Deste modo os clientes operam desconectados na maioria do tempo, conectando ocasionalmente em linha fixa ou retornando para seus ambientes de trabalho.

No esquema proposto para a conectividade intermitente são formados clusters agrupando os dados localizados em sites fortemente conectados. Para cópias localizadas no mesmo cluster a consistência mútua é requerida, contudo níveis de inconsistência são tolerados em diferentes clusters.

Operações fracas permitem o acesso a dados localmente disponíveis, leituras fracas acessam cópias limitadas inconsistentes e escritas fracas fazem atualizações condicionais. As chamadas operações estritas oferecem acessos a dados consistentes e executam atualizações permanentes.

O uso balanceado de ambas as operações, fraca e estrita, fornece melhor utilização de largura de banda, latência e custo em casos de conectividade fraca, sendo o uso de somente operações estritas recomendado em casos de conectividade forte. Há suporte a operação desconectada desde que os usuários possam operar mesmo quando desconectados através do uso de operações fracas.

3.3.7 Recuperação

Apesar de possuir abordagens diferentes, os esquemas de recuperação em banco de dados móveis possuem a mesma plataforma, que contém um grupo de unidades móveis e estações base. Essas unidades salvam logs e checkpoints, garantindo a disponibilidade da informação necessário de maneira eficiente.

3.3.7.1 Esquema de recuperação híbrida de três fases

Combina esquemas de checkpoint e comunicação coordenada e induzida. As estações base usam checkpoint coordenado e o checkpoint de comunicação baseada é usado entre unidades móveis e estações base. O algoritmo usa as unidades móveis e estações base para descrever o tráfego da mensagem. Inicialmente, um coordenador (estação base) envia a todos um pedido de mensagem

com um índice de checkpoint para outras estações base. Cada estação base atualiza um timer. Ao final cada estação base envia mensagens de recuperação para todas suas unidades móveis. Estas unidades móveis retornam (rollback) para seus últimos estados consistentes.

3.3.7.2 Recuperação de falhas e checkpointing de baixo custo

Assumindo que os nós de processamento sejam dependentes, os nós enviam mensagens entre si requisitando o registro do estado do dado (snapshot), se o nó não tiver processado nenhuma mensagem desde seu último snapshot, então ele registra o evento e propaga o snapshot. Se houver falha, o nó volta para seu último checkpoint e envia pedidos de rollback para um subgrupo de nós, que ao receberem sua primeira mensagem de rollback retornam para seu último checkpoint e enviam o pedido de rollback para um outro grupo de nós. No caso de unidades móveis, todos seus pedidos de rollback são roteados através de suas estações base.

3.3.8 Oracle Database Lite

O Oracle Database Lite é um banco desenvolvido pela Oracle que faz uso da linguagem SQL e é uma das opções de mercado para projeto em dispositivos móveis, alguns o considerando ideal por sincronizar com uma instalação baseada em servidor.
Alguns exemplos de códigos são:

O código acima altera o valo máximo de eseq para 1500

```
ALTER SEQUENCE eseq MAXVALUE 1500
CREATE TABLE t1 (c1 INT AUTO INCREMENT, c2 INT, c3 INT);
ALTER TABLE T1 HIDE C1;
INSERT INTO T1 VALUES (123,456);
```

O código acima cria a tabela t1, omitindo o argumento c1 (valores marcados como autoincremento são gerados pelo SBD, portanto não devem ser inseridos) e adicionando os valores 123 e 456 em c2 e c3 respectivamente.

```
CREATE SNAPSHOT TABLE system.ssl (c1 INTEGER, c2 CHAR(10));
```

O trecho de código acima cria uma tabela snapshot.

```
import java.lang.*;
import java.sql.*;
class TriggerExample {
public void EMP_SAL(Connection conn, int new_sal) {

System.out.println("new salary is :"+new_sal);
}
 }
```

```
ALTER TABLE EMP ATTACH JAVA SOURCE "TriggerExample" in '.';
CREATE TRIGGER SAL_CHECK BEFORE UPDATE OF SAL ON EMP FOR EACH ROW
EMP_SAL(NEW.SAL);
```

O exemplo acima cria o programa em Java TriggerExample, coloca seu conteúdo na tabela emp e cria um trigger Java.

CREATE VIEW EMP_SAL (Name, Job, Salary) AS SELECT ENAME, JOB, SAL FROM EMP;
SELECT * FROM EMP_SAL;

O código acima cria uma view emp_sal com o nome, trabalho e salário de cada linha da tabela emp.

3.3.9 SQLite

O exemplo de código abaixo cria um banco de dados ex1 com uma tabela tbl1. Posteriormente são inseridos no banco os valores (‘hello!’, 10) e (‘googbye’, 20).

$ **sqlite3 ex1**
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> **create table tbl1(one text, two int);**
sqlite> **insert into tbl1 values('hello!',10);**
sqlite> **insertinto tbl1 values('goodbye', 20);**
sqlite> **select * from tbl1;**
hello!|10
goodbye|20
sqlite>

sqlite> **.open ex1.db**
sqlite>

O código acima está abrindo o banco de dados de nome “ex1.db”

sqlite> **select * from tbl1;**

O comando acima está selecionando todos os arquivos da tbl1

4. BANCO DE DADOS MULTIMIDEA

4.1 Conceito

O banco de dados de multimidia é uma coleção de dados multimídia inter-relacionados, incluindo texto, gráficos (miniaturas, desenhos), imagens, animação, vídeo, áudio etc. e tem uma grande quantidade de dados de multimidia de várias fontes. A estrutura gerencia diferentes tipos de dados de mídia que podem ser armazenados, remetido e usado de diferentes maneiras é conhecido como um sistema de gerenciamento de banco de dados multimídia.

Existem três classes de bancos de dados multimídia. que inclui mídia estática, mídia dinâmica e mídia dimensional. A principal característica desse tipo de banco de dados é que os recursos de vídeo e áudio são postas para exibir os dados. Consequentemente, taxas fixas e predeterminadas de resgate de dados são chamadas de dados de mídia contínua. Todas essas informações são armazenadas no BLOB (Binary Large Object). Sobre o qual trataremos mais adiante.

4.2 Arquitetura de um banco de dados multimidia

O Banco de Dados Multimídia pode estar disposto de três maneiras, podendo ser conforme os princípios de autonomia, princípios de uniformidade e princípios de organização híbrida. A seguir serão apresentados cada um desses princípios.

4.2.1 Princípio de autonomia

Neste modelo organizacional, os objetos multimídia são organizados em
Mídias específicas para cada tipo de mídia. isso requer criar controles (algoritmos e estruturas de dados) específicos para cada tipo mídia, além de exigir uma tecnologia que combine diferentes estruturas dados .Embora pareça complicado, pois requer muito trabalho computacional, então organizar objetos multimídia individualmente faz com que o tempo das consultas seja rápido.

A figura demonstra a arquitetura de um banco de dados multimídia, quando utilizado o princípio de autonomia, onde cada objeto multimídia possui seu respectivo índice.

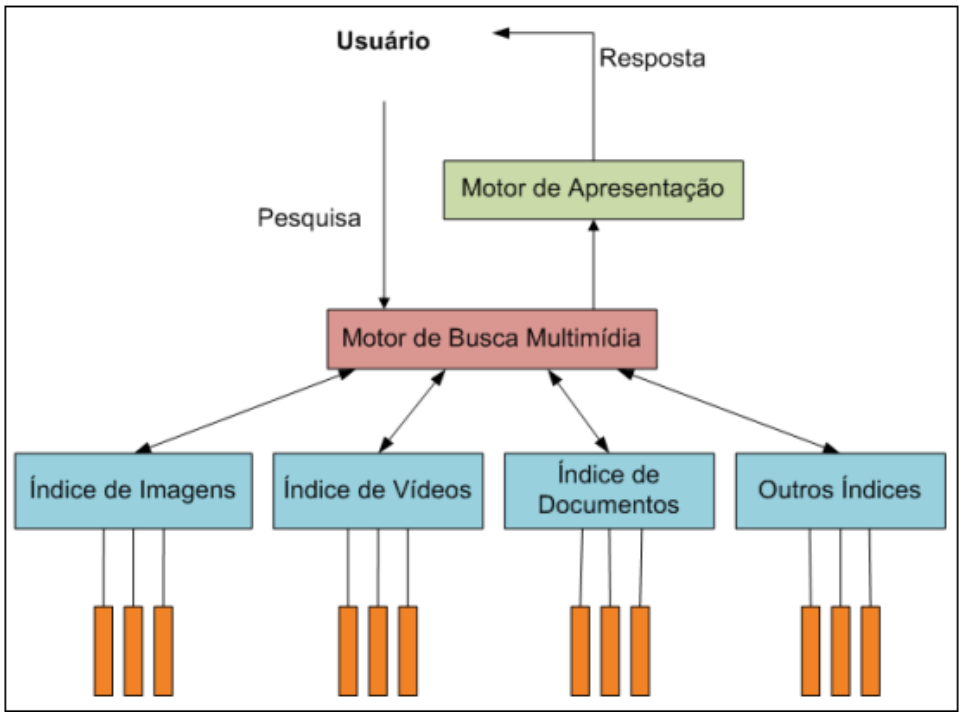
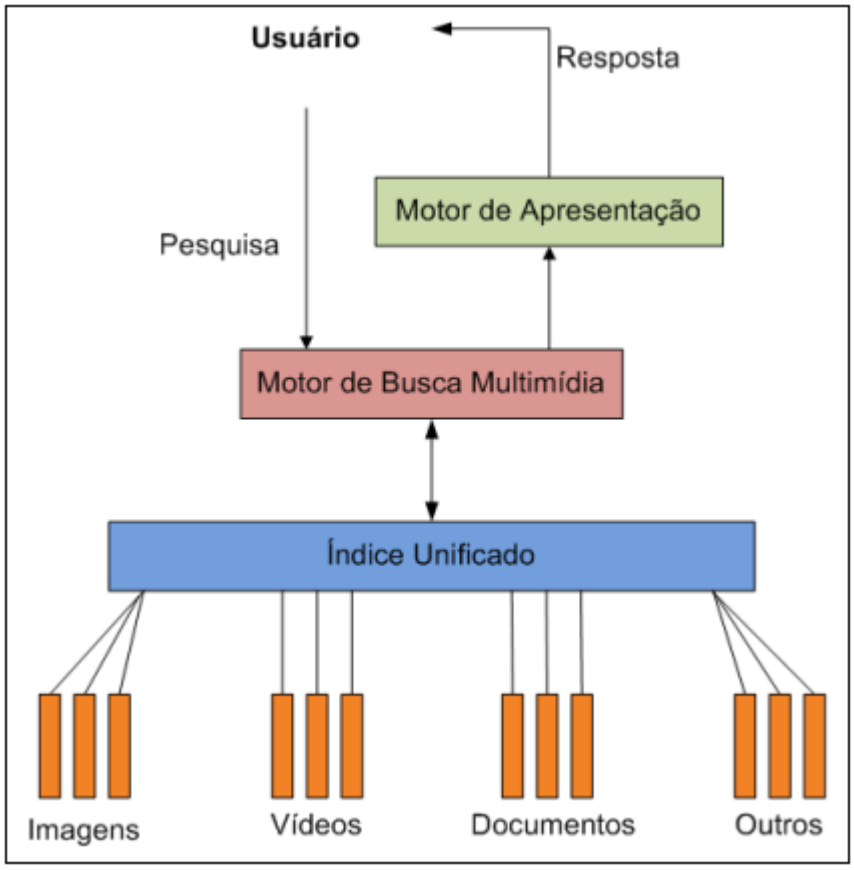


Figura 2-8: Arquitetura de banco de dados multimídia com o princípio de autonomia. Fonte: Adaptado de (SUBRAHMANIAN, apud BIAZI, 2005, p. 7).

4.2.2 Princípio de uniformidade

Neste princípio, uma estrutura é usada onde todos os tipos de mídia são
indexados, capaz de armazenar informações sobre o conteúdo de imagens, vídeos,
documentos, áudio etc. Dessa forma, é preciso analisar
o conteúdo de cada mídia para encontrar características comuns aos objetos,
Então você pode construir índices. Este tipo de arquitetura é normalmente usados para
dispositivos de anotação ou metadados, onde as informações de cada fonte de dados
As mídias são expressas em uma metalinguagem comum, e esses metadados são
índice.
Este tipo de arquitetura é muito simples de implementar, e o algoritmo
O resultado é rápido, mas a anotação tem que ser criada de alguma forma

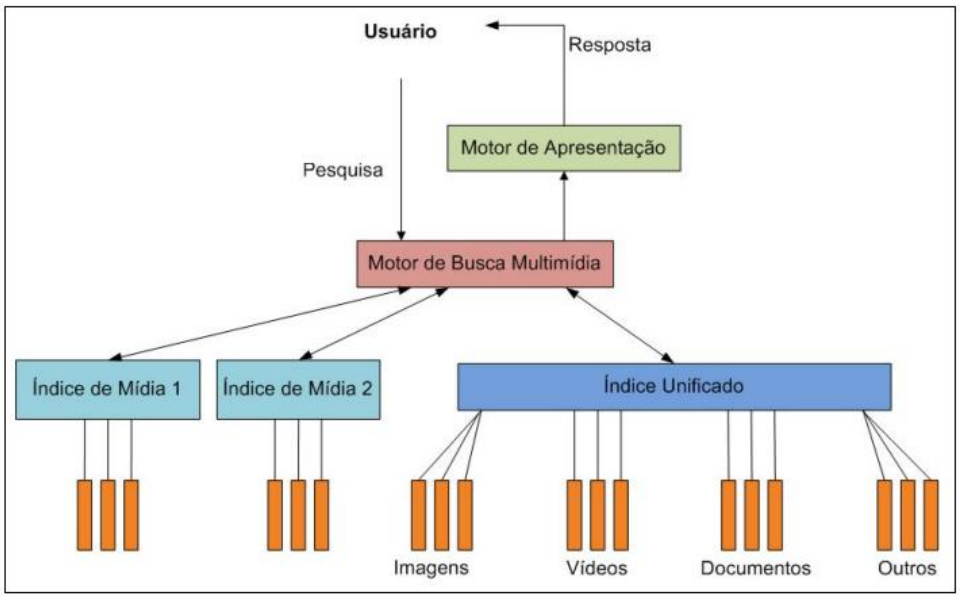
A figura abaixo demonstra arquitetura de um banco de dados multimídia, quando utilizado o princípio de uniformidade, onde existe apenas um índice para todos os objetos multimídia.



Arquitetura de banco de dados multimídia com o princípio de uniformidade. Fonte: Adaptado de (SUBRAHMANIAN, apud BIAZI, 2005, p. 8)

4.2.3 Princípio de Organização Híbrida

O princípio organizador Híbrido, alguns tipos de mídia usam um índice unificado em sua arquitetura, enquanto outros usam seus próprios índices, assumindo as melhores características de ambas as arquiteturas previamente analisada e eliminando deficiências existentes entre elas. A figura abaixo demonstra a arquitetura do banco de dados multimídia, ao usar o princípio da uniformidade, há apenas um índice para todos os objetos multimídia.



Arquitetura de banco de dados com o princípio de organização híbrida. Fonte: Adaptado de (SUBRAHMANIAN, apud BIAZI, 2005, p. 9).

4.3 DADOS MULTIMIDIA NAO INTERPRETADOS

Este tipo de armazenamento utiliza campos do próprio banco de dados para armazenar dados multimídia. No entanto, esses dados são simplesmente processados como uma grande sequência de bytes não interpretados, impossibilitando a realização da estação das características semânticas de objetos diretamente do campo.

Nesse caso, o DBMS fornece apenas acesso persistente e "bufferizado " Para dados, suporte multiusuário, recuperação e controle de autorização, diferente do armazenamento por referência externa, onde tal controle é possível. Sem fornecer um mecanismo de abstração, não será possível desacoplar as aplicações multimídia relacionadas à semântica de dados. Portanto, toda aplicação deve implementar essa semântica.

Para armazenamento e recuperação de dados multimídia, cada SBGD existente no mercado desenvolveu sua própria estrutura interna de armazenamento de dados multimídia. Por exemplo, o Oracle Data base usa LOBs (Large objeto), podendo armazenar de 8 a 128 terabytes dependendo de como o banco de dados foi configurado.

Os LOBs podem ser classificados em quatro tipos de dados LOBs são: BLOB (Binary Large Object), CLOB (Character Large Object), NCLOB (National Character Large Object) e BFILE (Binary File Object). Cada tipo de dados LOB é nomeado de acordo com o tipo de dados que irá armazenar.

Para armazenar arquivos do tipo XML, TXT ou HTML, é recomendável utilizar o tipo CLOB ou NCLBO. Para imagens, vídeo e áudio, recomenda-se o tipo BLOB.

A Figura abaixo apresenta os tipos de dados LOB, com suas respectivas descrições e características.

Tipo de Dado	Descrição	Característica
BLOB	*Binary Large Object*	Armazena qualquer tipo de dado em formato binário. Normalmente usado para dados multimídia tais como imagens, áudio e vídeo.
CLOB	*Character Large Object*	Armazena dados de sequências de caracteres ou documentos que usam uma codificação de caracteres exclusiva (ISO-8859-1, por exemplo).
NCLOB	*National Character Set Large Object*	Armazena dados de sequências de caracteres no banco de dados ou documentos que usam uma codificação de caracteres universal (UNICODE).
BFILE	*External Binary File*	Armazena um arquivo binário armazenado fora do banco de dados no sistema de arquivos do sistema operacional, mas que pode ser acessado a partir de tabelas do banco de dados. Pode ser usado para armazenar dados estáticos, como dados de imagens que não precisam ser manipulados por aplicativos.

Tipos de Dados para Grandes Objetos (LOBs) no SGBD Oracle. Fonte: Adaptado de (ORACLE, 2008, p. 1-5).

4.4 BLOB (BINARY LARGE OBJECT)

4.4.1 O que é o Blob

Um blob (Binary Large Object) é um campo que devido sua característica como um objeto de armazenamento de dados, ele também pode ser considerado como um campo de dados que pode conter qualquer informação binaria, desde fotos, vídeos, áudio assim como qualquer multimidia ou dado no geral.

4.4.2 Origem do nome Blob

O nome Blob se origina de 'lob', que significa Large Object, que traduzindo seria, objeto largo. O B de Blob vem de seu tipo de informação que no caso é binaria.

4.4.3 Vantagens do Blob

Os dados maiores são armazenados fora da página em que se encontra as demais tabelas fazendo com que tenha um aumento da densidade dos registros de dados que estão na tabela elevando a performance e o desempenho de consultas de dados triviais assim como dados blob.

4.4.4 Desvantagens do Blob

Os campos Blob não podem ser chave primária (exceto o TinyBlob que é utilizado no mysql).
O MySQL trabalha com campos Blob, porém, são interpretados como campos texto (TEXT).

OBS.: A única diferença é que esses campos são case sensitive.

4.4.5 Divisão do blob no mysql

No MySQL esse tipo de dado é classificado em:
TinyBlob - campo blob de armazenamento máximo igual a 255 caracteres (8 bits) mais 1 de controle;
Blob - o mesmo que o Tinyblob, porém armazenando até 16535 caracteres (16 bits) mais 2 de controle;
MediumBlob - o mesmo que o Tinyblob, porém armazenando até 16777216 caracteres (24 bits) mais 3 de controle;
LongBlob - o mesmo que o Tinyblob, porém armazenando até 4294967295 caracteres (32 bits) mais 4 de controle.

Para utilização dos tipos objetos no mysql basta criar uma tabela com um campo utilizando um dos tipos descritos.

4.4.6 Divisão do blob no oracle

No Oracle são utilizados os tipos Blob e cBlob que é quem são responsáveis pelo armazenamento destas informações.
Coluna BLOB armazena objetos binários tais como gráficos, vídeos ou arquivos de áudio.
Coluna CLOB armazena objetos caracteres simples, de largura fixa, como documentos de texto.

4.4.7 Divisão do blob no sqlserver

Já no Microsoft SQL Server podemos utilizar o campo Image, Text e NText, aceitando arquivos multimidia de até 2gb de tamanho. Possui outros tipos de dados como binary e o varbinary que armazenam pequenos objetos multimidia como ícones e bitmaps.

4.4.8 Exemplos de SGBD que permitem sua implementação

Os dados multimídia são diferentes por natureza dos tradicionais dados de formato de texto ou numérico. As características dos dados multimídia causam impactos diretos ou indiretos no design do SGBD (Sistema gerenciador de bancos de dados) multimídia
Dentre os SGBD mais utilizados (Microsoft SQL Server, Oracle e MySQL) todos estes oferecem suporte a este tipo de informação.

4.4.9 Exemplos de Utilização:

Aplicações em multimídia podem ser encontradas onde exista a necessidade de gerenciar dados complexos. Educação (treinamento local e a distância, bibliotecas digitais). Saúde (telemedicina, bancos de dados de imagens médicas, exames). Entretenimento (jogos, vídeo sob demanda, TV interativa). Negócios (vídeo conferência, comércio eletrônico)

4.5 CRIANDO BANCO BLOB

Primeiramente, criamos nossa tabela:

----- TABELA DE JORNAL TESTE -----

```
CREATE TABLE 'tbl_jornal' (
  'id ' int(11) NOT NULL auto_increment,
  'titulo' varchar(150) NOT NULL default ' ',
  ' descricao ' varchar(150) NOT NULL default ' ',
  ' texto ' text default NULL,
  ' imagem ' MEDIUMBLOB,
  ' Status ' varchar(10) NOT NULL default ' ',
  PRIMARY KEY  (' id ')
);
```

E então inserimos o conteúdo nesta tabela:

```
INSERT INTO tbl_noticias (titulo, descricao, texto, imagem, status) values("Lorem Ipsum Dolor", "Lorem Ipsum Dolor Lorem Ipsum Dolor", "Lorem Ipsum Dolor Lorem Ipsum Dolor", "teste.gif", "1");
```

5 Conclusão.

Apresentamos os principais conceitos para a modelagem de um banco de dados orientado a objeto. E vimos à importância que este novo conceito trouxe para modelagem de banco de dados, tanto que as grandes empresas que administram os bancos de dados do modelo relacional incorporaram diversos tópicos, criando o modelo Objeto-Relacional conforme demonstrado na subsecção 2.3.2. Apresemos os principais SGOO e suas características e podemos concluir que ainda são poucos os SGOO open source disponível no mercado.

Também foram apresentados alguns exemplos de scripts de código para compreendermos como os conceitos da programação orientada a objeto é incluída no SQL. Conhecemos as linguagens ODL e OQL e apresentamos o modelo de caso da Universidade usando-a.
A partir do que foi interpretado por meio deste conteúdo, vemos que o Blob é um tipo de campo de dados que pode ser atribuído diversos tipos de informações desde que estejam em binário, aceitando desde áudio, vídeo, fotos, e que em certas situações, se torna muito eficiente, porém, tem algumas desvantagens, como o fato de não poder ser chave primaria dentre outros. A forma na qual os bancos de dados no geral se reestruturaram para a adequação do blob a sua própria forma, demonstra o qual necessário e importante ele é. Também foi apresentado seus formatos nos mais populares bancos de dados, assim como um script de uso dele no mysql e sua arquitetura.

Com o presente exposto foi também capaz de ver algumas questões que rodeiam os bancos de dados pra dispositivos móveis, notar as questões pertinentes sobre a manutenção e integridade dos dados e a visualização de alguns Scripts em Oracle Lite e SQLite

6 REFERÊNCIAS BIBLIOGRÁFICAS

1 Navathe, Elmasri, Sistemas de banco de dados – 6º Edição, 2011.

2 Boscarioli, Clodis, Uma reflexão sobre Banco de Dados Orientados a Objetos. Disponível em: https://conged.deinfo.uepg.br/artigo4.pdf, acessado em 26/12/2022.

3 Galante, Alan Carvalho, Banco de Dados Orientado a Objetos: Uma Realidade. Disponível em: http://www.fsma.edu.br/si/edicao3/banco_de_dados_orientado_a_objetos.pdf, acessado em 26/12/2022.

4 Carvalho, Guilherme Cantuária, Sistema de Banco de Dados Orientados a Objetos, 2011. Disponível em: http http://www.fatecsp.br/dti/tcc/tcc0002.pdf, acessado em 26/12/2022.

5 Silva, Priscilla Aparecida Vale. Banco de Dados Orientado a Objetos: Estudo Comparativo, 2004. Disponível em: https://ri.unipac.br/repositorio/wp-content/uploads/2019/08/PRISCILLA-APARECIDA-VALE-SILVA.pdf, acessado em 28/12/2022.

6 Coelho, Taiany, Análise Comparativa Para Avaliações de Tecnologias de Banco de Dados Para Dispositivos Móveis. Disponível em: http://seer.uniacademia.edu.br/index.php/cesi/article/view/128/48, acessado em 30/12/2022.

7 Conceição, Marcos Flávio Rocha. Banco de Dados Móveis. Disponível em: https://ri.unipac.br/repositorio/wp-content/uploads/2019/07/Marcos-Fl%C3%A1vio-Rocha-Concei%C3%A7%C3%A3o.pdf acessado em 30/12/2022.

8 Amado, Paulo Gustavo Fell. Bancos de Dados Móveis: Visão Geral, Desafios e Soluções Atuais. Disponível em: https://www.cin.ufpe.br/~bfl/artigos/mono_BDMoveis_CIn.pdf acessado em 30/12/2022.

9 Ito, Giani Carla. Banco de Dados Móveis: Uma Análise de Soluções Propostas para Gerenciamento de Dados. Disponível em: https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/79554/178707.pdf?sequence=1&isAllowed=y, acessado em 01/01/2023

10 Galliano, Eduardo. Banco de Dados Móveis. Disponível em: https://www.cin.ufpe.br/~rps/transacoes.pdf, acessado em 03/01/2023.

11 Oracle Database Lite. Disponível em: https://docs.oracle.com/cd/E12095_01/doc.10303/e12092.pdf acessado em 03/01/2023

12 https://sqlite.org/cli.html acessado em 04/01/2023

13 Michelazzo, Paulino, Blob fields in **MySQL** Databases, Disponível em: http://www.linhadecodigo.com.br/artigo/100/blob-fields-in-mysql-databases.aspx, acessado em 02/01/2023
14 Lamin, Jonathan, Manipulação de dados BLOB com PHP e MySQL, Disponível em: https://imasters.com.br/back-end/manipulacao-de-dados-blob-com-php-e-mysql, acessado em 02/01/2023
15 Vinkler, Thales, Banco de Dados Multimídia. Disponível em: https://prezi.com/jotzzjhuvj2a/banco-de-dados-multimidia/, acessado em 03/01/2023
16 Kiss Leme, Glauco, MySQL: Banco de Dados Multimidia. Disponível em: http://www.linhadecodigo.com.br/artigo/900/mysql-banco-de-dados-multimidia.aspx, acessado em 03/01/2023
17 MySQL – Trabalhando com Blob Fields. Disponível em: http://www.webmaster.pt/mysql-blob-fields-3293.html, acessado em 04/01/2023.