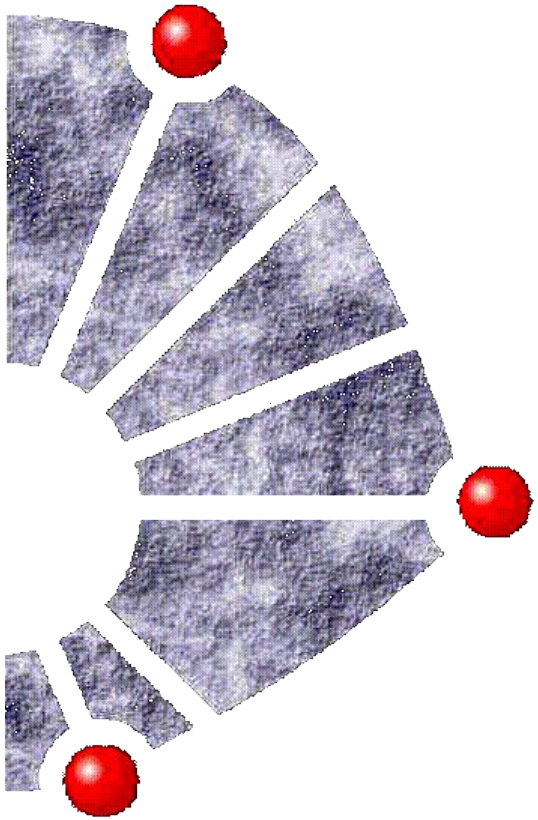
Conectando com Bases de Dados



AULA 4

Bases de Dados

- O que é uma Base de Dados ?
  - É uma coleção de dados organizados de tal forma que possam ser facilmente buscados e atualizados
- Característica Importante
  - organização dos dados
    - facilidade de uso
    - eficiência no processamento
- Servidor de Bases de Dados (ou Banco de Dados)
  - programa que gerencia bases de dados
    - mantém os dados organizados e íntegros
    - provê acesso compartilhado aos dados para múltiplos usuários

Bases de Dados

- Tecnologia de Bases de Dados
  - já sofreram diversas gerações
  - tecnologias vigentes hoje utilizam técnicas da 4a. E 5a. Geração
- Tecnologia de 4a. Geração
  - Base de Dados Relacional
- Tecnologia de 5a. Geração
  - Base de Dados Orientada a Objetos
- Modelo Relacional
  - método predominante nas BD comercialmente disponíveis
  - organiza os dados em tabelas formadas por linhas e colunas
  - **coluna:** identifica o tipo ou domínio da informação
  - **linha:** especifica um conjunto específico de dados sendo armazenados

Bases de Dados Relacionais

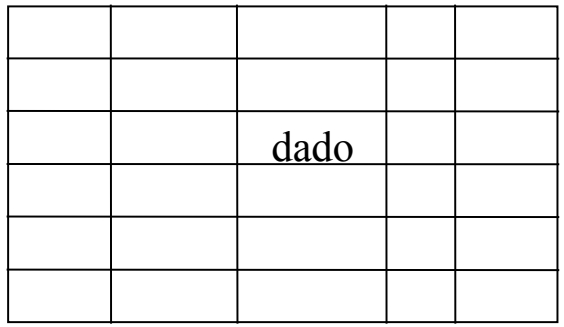
- Estrutura
  - Dado : unidade básica (atômica). Um valor do domínio
  - Relação : estrutura básica (composta)

representada por

Tabela :

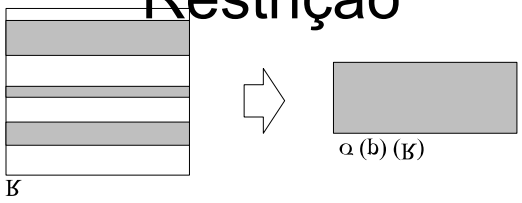
Coluna: representa um domínio

Linha : representa uma tupla da relação

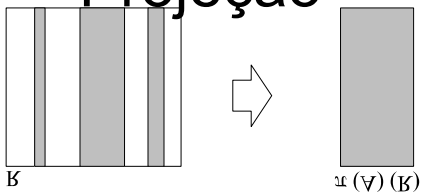


Operações do Modelo Relacional

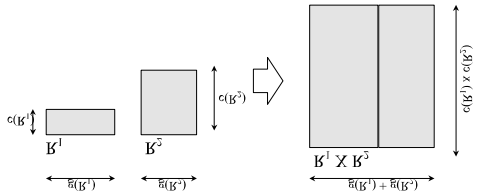
Restrição



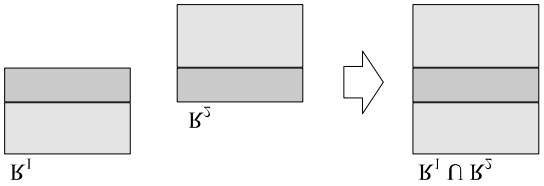
Projeção



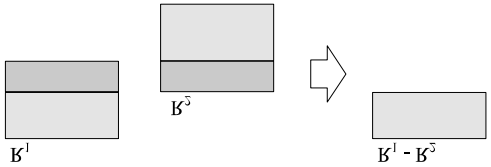
Produto



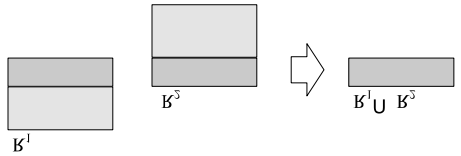
União



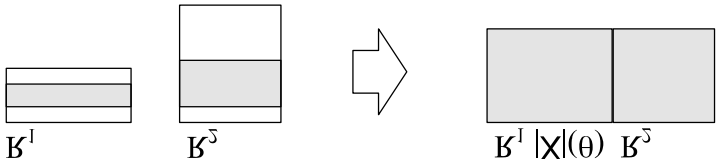
Diferença



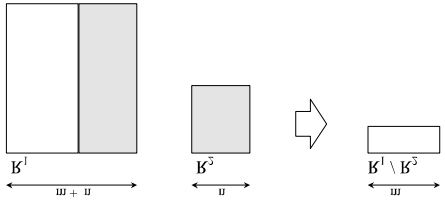
Intersecção



Conjunção (Join)



Divisão



Asserção
Inserção
Eliminação
Atualização

BDs Relacionais e BDs Orientadas a Objeto

- Problemas e Restrições do Modelo Relacional
  - dificuldades na representação e manipulação de dados que não sejam números e textos, tais como dados estruturados de engenharia (CAD, CAE, CAM, CASE), sistemas baseados em conhecimento, sistemas de multimídia e linguagens de programação de sistemas
  - dificuldade de modelar relacionamentos complexos, tais como generalização, agregação e composição
  - diferenças entre as linguagens de programação (que usa os dados) e as linguagens de bancos de dados (que armazenam os dados
- Modelo Orientado a Objetos
  - vem suprir essas deficiências das BD Relacionais

Bases de Dados Orientados a Objetos

- No Modelo Orientado a Objetos
  - dados armazenados nas bases de dados têm a mesma natureza dos dados manipulados pelo programa que os utiliza, ganhando somente o atributo de "persistência"
  - não há a necessidade de se converter os dados para um formato de tabela como no modelo relacional
  - dados são objetos de uma linguagem de programação, que são armazenados em meio persistente, da mesma maneira que apareceriam na memória do computador
  - na verdade, bases de dados orientadas a objetos são somente "espelhos" para um grupo de objetos sendo manipulados pelo programa aplicativo, e que tem a marca de "persistentes"
  - assim, as mesmas técnicas de análise de domínio utilizadas para o desenvolvimento de software podem ser utilizadas para o desenvolvimento de bases de dados orientadas a objeto

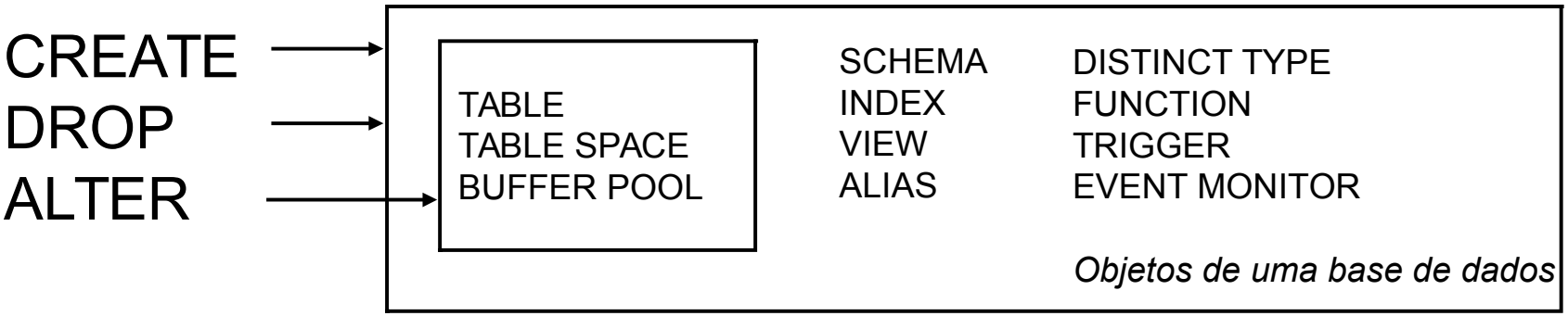
Modelos ER e Modelos Orientados a Objetos

- Diferenças entre os Modelos Entidade-Relacionamentos e Modelo Estrutural Orientado a Objetos
  - representação das atividades dos objetos
    - não é possível no modelo ER
  - referência de objeto x chaves primárias p/ entidades
- Modelos ER
  - somente podem representar dados passivos (que não exercem atividades)
- Modelos OO
  - podem representar entidades (objetos) que exercem algum tipo de atividade no sistema
  - podem portanto, representar o papel ativo que alguns tipos de entidades exercem no mundo real, e que não era representado no modelo ER

SQL: Structured Query Language

- SQL
  - linguagem para interação com bases de dados relacionais, desenvolvida pela IBM na década de 70, tendo se tornado um padrão para BD no final dos anos 80
  - tem diversos usos:
- DDL - Data Definition Language
  - utilizada para criar e projetar bases de dados
- DML - Data Maintenance Language
  - utilizada para atualizar os dados contidos em umba base de dados
- DQL - Data Query Language
  - utilizada para a busca de informações em uma base de dados

SQL para a Definição de Dados (DDL)



Tipos de dados SQL

Numéricos	Cadeias de caracteres de 1 byte	Cadeias de caracteres de 2 bytes	Cadeias binárias (multimídia)	Dados de tempo (data e hora)
SMALLINT INTEGER DECIMAL REAL DOUBLE	CHAR VARCHAR LONG VARCHAR CLOB	GRAPHIC VARGRAPHIC LONG VARGRAPHIC DBCLOB	BLOB	DATE TIME TIMESTAMP

Projeto Físico da Base de Dados

```
CREATE TABLE DEPARTAMENTO (
  IdDep            CHAR(3) NOT NULL,
  NomeDep          VARCHAR(40),
  Responsavel      udt_idper,
  DepSuperior      CHAR(3),
  PRIMARY KEY (IdDep))
```

especificação de chave primária

Sempre deve conter informação

```
CREATE TABLE PESSOA (
       IdPes             udt_idper    NOT NULL,
       IdDep             CHAR(3)      NOT NULL,
       Sobrenome         VARCHAR(20)  NOT NULL,
       Nome              VARCHAR(20)  NOT NULL,
       Foto              BLOB(1000),
       DataIngresso      DATE,
       IdCargo           VARCHAR(20)  NOT NULL,
       Instrucao         SMALLINT,
       Sexo              CHAR(1),
       DataNascimento    DATE,
       Salario           udt_cred,
       Bonificacao       udt_cred,
       Comissao          udt_dolares,
       PRIMARY KEY (IdPer))
```

Tipo de dado definido pelo usuário

```
CREATE DISTINCT TYPE udt_cred
                  AS DECIMAL (9,2)
                  WITH COMPARISONS
```

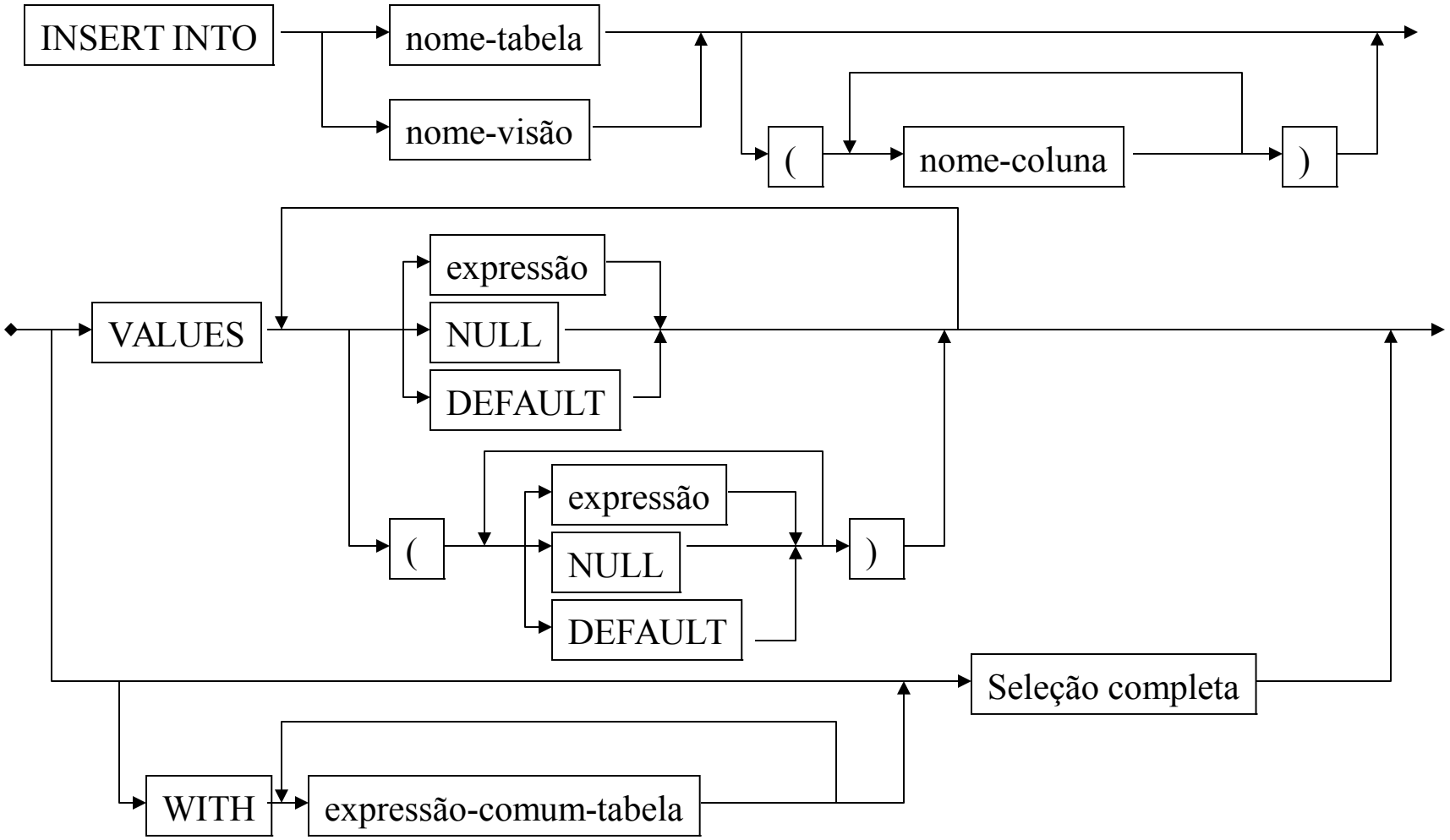
Regra de integridade referencial

```
ALTER TABLE PESSOA
      ADD cfkiddep FOREIGN KEY (iddep)
          REFERENCES departamento
          ON DELETE RESTRICT
```

SQL como DDL e DQL

- SQL para a Manipulação de Dados (DDL)
  - Sentenças INSERT
    - utilizadas para a inserção de linhas em uma tabela
  - Sentenças DELETE
    - utilizadas para a eliminação de linhas em uma tabela
  - Sentenças UPDATE
    - utilizadas para a atualização de linhas em uma tabela
- SQL Para a Consulta de Dados (DQL)
  - Sentenças SELECT
    - utilizada para a busca de informações em uma tabela
  - Cláusula WHERE
    - permite a especificação do espaço de busca nas tabelas consultadas

INSERT : Inserção de Uma ou Mais Colunas



DELETE : Eliminação de Uma ou Mais Colunas

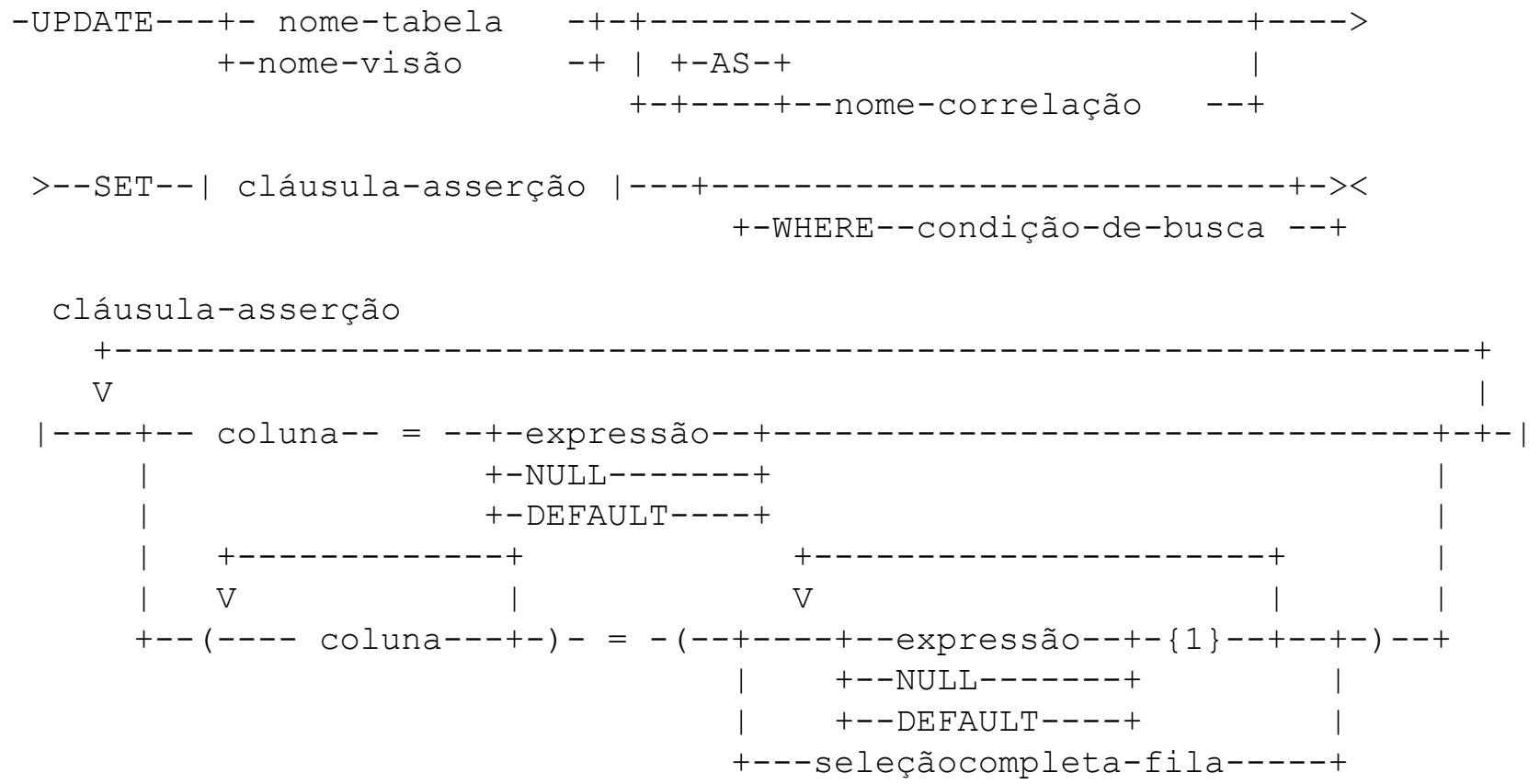
DELETE com busca:

```
 >>-DELETE FROM---+- nome-tabela-+------------------------------>
                  +- nome-visão -+

 >--+-----------------------------+------------------------------>
    | +-AS-+                      |
    +-+----+-- nome-correlação  --+

 >--+-----------------------------+------------------------------><
    +-WHERE--condição-de-busca --+
```

DELETE com posição :

```
>>-DELETE FROM---+- nome-tabela-+------------------>
                 +- nome-visão-+

>-WHERE CURRENT OF-- nome-cursor ----------------><
```

UPDATE : Atualização de Uma ou Mais Colunas

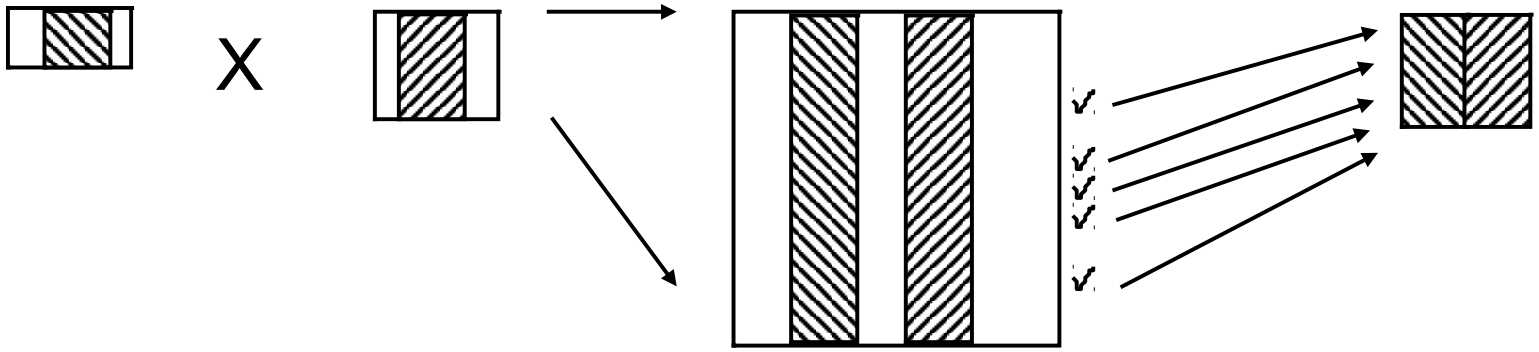


SELECT : Recuperação (Consulta) de Dados

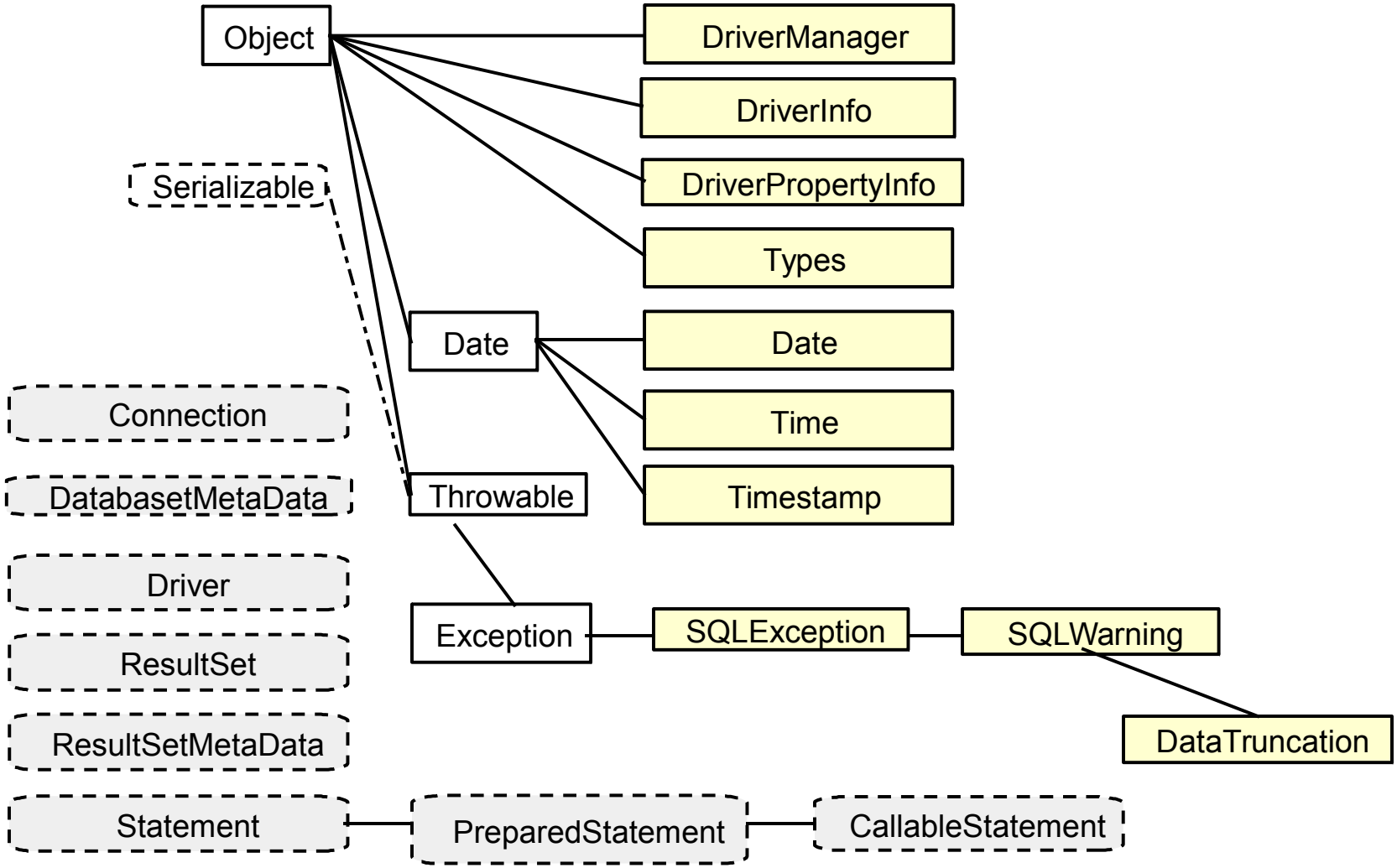
SELECT colunas e/ou expressões (Projeção Relacional)

FROM tabelas e/ou visões (Produto Cartesiano Relacional)

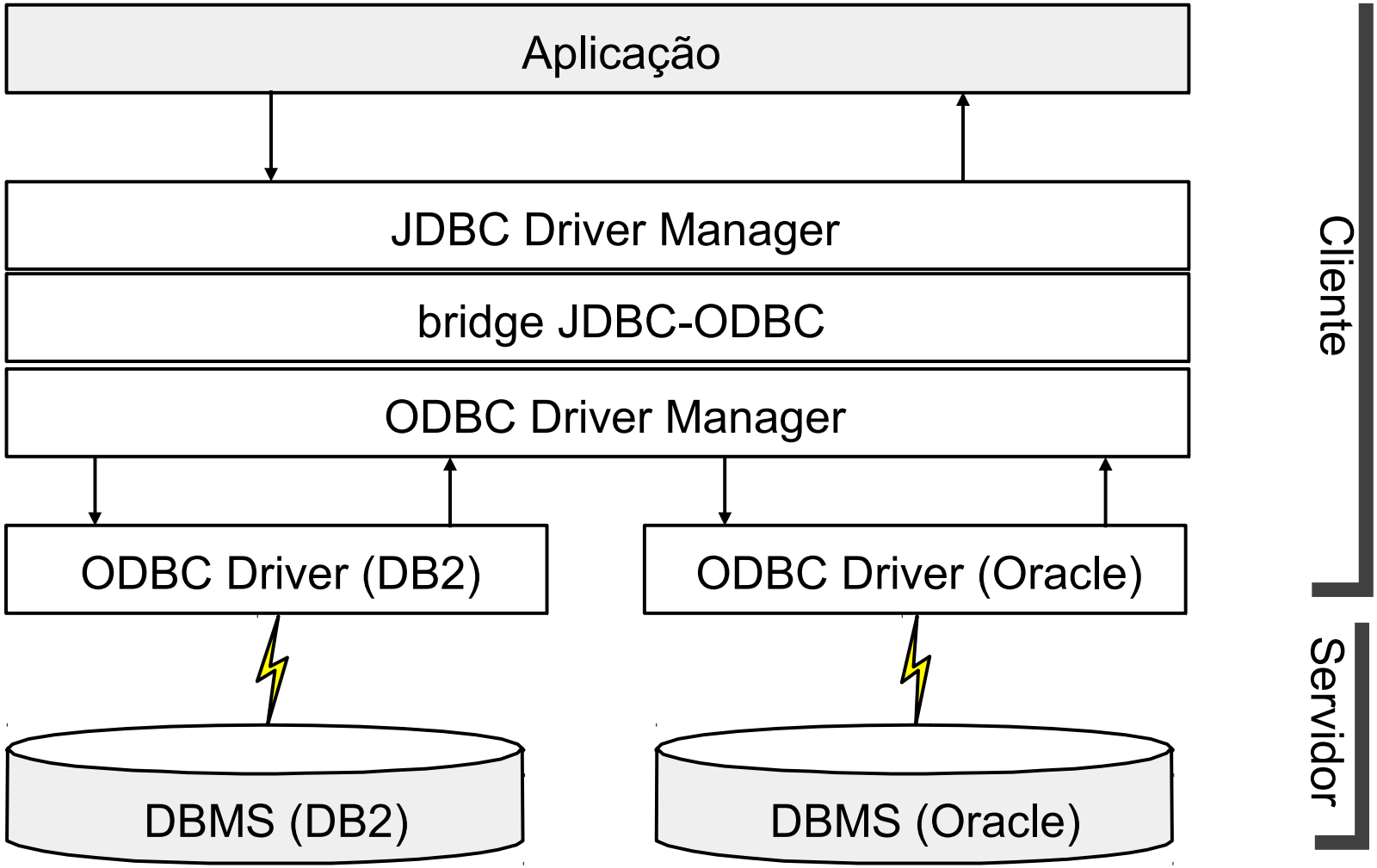
WHERE expressão condicional (Seleção Relacional)



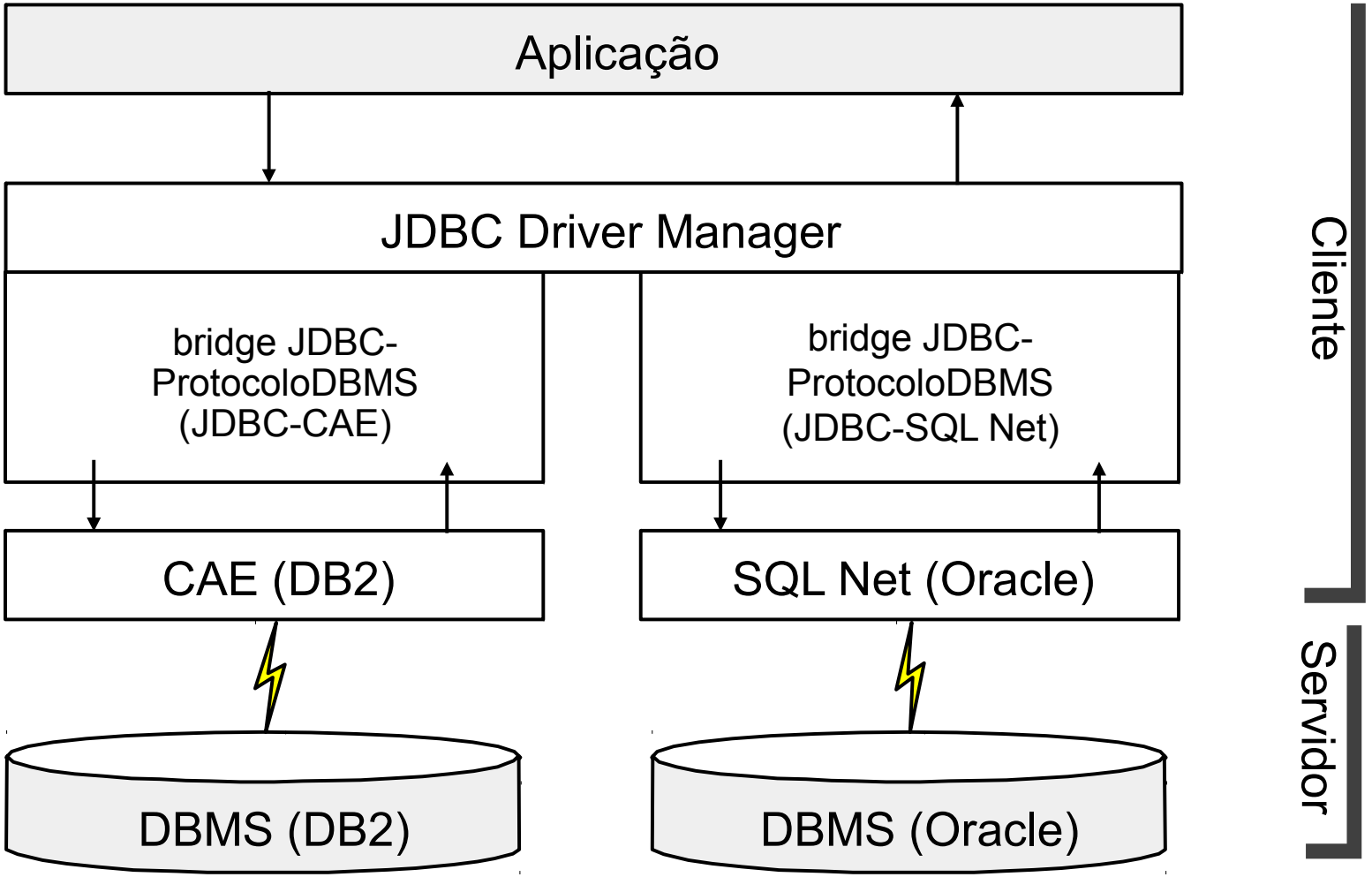
java.sql : o pacote JDBC



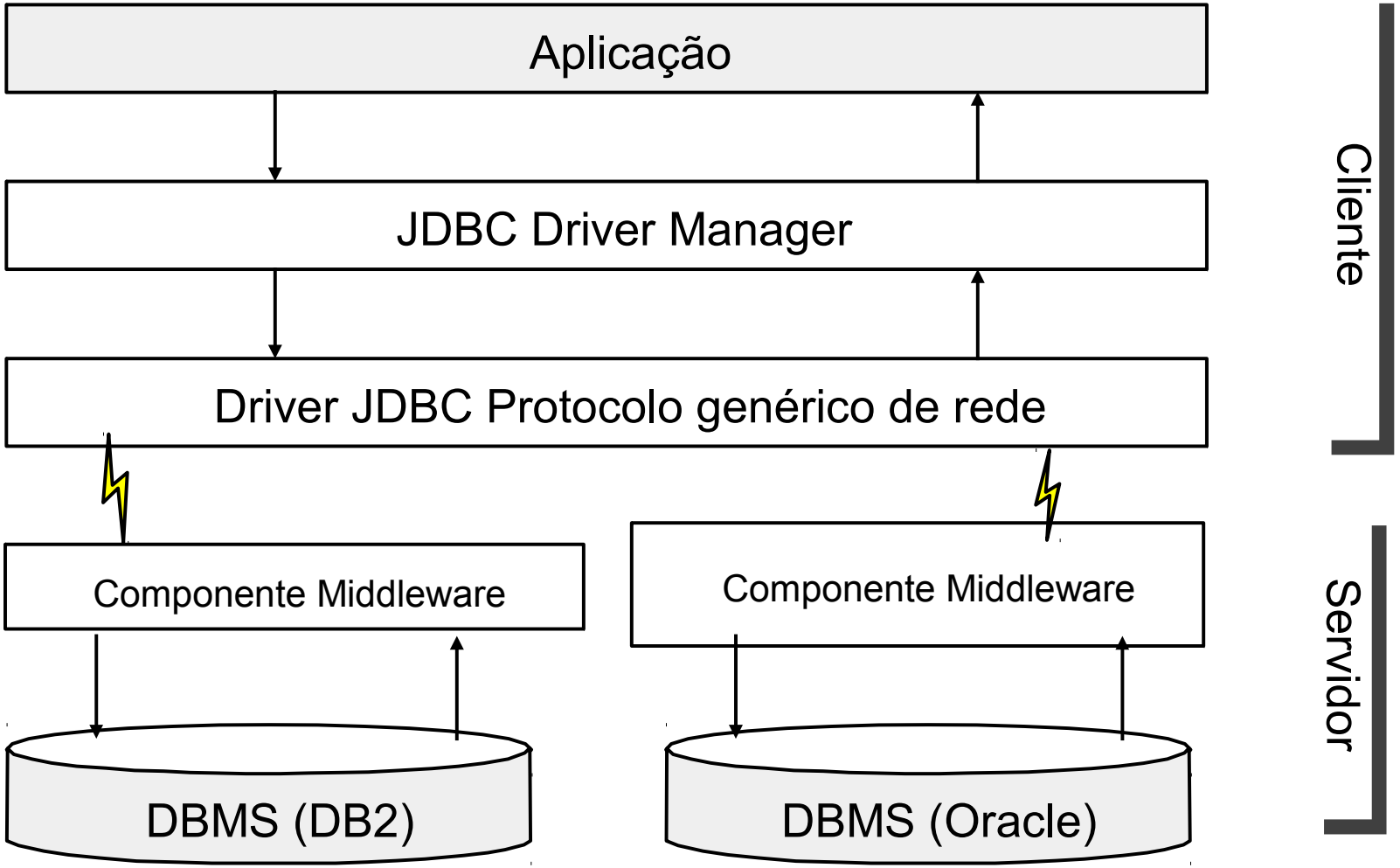
JDBC Categoria 1 : A Ponte JDBC-ODBC



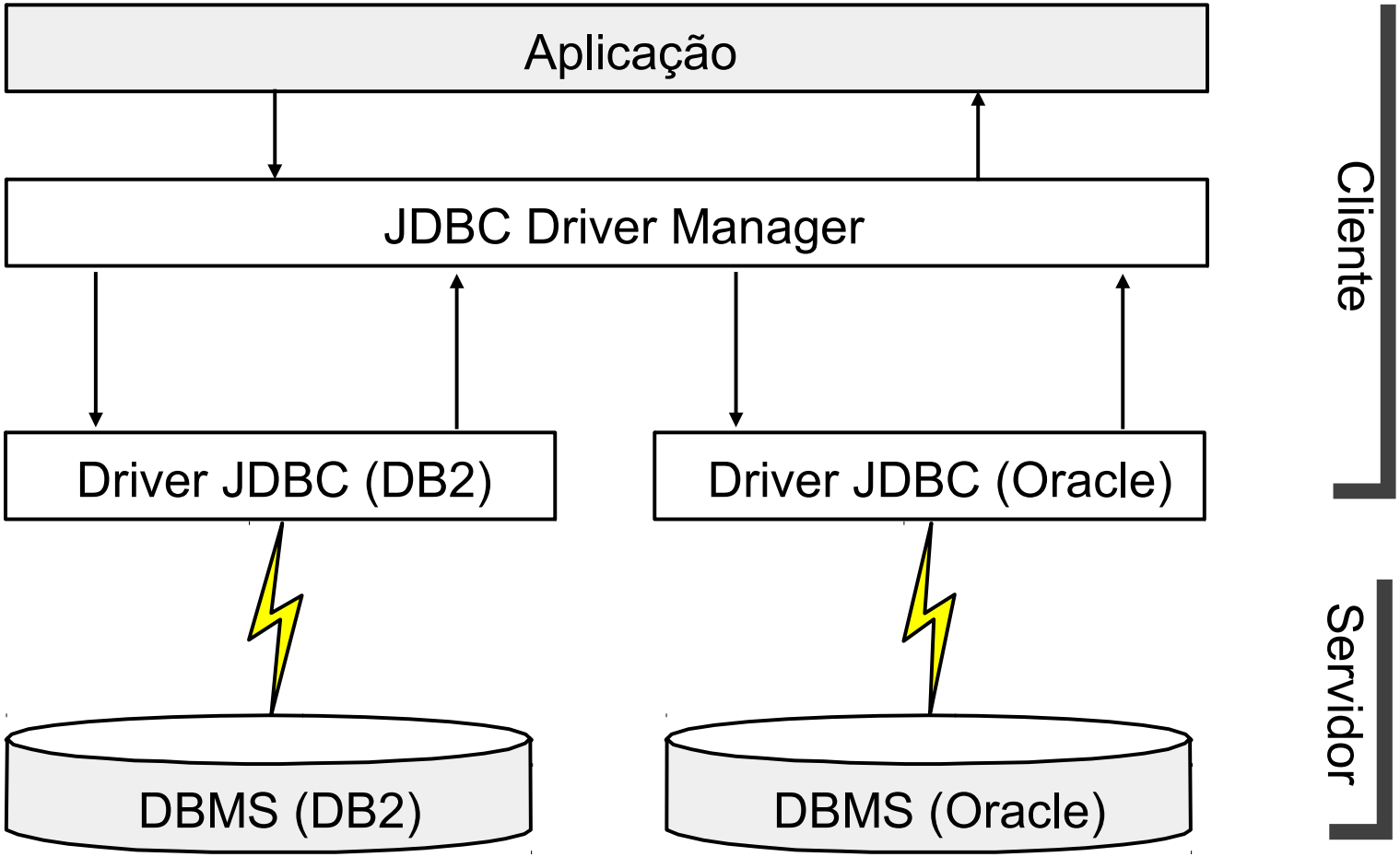
JDBC Categoria 2 : JDBC-DriverDBMS



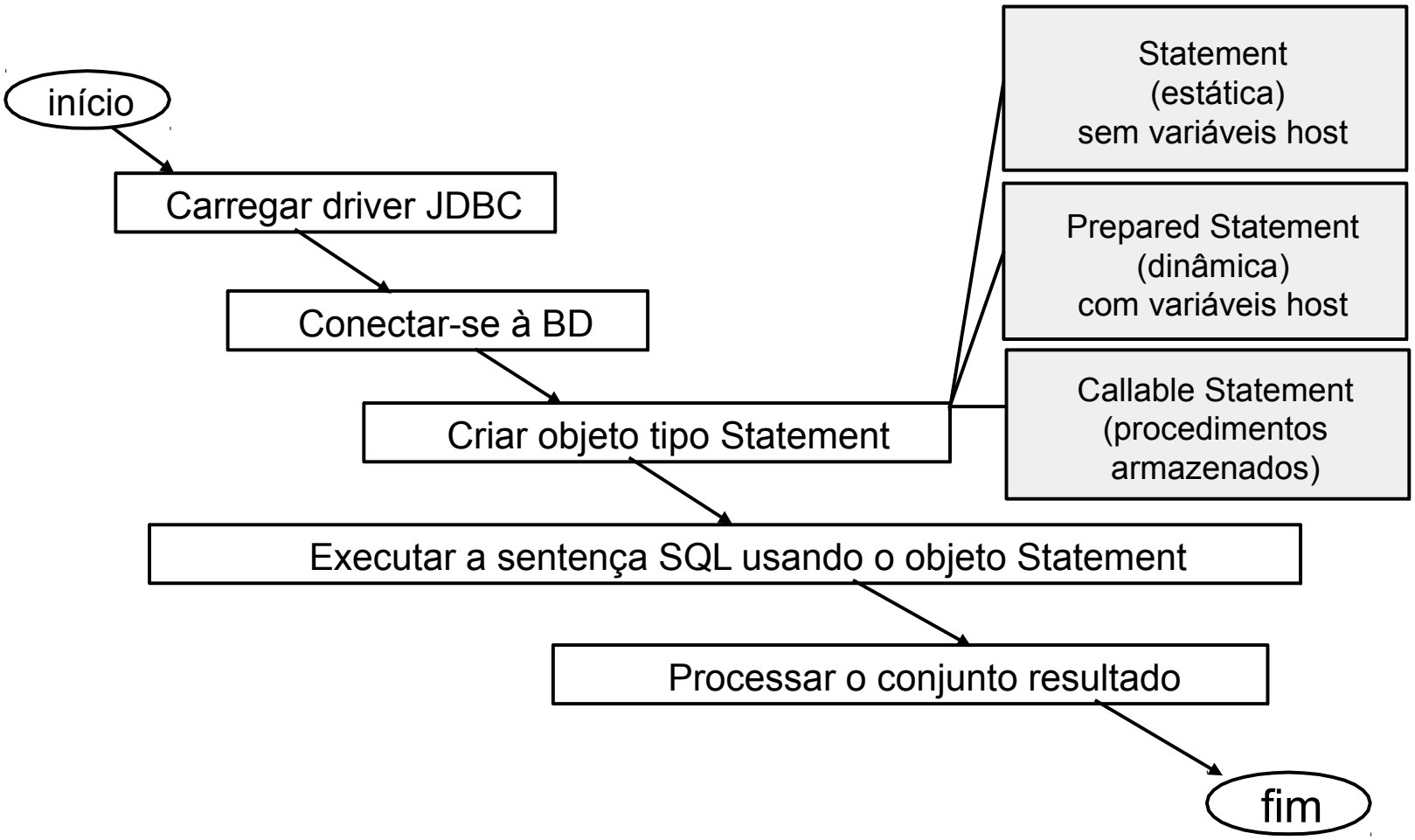
JDBC Categoria 3 : JDBC-Driver de Rede



JDBC Categoria 4 : JDBC Nativo (O Ideal)



Estrutura de Uma Aplicação JDBC



Carregar driver JDBC e Conectar-se à BD

```
import java.sql*;

public class JDBCConnection {

 public static void main (String args []) {
    try {
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();

        String url = "jdbc:db2:sample";

        Connection con = DriverManager.getConnection(url,"usrid","psswd");

        System.out.println("conectado à base de dados");

        con.close();
    }
     catch(Exception e) {
        System.out.println(e);
     }
 }
}
```

Carrega o driver com DriverManager

A fonte dos dados

Cria um objeto conexão

Fecha a conexão liberando recursos

Executando Sentenças SQL e Manipulando o Resultado

```
public static void main(java.lang.String[] args) {
      try {
            int n, i = 0;
            StringBuffer dado;
            Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
            Connection con = DriverManager.getConnection("jdbc:db2:sample","usrid","pwd");
            Statement   st = con.createStatement();
            ResultSet   rs = st.executeQuery("SELECT * FROM ORG");
            ResultSetMetaData md = rs.getMetaData();
            int colunas = md.getColumnCount();
            while (rs.next()) {
                  for (n=1; n<=colunas; n++) {
                        dado = new StringBuffer(rs.getString(n));
                        dado.setLength(md.getColumnDisplaySize(n)+1);
                        System.out.print(dado);
                  }
                  System.out.print("\n");
                  i++;
            }
            System.out.println("\n"+i+" colunas selecionadas");
            rs.close();          st.close();          con.close();
      }
      catch (ClassNotFoundException   e) {System.out.println(e);}
      catch (SQLException          sqle) {System.out.println(sqle);}
      catch (InstantiationException  ie) {System.out.println(ie.toString());}
      catch (IllegalAccessException iae) {System.out.println(iae.toString());}
}
```

Usando um objeto de tipo Connection pode-se criar um objeto do tipo Statement

A execução da sentença retorna como resultado um objeto de tipo ResultSet

O objeto ResultSet mantém um cursor que inicialmente se posiciona na 1a coluna. O método next() avança para a coluna seguinte.

PreparedStatement: Sentença para SQL dinâmico

```
public static void main(java.lang.String[] args) {
     try {
          StringBuffer dado;
          Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
          Connection con =
              DriverManager.getConnection("jdbc:db2:sample","usrid","pwd");
          PreparedStatement st =
              con.prepareStatement("SELECT * FROM ORG WHERE DEPTNUMB > ?");
          st.setInt(1,40);
          ResultSet   rs = st.executeQuery();
          imprime(rs);
          st.setInt(1,60);
          ResultSet   rs = st.executeQuery();
          imprime(rs);
          rs.close();        st.close();        con.close();
     }

     catch (Exception e) {System.out.println(e);}
}
```

Prepara-se a sentença com uma variável host

Atribui-se um valor adequado à variável host a cada execução

Novos Recursos do JDBC 2.0

- Novo ResultSet com Capacidade de Scroll
  - O ResultSet convencional não pode atualizar a base de dados e tem um cursor (apontador) que se move somente para frente.
  - Utilizando novos métodos da interface ResultSet é possível consultar os elementos da tabela de forma assíncrona. Assim o apontador da tupla corrente pode ser posicionada de forma relativa à posição corrente.

```
Statement stmt = con.createStatement(
ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
```

- Atualizações em ResultSet
  - Da mesma forma que os métodos ResultSet.getXXX recuperam informações do resultado da *query,* os métodos ResultSet.updateXXX atualizam a tupla corrente.
- Atualizações em Lotes
  - É possível agregar comandos SQL para serem executados de uma só vez com o método Statement.addBatch() e Statement.executeBatch()

Novos Recursos do JDBC 2.0

- Novos Tipos de Dados Seguindo a Norma SQL3
  - BLOB
  - CLOB
  - Array
  - etc...
- Mapeamento Customizado de Tipos Definidos pelo Usuário (UDTs)
  - É possível fazer-se um mapeamento entre um UDT (*User Defined Type*) definido no SQL e uma classe no Java.
  - Para tanto esta classe deve implementar a *interface* SQLData, implementado os métodos readSQL() e writeSQL()

```
java.util.Map map = con.getTypeMap();
map.put("mySchemaName.ATHLETES", Class.forName("Athletes"));
con.setTypeMap(map);
```

JDBC de Alto Nível

- Ferramentas de Desenvolvimento
  - Implementam Java Beans especializados para o acesso a base de dados via JDBC
  - Estes Java Beans fazem grande parte da programação
- Por Exemplo, no JBuilder:
  - JDataStore para cache de dados e persistência compacta
    - Suporte a transações e recuperação de *crash*
    - Controle de concorrência avançado para melhoria de desempenho
    - Drivers de JDBC 2.0 tipo 4 (local e remoto)
    - JDataStore Explorer - gerenciamento visual de DataStores
  - JDBC database tools
    - SQL Builder – criação visual de queries SQL
    - JDBC Explorer – visualização de base de dados, schemas e criação de conexões para URLs
    - JDBC Monitor – monitoração de aplicações SQL
    - Data Modules
    - Data Module designer
    - Data Modeler
    - Connection URL Builder