Unidade III

5 PROJETO DE BANCO DE DADOS

Antes de iniciar um projeto de banco de dados, você precisa conhecer como ele funciona e qual a linguagem de comunicação. De modo geral, os Sistemas Gerenciadores de Banco de Dados (SGBD) são compostos de diversos módulos e cada um deles é responsável por um conjunto de funcionalidades. Como existem muitos sistemas, cada um com suas particularidades, é muito raro um profissional dominar o funcionamento de todos. Normalmente, encontramos nas empresas um especialista que conhece profundamente o SGBD que a organização comprou. Por ser um profissional altamente qualificado, somente as grandes organizações costumam mantê-lo como um funcionário de carreira. As pequenas empresas têm o hábito de contratar esses especialistas somente na etapa de concepção e implementação do projeto e manter um contrato de consultoria para os momentos críticos, ou quando necessitam de atualização e otimização do ambiente.

5.1 Linguagens de banco de dados

A linguagem para acessar um banco de dados depende do tipo de banco de dados. Aqueles do tipo relacional usam a linguagem SQL (*Structured Query Language*).

Essa linguagem originalmente foi criada pela IBM, e de forma muito rápida começaram a surgir diversas variações criadas por outros fabricantes. Para solucionar esse problema, houve a necessidade de se criar uma linguagem-padrão, mantida pela *American National Standards Institute* (ANSI) desde 1986; aqui no Brasil, pela ISO desde 1987, sendo ambos os órgãos governamentais que cuidam das normas e padrões.

Em 1992, o SQL foi revisto e surgiu o SQL-92. Com a revisão em 1999, foi criado o SQL-99 (SQL3) e depois, em 2003, com nova revisão, surgiu o SQL:2003. No entanto, mesmo com a padronização da ANSI e do ISO, o SQL possui muitas variações e extensões produzidas pelos fabricantes de SGBD.

A maioria das instruções no SQL funciona da mesma forma nos diversos bancos de dados, mas nem todas as funções têm a mesma característica e forma de escrita. Por exemplo, o comando **SELECT** funciona da mesma forma para todos. Se você executar: **SELECT * FROM PRODUTO**, esse comando retornará todos os dados de todos os atributos da tabela produto, independentemente se o seu banco de dados é SQL Server, Sybase, MySQL ou Oracle. Mas, se você executar o comando **SELECT GETDATE()** no banco de dados Oracle, ele não vai funcionar, por se tratar de um comando do SQL Server, que também funciona no Sybase. Para obter a data corrente no Oracle, você deverá executar o comando **SELECT SYSDATE FROM DUAL.**

No caso de banco de dados multidimensionais, a linguagem mais apropriada é a MDX (*Multidimensional Expressions*), que é semelhante à sintaxe do SQL. O MDX não é uma extensão do SQL. Para criar expressões

MDX usadas para desenhar ou projetar cubos, ou para criar consultas para retornar e formatar dados multidimensionais, você precisa entender os conceitos básicos de modelagem dimensional, pois essa linguagem tem elementos de sintaxe, operadores, instruções e funções próprias diferentes do SQL.

5.2 Tipos de linguagem SQL

A linguagem SQL pode ser dividida em tipos de acordo com a sua funcionalidade. Os tipos usados nos sistemas gerenciadores de banco de dados são definidos a seguir.

5.2.1 *Data Definition Language* (DDL)

É o tipo de linguagem que interage com os objetos do banco de dados: tabelas, *procedures*, *functions*, *views*, *triggers*, entre outros. Cada SGBD tem seus manuais técnicos com a lista completa de comandos DDL e suas respectivas sintaxes completas.

São exemplos de comandos DDL:

- **CREATE**
- **ALTER**
- **DROP**

Para criar uma tabela para armazenar o registro do aluno (RA) e o nome do aluno, a sintaxe do comando é:

```
create table ALUNO ( RA char(7), NOME varchar(100) )
```

Para incluir uma nova coluna na tabela aluno, que será usada para armazenar a data de nascimento, precisamos alterar a tabela criada anteriormente por meio do comando **ALTER TABLE**:

```
alter table ALUNO add DATA_NASCIMENTO date
```

Caso você precise alterar a propriedade do campo RA para **NULL**, o comando a ser executado é:

```
alter table ALUNO alter column RA char(7) NULL
```

Observação

Em alguns SGBDs, quando não informamos a nulidade do atributo, o padrão do gerenciador é considerar como nulo. Recomendamos ler os manuais de referência do SGBD com que você for trabalhar.

Importante: é altamente recomendado que você conheça o SGBD de que você está cuidando, pois, no caso de criar um atributo nulo, mas depois precisar alterá-lo para não nulo, nem todos os

gerenciadores permitem esse tipo de alteração sem a definição de um valor *default* de preenchimento do atributo na execução do comando **ALTER TABLE**. Caso não exista um valor-padrão para todas as linhas, você não conseguirá executar o comando e terá que recriar a tabela novamente.

Para definir, isto é, criar a chave primária na tabela aluno, caso você não tenha usado o comando completo do **CREATE TABLE**, você pode fazer uma nova alteração na tabela, por meio do comando:

```
alter table ALUNO add constraint PK_ALUNO primary key(RA)
```

Se você precisar excluir a tabela aluno do seu banco de dados, basta executar o comando:

```
drop table ALUNO
```

Para excluir apenas uma coluna da tabela, o comando é:

```
alter table ALUNO drop column DT_NASCIMENTO
```

5.2.2 *Data Manipulation Language* (DML)

É o tipo de linguagem que interage com os dados armazenados dentro das tabelas do banco de dados. São comandos DML:

- **INSERT**
- **UPDATE**
- **DELETE**
- **TRUNCATE**

Depois de criar a tabela aluno, precisamos cadastrar os dados dos alunos, ou seja, inserir dados na tabela por meio do comando **INSERT**:

```
insert into ALUNO (RA,
                   NOME,
                   DT_NASCIMENTO)
           values ('1234567',
                  'LUIZ FERNANDO',
                  '1978-08-28')
```

Caso você precise alterar algum dado na tabela, o comando a ser executado é o **UPDATE**:

```
update ALUNO set NOME = 'LUIS FERNANDO' where RA = '1234567'
```

O comando **update ALUNO set NOME = 'LUIS FERNANDO'** sem a informação da cláusula **where RA = '12345671'** fará a alteração do nome de todos os alunos cadastrados na tabela para "LUIS FERNANDO".

Lembrete

Para realizar alterações nos dados da tabela, você precisa realizar a operação de **UPDATE** com muita atenção e cautela. Se você não informar a chave primária na cláusula *where* do comando, a alteração será realizada para todos os registros da tabela.

Para excluir dados de uma tabela, o comando utilizado é:

```
delete from ALUNO where RA = '1234567'
```

Se você quiser excluir todos os dados de uma tabela, é só executar o comando **delete from ALUNO.**

O comando **DELETE** deve ser usado com atenção, pois, se você não informar o valor da chave primária na cláusula *where*, todos os dados da tabela serão excluídos.

Para efetuar alterações e exclusões em dados, no caso de as tabelas se relacionarem com outras, você só poderá efetuar as alterações respeitando a hierarquia desses relacionamentos. Portanto, antes de executar os comandos de DML, você deve conhecer o modelo de dados que está implementado.

Para excluir todos os dados, você também pode usar o comando **TRUNCATE**. A diferença entre ele e o comando **DELETE** é que a ação de *truncate* não é gravada no *log* de transações.

```
truncate table ALUNO
```

Observação

Como a ação de *truncate* não é uma transação gravada no *log* do banco de dados, ela não é recomendada para ser usada nas aplicações. Por questões de segurança, as transações que modificam os dados (*insert*, *delete* e *update*) são auditadas, ou seja, o administrador de banco de dados consegue identificar e rastrear essas ações. Mas, se a transação de exclusão de dados for executada por meio do comando **TRUNCATE**, não deixará nenhum rastro.

5.2.3 *Data Control Language* (DCL)

É o tipo de linguagem que interage com a parte de segurança do banco de dados. Baseado na política de segurança definida para o sistema desenvolvido, é com esses comandos que podemos garantir quais dados serão acessados, que tipo de ações poderão ser realizadas por uma pessoa ou um grupo de pessoas. São comandos DCL:

- **GRANT** (*select*, *insert*, *delete*, *update*)
- **GRANT EXECUTE**
- **REVOKE** (*select*, *insert*, *delete*, *update*)
- **REVOKE EXECUTE**

O comando para permissão de execução em todas as *procedures* do banco de dados é:

grant execute to usuario

Lembrete

A sintaxe sempre vai depender do SGBD que você está usando. No caso de um SGBD da Sybase, por exemplo, não é possível executar o **GRANT EXECUTE** para todas as *procedures* num único comando. Você terá que escrever um *script* contendo o **GRANT EXECUTE** para cada uma delas e depois executar esse *script*, que fará a execução do comando linha a linha.

Para garantir a permissão de execução da *procedure* **SPalterarMeuNome** para uma determinada pessoa, o comando é:

grant execute on SPalterarMeuNome to usuario

Para dar permissão de execução na *procedure* para um grupo de pessoas, o comando é:

grant execute on SPalterarMeuNome to grupoRH

Quando criamos as tabelas no banco de dados, precisamos aplicar as permissões adequadas para os tipos de operações que a tabela pode sofrer. Por exemplo, os dados da tabela aluno podem ser vistos por todos os alunos e funcionários da Seção de Alunos, mas somente alguns funcionários dela e o próprio aluno poderão alterar os dados da tabela. Para garantir essas ações, usamos o comando **GRANT**, conforme sintaxe a seguir:

grant select on ALUNO to grupo_alunos

grant select on ALUNO to secao-alunos

grant update on ALUNO to aluno

grant delete on ALUNO to funcionário

Observação

Podemos aplicar todas as permissões (*select*, *insert*, *delete*, *update*) em uma tabela com a execução do comando **grant all on TABELA to usuário**.

No caso de necessitar retirar a permissão de algum objeto do banco de dados, você deve executar o comando **REVOKE**. Por exemplo, para remover a permissão de execução em todas as *procedures* do banco de dados, você deve executar o comando:

revoke execute to usuário

Para remover a permissão de execução de apenas uma *procedure* de um determinado usuário ou grupo de usuários, o comando é:

revoke execute on SPminhaProcedure to usuario

Como você pode ter notado, o comando **REVOKE** é o oposto do comando de **GRANT**, e as mesmas recomendações e observações feitas para o comando de **GRANT** se aplicam para o comando **REVOKE**.

5.2.4 *Data Transaction Language* (DTL)

É o tipo de linguagem que controla as transações que são executadas em um banco de dados. Transação é uma unidade de execução de programa que acessa e atualiza os dados.

A transação consiste em todas as operações executadas entre o começo (**BEGIN TRANSACTION**) e o fim (**END TRANSACTION**) da transação.

Quando executamos qualquer um dos comandos DML (**INSERT**, **DELETE**, **UPDATE**) ou de DDL (**CREATE**, **ALTER**, **DROP**), também estamos realizando uma transação dentro do banco de dados.

Durante o tempo em que a transação estiver em aberto, os comandos DML estão sendo realizados em memória. Assim que a transação for confirmada pela instrução denominada *commit*, todas as operações serão efetuadas fisicamente no banco de dados. No caso de falha de alguma das operações, a transação será desfeita pela instrução de *rollback*, e nenhuma operação será realizada no banco de dados.

O comando **COMMIT** efetiva todas as alterações realizadas no banco de dados. A operação de *commit* aplica-se a todos os comandos SQL, incluindo comandos DDL. As transações que estão bloqueadas (*lock*), aguardando o término de uma transação, são sempre liberadas depois do *commit* na transação que está causando o bloqueio.

O comando **ROLLBACK** finaliza uma transação atual. Quando ele é executado, o SQL aborta a transação atual. Isso restaura o banco de dados ou até o estado em que ele estava antes do último *commit* ou *rollback*.

Lembrete

Enquanto nenhuma das instruções, seja de *commit* ou de *rollback*, não for processada, as alterações ficam esperando e consumindo recursos de memória, por exemplo.

Os comandos DTL são normalmente utilizados dentro de procedimentos (*procedures*) ou *triggers*, pois esses objetos costumam ter diversas ações de alterações nos bancos de dados.

Os comandos da DTL são os seguintes:

- **BEGIN TRANSACTION**
- **COMMIT TRANSACTION**
- **ROLLBACK TRANSACTION**
- **END TRANSACTION**

5.2.5 *Data Query Language* (DQL)

É o tipo de linguagem que cuida da parte de *queries* (consultas) aos dados. Apesar de interagir com os dados como a DML, ela não produz alterações neles, serve apenas para consulta.

O formato-padrão para consulta é:

SELECT atributo FROM relação WHERE condição

Observação

Relação pode ser uma tabela física no banco ou várias tabelas relacionadas. Atributo é a coluna da tabela.

Em algumas bibliografias, o comando SQL está dentro de DML.

Exemplos de instrução DQL:

Consideremos a relação (tabela) Aluno e seus respectivos atributos, assim definidos:

- Aluno (nome, idade, curso, ra_aluno)

Para obter a lista de todos os atributos de todos os alunos da tabela aluno, você deve executar o comando:

SELECT * FROM aluno

ou

SELECT nome, idade, curso, ra_aluno FROM aluno

Quando usamos asterisco (*), queremos dizer que todas as colunas da tabela devem ser retornadas.

Para obter todas as informações do aluno cujo RA é 1234567, o comando é:

SELECT * FROM aluno

WHERE ra_aluno = '1234567'

Para selecionar o nome dos alunos que cursam *Gestão de Tecnologia da Informação*, o comando é:

SELECT nome FROM aluno

WHERE curso = 'Gestão de Tecnologia da Informação'

Para saber quais alunos têm mais de 25 anos:

SELECT nome FROM aluno

WHERE idade > 25

O comando **SELECT** é o mais utilizado dentro de um banco de dados. Ele serve para retornar dados de uma ou mais tabelas. Como ele não gera nenhuma modificação nos dados, você pode usá-lo antes de fazer a alteração. Por exemplo, suponha que você fosse alterar alguma informação do aluno cujo RA é 1234567. Antes de executar o comando **UPDATE**, você deve executar o comando **SELECT** e analisar o resultado. Isso evita o risco de realizar alterações indevidas.

Para retornar dados de mais de uma tabela, utilizamos o conceito de *join*. É extremamente importante conhecer o modelo de dados e como as tabelas estão relacionadas para obter resultados corretos.

O comando **JOIN** ou junção é baseado na teoria dos conjuntos. Considere as tabelas como um conjunto de dados sobre um determinado assunto.

Observação

A sintaxe do comando **JOIN** entre duas ou mais tabelas relacionadas é diferente entre os diversos dialetos do SQL. Para escrever o comando corretamente, você deverá consultar o manual do gerenciador de banco de dados.

Os exemplos de **JOIN** que serão mostrados a seguir estão usando a sintaxe do SQL padrão ANSI. Caso você queira testar os comandos em um banco de dados, você precisa estar atento se ele é compatível com o SQL ANSI, pois a sintaxe do **JOIN** difere muito.

Exemplo de sintaxe do comando **SELECT** com mais de uma tabela: vamos considerar as tabelas indicadas a seguir.

- Turma (id_turma, nome, sala, disciplina)
- Disciplina (disciplina, sigla, créditos)

Queremos a lista das salas de cada turma, indicando o nome da turma e a sigla de cada disciplina correspondente. O comando **SELECT** ficará assim:

SQL padrão ANSI:

```
SELECT sala, nome, sigla FROM Turma, Disciplina
INNER JOIN Disciplina
ON Turma.disciplina = Disciplina.disciplina
```

ou

Transact-SQL:

```
SELECT sala, nome, sigla FROM Turma, Disciplina
WHERE Turma.disciplina = Disciplina.disciplina
```

Para entender como esse **JOIN** funciona, com base na teoria dos conjuntos, vamos considerar as tabelas conjuntos, da seguinte forma: o conjunto das turmas será denominado conjunto A; o conjunto das disciplinas será o conjunto B, conforme a figura seguinte.

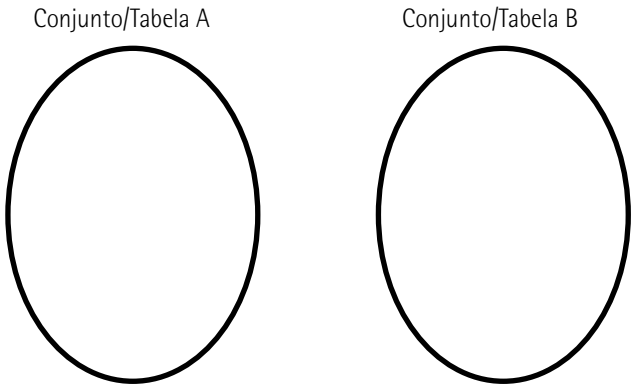

Figura 26 – Tabelas como diagramas de conjuntos

INNER JOIN – é o tipo de **JOIN** mais usado. Retorna apenas os dados que existem em comum nas duas tabelas. Com base na teoria dos conjuntos, podemos dizer que é a intersecção entre os conjuntos, conforme representado na figura a seguir.

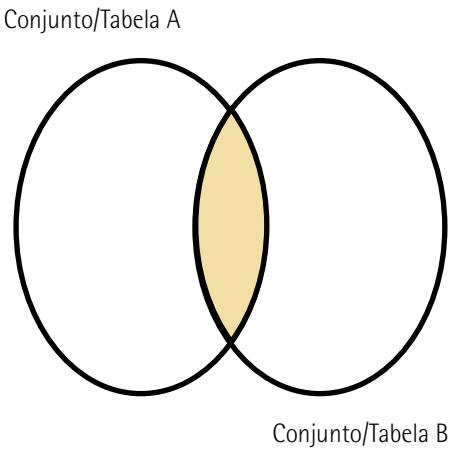

Figura 27 – Representação gráfica do INNER JOIN

Sintaxe do **INNER JOIN** usando o SQL padrão ANSI:

```
select * from A
    inner join B on A.atributo_chave = B.atributo_chave
```

E agora utilizando o Transact-SQL:

```
select * from A, B
    where A.atributo_chave = B.atributo_chave
```

LEFT JOIN – usado quando é necessário retornar todos os dados de uma tabela, mesmo que não exista na outra tabela. O **LEFT JOIN** retorna os dados da tabela da esquerda, conforme a figura a seguir.

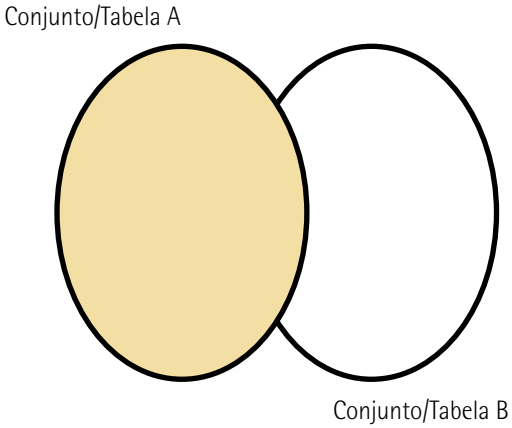

Figura 28 – Representação gráfica do LEFT JOIN

Sintaxe do **LEFT JOIN** usando o SQL padrão ANSI:

```
select * from A
    left join B on A.atributo_chave = B.atributo_chave
```

Agora, utilizando o Transact-SQL:

```
select * from A, B
    where A.atributo_chave *= B.atributo_chave
```

RIGHT JOIN – assim como o **LEFT JOIN**, é usado quando é necessário retornar todos os dados de uma tabela, mesmo que não existam na outra tabela. O **RIGHT JOIN** retorna os dados da tabela da direita, conforme a figura a seguir.

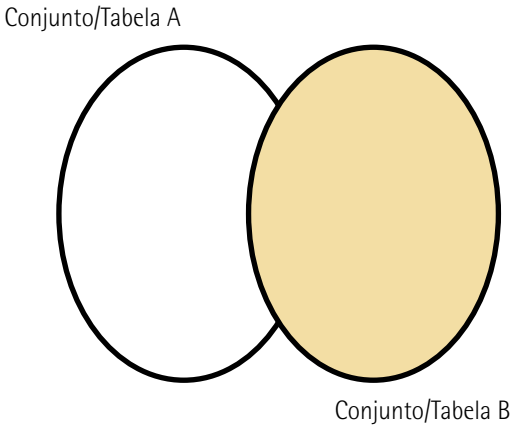

Figura 29 – Representação gráfica do RIGHT JOIN

Sintaxe do **RIGHT JOIN** usando o SQL padrão ANSI:

```
select * from A
    right join B on A.atributo_chave = B.atributo_chave
```

Agora, utilizando o Transact-SQL:

```
select * from A, B
    where A.atributo_chave =* B.atributo_chave
```

5.3 Módulos de funcionamento do SGBD

O núcleo de um SGBD é composto do módulo central, que é o módulo gerenciador do banco de dados, responsável pela transparência do acesso físico aos dados armazenados, seja para aplicações, seja para os usuários especializados e o DBA. O SGBD apenas solicita os dados a esse módulo, que se encarrega de localizar os registros fisicamente em disco e transferi-los para as estruturas de dados.

Esse módulo realiza a comunicação com o sistema operacional do equipamento onde se encontram os dados, por meio do mapeamento de registros em memória para as páginas de disco onde se encontram, ou vice-versa.

Em outras palavras, esse módulo é responsável pelo atendimento das requisições de dados e metadados feitas pelos módulos que interagem com os agentes. Atende também às solicitações de operações sobre dados enviados pelo código objeto das aplicações. Retornam dados e/ou *status* que indicam situações como execução realizada com sucesso, erros de acesso, violações de integridade ou permissão etc.

É o único módulo que se comunica com o módulo gerenciador de arquivos. Os controles de segurança e concorrência são de responsabilidade desse módulo central.

O módulo gerenciador de arquivos gerencia o acesso aos dispositivos de armazenamento. Recebe e realiza requisições para leitura e gravação de dados, metadados e dados de segurança do módulo gerenciador do banco de dados.

Responsável pela tradução de comandos DML, temos o módulo compilador DML. No caso em que os comandos DML são embutidos em um programa de aplicação, esse módulo realiza a pré-compilação e a geração do código objeto para esses comandos, criando um programa pré-compilado que é remetido para o compilador do ambiente de desenvolvimento utilizado pelo programador. Já no caso em que os comandos DML são enviados de forma *ad hoc* por usuários especializados, o módulo traduz e já executa o comando, caso esteja correto.

Em ambos os casos, o módulo compilador DML solicita metadados ao módulo gerenciador do banco de dados, para poder traduzir os comandos; caso a solicitação seja uma consulta, ele a remete ao módulo processador de consultas, recebendo-a modificada.

O módulo processador de consultas recebe um comando de consulta do módulo pré-compilador DML e define a melhor estratégia de acesso para processá-la. Para tanto, esse módulo necessita solicitar metadados ao módulo gerenciador do banco de dados, com o objetivo de analisar o tamanho dos arquivos envolvidos na consulta, os relacionamentos existentes, a estimativa de valores distintos de atributos, visando determinar um procedimento que processe a consulta mais rapidamente. Esse procedimento, uma vez determinado, é enviado ao módulo compilador DML para ser traduzido.

Para traduzir o programa-fonte da aplicação, escrito em alguma das linguagens de programação suportadas pelo SGBD, temos o módulo compilador do utilitário. Esse módulo retorna um código de *status* à ferramenta de desenvolvimento e gera, no caso de uma tradução bem-sucedida, o código objeto da aplicação.

Já o módulo compilador DDL traduz comandos DDL e gera os esquemas conceitual e físico, ou seja, solicita a criação do banco de dados da aplicação para o módulo gerenciador do banco de dados, que por sua vez solicita ao módulo gerenciador de arquivos a criação de arquivos de dados e metadados.

O módulo compilador DCL e de autorização gera procedimentos para verificação de integridade e permissão de acesso. Esses procedimentos são remetidos ao módulo gerenciador do banco de dados, que por sua vez solicita ao módulo gerenciador de arquivos seu armazenamento no dicionário de dados.

Os procedimentos para monitoramento de *performance* e configuração do banco de dados, e também recuperação de falhas, são funcionalidades do módulo de administração. A primeira classe de procedimentos solicita e/ou modifica dados de configuração e solicita estimativas sobre *performance* ao módulo gerenciador do banco de dados, que os obtém no dicionário de dados e/ou nos dispositivos que mantêm dados de segurança. A segunda classe de procedimentos solicita dados ao banco de dados de *recovery* para restaurar os dados do banco de dados na ocorrência de uma falha.

5.4 Etapas do projeto de banco de dados

O desenvolvimento de um sistema de informação envolve a análise e o projeto de dois componentes importantes: os dados e os processos.

O projeto de dados é considerado a parte estática do sistema, uma vez que diz respeito a um universo persistente de características que dificilmente sofre modificações após a sua definição.

O projeto de processos, por sua vez, é chamado de parte dinâmica, uma vez que as tarefas a serem realizadas sobre os dados podem variar, conforme ocorre a evolução do sistema.

Consideram-se projeto de um banco de dados a análise, o projeto e a implementação dos dados persistentes de uma aplicação, levando em conta a determinação da sua semântica (abstração dos dados de uma realidade) e, posteriormente, o modelo de dados e o SGBD a serem adotados.

O projeto de um banco de dados é composto de quatro etapas, descritas a seguir.

5.4.1 Levantamento de requisitos

É a fase principal de todo o projeto de banco de dados, em que é necessário **entender** do que o usuário realmente necessita. Nessa etapa, são coletadas informações sobre os dados de interesse da aplicação, o seu uso, as suas operações de manipulação sobre os dados e suas relações. Para a realização dessa etapa, são necessárias tarefas como:

- entrevistas com os futuros usuários do sistema;
- análise de documentações disponíveis (arquivos, relatórios etc.).

O resultado dessa etapa normalmente é uma descrição dos requisitos da aplicação, o mais detalhado e claro possível. Caso nenhuma metodologia de engenharia de *software* esteja sendo empregada, essa descrição de requisitos é informal, no formato de um texto, por exemplo, que narra as informações comentadas anteriormente.

5.4.2 Projeto conceitual

Pode ser considerada a fase de análise dos dados e dos requisitos capturados na etapa anterior. Nessa etapa, é realizado o que se chama de desenho da solução ou modelagem conceitual, momento em que são analisados os fatos (entidades ou conjunto de ocorrências de dados) de interesse e seus relacionamentos, juntamente com seus atributos (propriedades ou características), e é construída uma notação gráfica (abstrata, uma representação de alto nível) para facilitar o entendimento dos dados e suas relações, tanto para os analistas quanto para os futuros usuários.

Essa etapa resulta em um modelo conceitual, em que a semântica da realidade deve estar correta. A ferramenta mais empregada nessa etapa é o Diagrama Entidade-Relacionamento (DER). Ainda nessa fase, as reuniões com o usuário são constantes, para mais entendimentos sobre o projeto e validação do modelo conceitual.

5.4.3 Projeto lógico

Nessa etapa, o desenvolvimento do banco de dados começa a se voltar para o ambiente de implementação, uma vez que é feita a conversão do modelo de dados de um banco de dados (modelo lógico). Esse modelo de dados pode ser o modelo relacional, orientado a objetos, multidimensional etc.

Essa etapa se baseia no uso de regras de mapeamento de um DER para o modelo de dados escolhido. O resultado é uma estrutura lógica, como um conjunto de tabelas relacionadas, caso o modelo de dados escolhido seja o relacional. Aqui, definimos efetivamente a chave primária, bem como refinamos os relacionamentos que vêm do diagrama construído na etapa anterior.

Nessa etapa, realizamos o processo de normalização das tabelas, a fim de remover as redundâncias e inconsistências.

5.4.4 Projeto físico

Essa última etapa realiza a adequação do modelo lógico gerado na etapa anterior ao formato de representação de dados do SGBD escolhido para a implementação.

Para a realização dessa etapa, devem-se conhecer a DDL e a DCL do SGBD, para realizar a descrição do modelo lógico. O resultado é a especificação do esquema da aplicação, juntamente com a implementação de restrições de integridade e visões.

Observação

Nessa etapa do projeto, criamos um "mapa" do banco de dados, da forma como ele será criado em SQL. Aqui, definimos os tipos de dados de cada atributo e, quando necessário, alteramos os nomes dos campos e das tabelas para que se ajustem ao padrão da organização.

5.4.5 Criação do banco de dados

Essa é a etapa da programação propriamente dita. Aqui, seguindo o que foi definido no modelo físico, serão criados todos os objetos dentro do banco de dados, utilizando a linguagem SQL.

5.4.6 Exemplo de projeto de banco de dados

1ª. etapa: levantamento de requisitos

Seu usuário pede que você crie um cadastro de pessoas. Durante a entrevista, você fará diversas perguntas para saber o que ele pretende com esse cadastro de pessoas. Além do nome e do CPF, ele quer ter o cadastro do telefone. No final dessa etapa, você terá um documento descrevendo da forma mais detalhada possível todas as operações que envolvem esse cadastro de pessoas.

2ª. etapa: projeto conceitual

Nessa etapa, você criará o modelo conceitual, identificando e analisando os fatos ou as entidades, e construirá uma representação gráfica para facilitar o entendimento do usuário, conforme pode ser visto na figura a seguir.

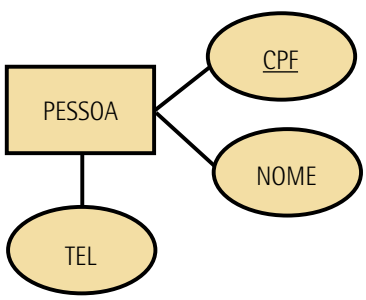

Figura 30 – Representação gráfica do modelo conceitual

3ª. etapa: projeto lógico

Nessa etapa, com o modelo conceitual analisado e aprovado pelo usuário, você começará a fazer a conversão para um modelo de dados adequado com as especificações realizadas nas etapas anteriores. Em nosso exemplo, podemos usar o modelo relacional, pois, no caso de o usuário querer incluir no seu cadastro de pessoas outra informação – como dados do endereço residencial, dados sobre as características dessa pessoa dentro do sistema –, você terá que alterar o seu modelo e incluir mais tabelas e seus respectivos relacionamentos. Por enquanto, a estrutura lógica que temos está representada na figura a seguir. Até aqui, sabemos que já temos uma tabela denominada **pessoa**, com os atributos **CPF**, **nome** e **tel**, e o atributo que irá garantir a identificação de uma única pessoa será o **CPF**, ou seja, o atributo candidato a ser considerado como chave primária já está definido.

Pessoa	
PK	**CPF**
	NOME TEL

Figura 31 – Modelo lógico de uma entidade

4ª. etapa: projeto físico

Nessa etapa, estamos prontos para criar a tabela no banco de dados, e, se já existir um arquivo com os dados, também poderemos providenciar a carga da primeira massa de dados. Tudo será realizado no banco de dados escolhido e com a utilização dos comandos DDL e DCL.

Para facilitar o trabalho, podemos gerar um arquivo (*script*) com a sequência dos comandos de criação da tabela, os comandos de permissão de acesso e manipulação e também os comandos para inserir a primeira massa de dados.

Pessoa		
PK	CPF	CHAR(11)
	NOME TEL	VARCHAR(100) CHAR(10)

Figura 32 – Modelo físico de uma entidade

Exemplo do arquivo (*script*) de criação dos objetos de banco de dados:

```
create table PESSOA      ( CPF char(11) not null,
                           NOME varchar(100) not null,
                           TEL char(10) null,
constraint PK_PESSOA primary key (CPF) )

grant select on PESSOA to grupo_usario
grant insert on PESSOA to usuario1
grant update on PESSOA to usuario1
```

```
grant delete on PESSOA to usuario1
BEGIN TRANSACTION

insert to PESSOA (CPF, NOME, TEL)
values ('13515794-22', 'João da Silva', '(011) 2244-56789'

insert to PESSOA (CPF, NOME, TEL)
values ('9995794-22', 'Pedro Toledo, '(011) 7644-53789'

insert to PESSOA (CPF, NOME, TEL)
values ('5691800-09, 'Clara Nunes', '(011) 3450-9900'

insert to PESSOA (CPF, NOME, TEL)
values ('09873456-35', 'Luis Alves, null)

COMMIT TRANSACTION

insert to PESSOA (CPF, NOME, TEL)
values ('19565797-02', 'Paulo da Silva')

insert to PESSOA (CPF, NOME, TEL)
values ('4569579-12', 'Carlos Manuel, '(011) 9084-13689'

insert to PESSOA (CPF, NOME, TEL)
values ('7891800-49, 'Maria Oliveira', '(011) 3400-9000'

insert to PESSOA (CPF, NOME, TEL)
values ('09803454-36', 'Ricardo Veiga, null)

COMMIT TRANSACTION

END TRANSACTION
```

6 TIPOS DE DADOS (*DATA TYPES*)

Um banco de dados existe para armazenar dados de forma efetiva para posteriormente recuperá-los. Para auxiliar nessa tarefa, existem tipos diferentes de dados para melhor armazená-los.

Os bancos de dados possuem vários tipos e subtipos de dados. Recomendados a leitura do manual do banco de dados fornecido pelo fabricante para você conhecer os tipos de dados. É muito comum fazer algumas conversões de um tipo de dado para outro durante uma consulta dos dados armazenados no banco de dados.

Alguns gerenciadores de bancos de dados conseguem fazer essas conversões diretamente, sem a necessidade de explicitar por meio de funções próprias de conversão. Os elementos de dados básicos e as estruturas de dados mais comuns, utilizadas em diversas linguagens e bancos de dados, estão a seguir.

6.1 Elementos de dados

- *Byte*, *Integer*: números inteiros positivos, negativos ou o zero.
- Real: números inteiros positivos ou negativos compostos por uma parte inteira e outra fracionada. Exemplo: 8,2.
- Char: caractere alfanumérico como letras, algarismos ou símbolos especiais ocupando uma única posição de memória. Na verdade, esse elemento armazena um numero da tabela ASCII.
- *Boolean*: armazena estados de uma operação lógica, podendo receber valores de verdadeiro ou falso, armazenando o *bit* 0 ou 1 dependendo do resultado.

Quando os dados estão organizados de forma coerente, temos uma estrutura de dados. As estruturas também são chamadas tipos de dados compostos e podem ser:

- **Homogêneos**: conjunto de dados formado pelo mesmo elemento de dado.
- **Heterogêneos**: conjunto de dados formado por mais de um elemento de dado.

Alguns bancos de dados, além de ter uma vasta coleção de elementos de dados, permitem a criação de novos elementos, por exemplo, o PostgreSQL, que permite a criação por meio do comando **CREATE TYPE**.

Existem elementos de dados nativos de um banco de dados que não existem em outros bancos de dados.

6.1.1 Estrutura de dados

As mais comuns serão descritas a seguir. Lembrando que nem todas as estruturas existem nos diferentes bancos de dados. Por exemplo, a estrutura de dados conhecida como vetor não existe no banco de dados Sybase, mas por meio do conceito de cursor podemos lidar com os dados com a mesma funcionalidade dos vetores.

- **Vetores**

Também conhecidos como *array*, são uma estrutura de dados simples linear e estática que mantém uma série finita de elementos de dados do mesmo tamanho e tipo.

- **Lista**

Cada elemento de dados referencia o seu sucessor. Os tipos de lista são:

– **Lista encadeada**

Os elementos são armazenados em sequência, não tendo como acessar um segundo elemento sem acessar o primeiro.

– **Lista ordenada**

Os elementos são armazenados seguindo algum critério de ordenação.

- **Fila**

Estruturas que se comportam como filas tradicionais e têm como política de funcionamento o FIFO (*First in, first out* – primeiro que entra, primeiro que sai). As inserções são realizadas no final, e a remoção no início.

Existem duas operações nas filas, que são:

- *Enqueue*: adiciona um elemento ao final da fila.
- *Dequeue*: remove um elemento do início da fila.

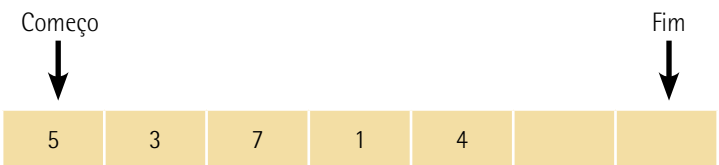

Figura 33 – Exemplo de estrutura de dados em fila

- **Pilha**

É baseada no princípio LIFO (*Last in, first out* – último que entra, primeiro que sai). O topo é o único local possível de inserir elementos e a remoção da pilha só ocorre nas extremidades do topo, isto é, os elementos são removidos na ordem inversa da inserção.

As funções que se aplicam a pilhas são:

- **PUSH**: insere um dado no topo da pilha.
- **POP**: remove o item no topo da pilha.
- **TOP**: retorna o elemento no topo.

- Árvores

Os dados estão dispostos de forma hierárquica, tendo seu elemento principal chamado de raiz, que possui ligação com os outros elementos denominados ramos. Uma árvore binária é aquela em que cada ramo possui mais dois ramos, como na figura a seguir.

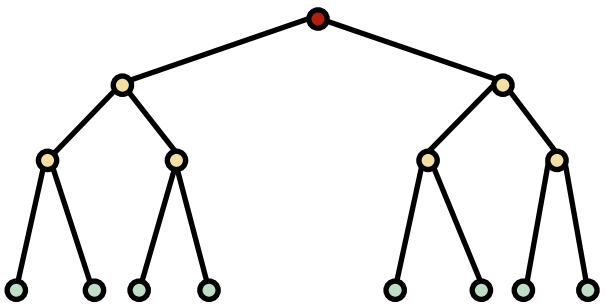

Figura 34 – Exemplo de estrutura de dados em árvore

Saiba mais

Para se aprofundar no assunto, recomendamos alguns livros de estrutura de dados, que estão relacionados na bibliografia ao final do livro-texto.

6.1.2 Unidades de medidas

Os computadores "entendem" impulsos elétricos, positivos ou negativos, que são representados por 1 ou 0. A cada impulso elétrico, damos o nome de *bit* (*BInary digiT*). Um conjunto de 8 *bits* reunidos como uma única unidade forma um *byte*. A seguir, apresentamos algumas conversões de medidas.

1 *byte* = 8 *bits*

1 *kilobyte* (KB ou *Kbytes*) = 1024 *bytes*

1 *megabyte* (MB ou *Mbytes*) = 1024 *kilobytes*

1 *gigabyte* (GB ou *Gbytes*) = 1024 *megabytes*

1 *terabyte* (TB ou *Tbytes*) = 1024 *gigabytes*

1 *petabyte* (PB ou *Pbytes*) = 1024 *terabytes*

1 *exabyte* (EB ou *Ebytes*) = 1024 *petabytes*

1 *zettabyte* (ZB ou *Zbytes*) = 1024 *exabytes*

1 *yottabyte* (YB ou *Ybytes*) = 1024 *zettabytes*

É também por meio dos *bytes* que se determina o comprimento da palavra de um computador, ou seja, a quantidade de *bits* que o dispositivo utiliza na composição das instruções internas, por exemplo:

8 *bits* → palavra de 1 *byte*

16 *bits* → palavra de 2 *bytes*

32 *bits* → palavra de 4 *bytes*

Na transmissão de dados entre dispositivos, geralmente, usam-se medições relacionadas a *bits* e não a *bytes*. Assim, há também os seguintes termos:

1 *kilobit* (Kb ou *Kbit*) = 1024 *bits*

1 *megabit* (Mb ou *Mbit*) = 1024 *kilobits*

1 *gigabit* (Gb ou *Gbit*) = 1024 *megabits*

1 *terabit* (Tb ou *Tbit*) = 1024 *gigabits*

Observação

Quando a medição é baseada em *bytes*, a letra "b" da sigla é maiúscula (como em GB). Quando a medição é feita em *bits*, o "b" da sigla fica em minúsculo (como em Gb).

Como já dito, a utilização de medições em *bits* é comum para informar o volume de dados em transmissões. Geralmente, indica-se a quantidade de *bits* transmitidos por segundo. Assim, quando queremos dizer que um determinado dispositivo é capaz de trabalhar, por exemplo, com 54 *megabits* por segundo, usa-se a expressão 54 Mb/s:

1 Kb/s = 1 *kilobit* por segundo

1 Mb/s = 1 *megabit* por segundo

1 Gb/s = 1 *gigabit* por segundo

Resumo

Vimos nesta unidade que a etapa de projeto conceitual de um banco de dados consiste na elaboração de um modelo conceitual, ou seja, um modelo de representação de categorias de fatos do mundo real (entidades) e dos relacionamentos existentes entre eles.

É dito um modelo de alto nível, pois não está vinculado a nenhum modelo de banco de dados. Sua intenção é facilitar a real compreensão dos fatos de uma realidade (semântica da realidade).

A etapa de projeto conceitual é uma das mais importantes do projeto do banco de dados, pois, se for mal definida, irá gerar uma estruturação de dados ineficiente no esquema implementado em um banco de dados.

As principais vantagens da confecção de um modelo conceitual são:

- melhor compreensão do modelo por parte de um usuário leigo;
- independência de detalhes de implementação, podendo ser facilmente modificado sem comprometer as outras etapas;
- tradução para qualquer modelo de dados (relacional, orientado a objetos, multidimensional);
- ferramenta indispensável para o processo de engenharia reversa de um banco de dados (isto é, obter o modelo conceitual a partir do modelo lógico de um banco de dados);
- maior estabilidade diante das mudanças de implementação;
- mais adequado para o exercício da criatividade durante a interpretação da realidade para ser traduzida para o modelo.

Exercícios

Questão 1. O Modelo Entidade Relacionamento (MER) é baseado na percepção do mundo real. O modelo representa a visão das entidades de dados envolvidas no problema analisado. Pode-se afirmar que ele consiste em um conjunto de objetos básicos chamados entidades e nos relacionamentos entre esses objetos. O objetivo do MER é facilitar o projeto de banco de dados, possibilitando a especificação da estrutura lógica geral do banco de dados. No modelo desenha-se um diagrama chamado de DER (Diagrama Entidade-Relacionamento) que mostra a estrutura lógica geral de um banco de dados. Ele

serve para checar se todos os pedidos dos usuários estão sendo atendidos e se não há conflitos entre eles. Não há preocupação com armazenamento físico.

Considerando os conceitos sobre Projeto de Banco de Dados, examine as afirmações a seguir e indique a alternativa **incorreta**:

A) O modelo de dados é a representação abstrata e simplificada de uma realidade, com o qual se pode explicar ou testar o seu comportamento.

B) O modelo de dados é uma coleção de conceitos que podem ser usados para descrever a estrutura de um banco de dados (tipos de dados, relacionamento e restrições entre eles).

C) Os modelos de dados não permitem a compreensão da estrutura dos dados armazenados e a sua manipulação.

D) Os modelos de dados, para efeito de organização, foram divididos em: modelo conceitual (projeto conceitual), modelo de implementação ou baseados em registros (projeto lógico), e modelo físico (projeto físico).

E) O modelo conceitual é usado na descrição do banco de dados, sendo independente de implementação e do monitor de banco de dados (SGBD).

Resposta correta: alternativa C.

Análise das alternativas

O modelo de dados (MER) é uma abordagem que foi criada por Peter Chen em 1976 e que permitiu a análise dos dados a serem armazenados independentes da solução de banco de dados oferecidos pelos fornecedores. É até hoje considerada como um padrão para a modelagem conceitual.

A) Alternativa correta.

Justificativa: o modelo de dados (MER) tem por base que o mundo real é formado por um conjunto de objetos chamados de entidades e pelo conjunto dos relacionamentos entre esses objetos.

B) Alternativa correta.

Justificativa: o objetivo do MER é representar a estrutura lógica do banco de dados de uma empresa, especificando o esquema da empresa, quais as entidades e como elas se relacionam.

C) Alternativa incorreta.

Justificativa: os autores definem que modelos são mentais e que são pressupostos profundamente arraigados, generalizações ou mesmo imagens que influenciam a forma de ver o mundo e de nele agir.

Muitas vezes, não estamos conscientes de nossos modelos mentais ou de seus efeitos sobre nosso comportamento. Essa definição na Engenharia de *Software* cobre todos os modelos previstos, inclusive a modelagem dos dados de um Sistema de Informação.

D) Alternativa correta.

Justificativa: a abordagem da modelagem de dados é uma visão do assunto que normalmente atende a três perspectivas: modelagem conceitual, modelagem lógica e modelagem física. A primeira é usada como representação de alto nível e considera exclusivamente o ponto de vista do usuário criador do dado ou proprietário, a segunda já agrega alguns detalhes de implementação e a terceira demonstra como os dados são fisicamente armazenados.

E) Alternativa correta.

Justificativa: o modelo conceitual é uma descrição mais abstrata do banco de dados e é o ponto de partida para o projeto do banco de dados.

Questão 2. Durante o projeto de um banco de dados é fundamental o estudo e a aplicação da normalização de dados. A normalização é um conjunto de passos que são aplicados e que permitem um armazenamento consistente e um eficiente acesso aos dados em bancos de dados relacionais. Esses passos reduzem a redundância de dados e as chances de se tornarem inconsistentes. No entanto, muitos monitores de banco de dados relacionais (SGBDs) não permitem uma abstração suficiente entre o projeto lógico da base de dados e a implementação física do banco de dados, e isso tem como consequência um desempenho fraco nas consultas mais complexas efetuadas no banco de dados. Nesses casos, usa-se por vezes a desnormalização para melhorar o desempenho, com o custo de menores garantias de consistência.

Considerando os conceitos sobre projeto de banco de dados, examine as afirmações a seguir e indique a alternativa **incorreta**:

A) Na normalização de um banco de dados relacional, a primeira Forma Normal (ou 1FN) requer que todos os valores de colunas em uma tabela sejam atômicos.

B) Na normalização de um banco de dados relacional, a segunda Forma Normal (ou 2FN) requer que não haja dependência funcional não trivial de um atributo que não seja a chave, em parte da chave candidata.

C) Na normalização de um banco de dados relacional, a terceira Forma Normal (ou 3FN) requer não haver dependências funcionais não triviais de atributos que não sejam chave, em qualquer coisa exceto um superconjunto de uma chave candidata.

D) Existem outras formas de normalização de banco de dados relacional, mas que raramente são aplicados a banco de dados comerciais e nem sempre os SGBDs dão suporte a elas, como a quarta e quinta forma normal.

E) Os bancos de dados hierárquicos e em redes também necessitam que os dados sejam normalizados com o uso de chaves primárias e chaves estrangeiras.

Resposta correta: alternativa E.

Análise das alternativas

A) Alternativa correta.

Justificativa: a normalização para a primeira forma normal elimina grupos repetidos, colocando cada deles um em uma tabela separada, conectando-os com uma chave primária ou estrangeira.

B) Alternativa correta.

Justificativa: uma relação ou tabela no banco de dados relacional está na 2FN se, e somente se, estiver na 1FN e cada atributo não chave for dependente da chave primária inteira, isto é, cada atributo não chave não poderá ser dependente de apenas parte da chave.

C) Alternativa correta.

Justificativa: para cada chave candidata de R, uma relação R está na 3FN se estiver na 2FN e cada atributo não chave de R não possuir dependência transitiva.

D) Alternativa correta.

Justificativa: os autores acreditam que, para a maioria dos efeitos práticos, considera-se que as bases de dados estão normalizadas se aderirem à terceira forma normal.

E) Alternativa incorreta.

Justificativa: a normalização aplica-se somente aos bancos de dados relacionais, pois eles foram desenvolvidos a partir da teoria das relações matemáticas.