UNIDADE 2 – MODELO DE DADOS RELACIONAL E SQL
MÓDULO 1 – MODELO DE DADOS E RESTRIÇÕES

01

1 - CONCEITOS DO MODELO RELACIONAL

Olá, seja bem-vindo a mais uma etapa do nosso estudo. Neste módulo, iremos tratar especificamente dos **bancos de dados relacionais**.

O conceito de banco de dados relacional surgiu em 1970, como um aprimoramento das estruturas lineares. Esse novo modelo teve uma grande aceitação pelo mercado de TI, devido a sua simplicidade e benefícios imediatos em relação à melhor organização (acesso e manipulação) das informações. O modelo relacional tem sua origem na matemática, com sua teoria de conjuntos e relações.

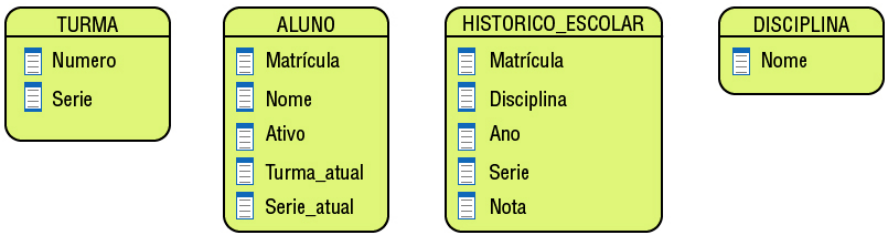
Logo em seguida, as grandes corporações como IBM e Oracle lançaram produtos que já utilizavam essa nova tecnologia, trazendo um ganho enorme em performance, em relação à tecnologia anterior. A partir de então, todos os fornecedores de SGBD adotaram o modelo relacional como tecnologia padrão.

O **modelo relacional** representa o banco de dados como uma coleção de relações (conjuntos de temas ou assuntos específicos que chamaremos de entidades). Informalmente, cada relação é semelhante a uma tabela de valores ou, até certo ponto, a um arquivo plano de registros. Ele é chamado de **arquivo plano** porque cada registro tem uma simples estrutura linear ou plana. No entanto, existem diferenças importantes entre relações e arquivos, conforme veremos em breve.

Quando uma relação é considerada uma tabela de valores, cada linha na tabela representa uma **coleção de valores de dados relacionados**. Uma linha representa um fato que normalmente corresponde a uma entidade ou relacionamento do mundo real. Os nomes da tabela e de coluna são usados para ajudar a interpretar o significado dos valores em cada linha.

02

O exemplo a seguir, já apresentado anteriormente, representa a estrutura de um banco de dados de uma escola, onde podemos ver quatro entidades, cada uma com seus campos específicos.



Exemplo de modelo relacional

Por exemplo, a segunda entidade (tabela) da imagem anterior é chamada de ALUNO porque cada linha de conteúdo representa fatos sobre uma entidade particular de aluno. Os nomes de cada coluna — Matrícula, Nome, Efetivo, Turma_atual e Serie_atual — especificam como interpretar os valores dos dados em cada linha (cada cadastro de aluno), com base na coluna em que cada valor se encontra. Todos os valores em uma coluna são do mesmo tipo de dado. Veja na tabela abaixo a representação hipotética do conteúdo da tabela ALUNO, com seus cinco campos (as colunas) e quatro registros (as linhas com alunos cadastrados):

ALUNO				
Matrícula	**Nome**	**Efetivo**	**Turma Atual**	**Série Atual**
14562/2	Thiago Ferreira Borges	Sim	5-1	5
432/2	Isadora Luccas Fernandes	Sim	7-1	7
332/5	Marcelo Correia Luz	Não		
4539/1	Mariana Gonçalves Coelho	Sim	6-1	6

Exemplo de conteúdo de uma tabela.

Na terminologia formal (e acadêmica) do modelo relacional, uma linha é chamada de **tupla**, um cabeçalho da coluna é chamado de **atributo** e a tabela é chamada de **relação** (cuidado, relação aqui não é o relacionamento entre duas tabelas!). O tipo de dado que descreve os tipos de valores que podem aparecer em cada coluna é representado por um **domínio** de valores possíveis. Agora, vamos definir esses termos — domínio, tupla, atributo e relação — de maneira formal.

03

1.1. Domínios, atributos, tuplas e relações

Um **domínio** é um conjunto de valores válidos para um determinado contexto.

Um método comum de especificação de um domínio é definir um tipo de dado do qual são retirados os valores de dados que formam o domínio. Também é útil especificar um nome para o domínio, para ajudar na interpretação de seus valores.

Alguns exemplos de domínios são:

- `Numeros_telefone_nacional;`
- `Numeros_telefone_local;`
- `Cadastro_pessoa_fisica;`
- `Nomes;`
- `Medias_nota;`
- `Idade_aluno.`

Estas são denominadas definições **lógicas** de domínios. Um **tipo de dado** ou **formato** também é especificado para cada domínio. Por exemplo, o tipo de dado para o domínio `Numeros_telefone_nacional` pode ser declarado como uma sequência de caracteres na forma "`(dd) ddddd-dddd`", onde cada "`d`" é um dígito numérico (decimal) e os dois primeiros dígitos formam um código de área de telefone válido. Dessa forma, um domínio recebe um nome, tipo de dados e formato.

Outras informações complementares para interpretar os valores de um domínio também podem ser informadas ou mesmo ser obrigatoriamente necessárias; por exemplo, o domínio `Idade_aluno` sabemos que a unidade de medida utilizada comumente é em anos. Por outro lado, um domínio de nome `Largura` poderia necessitar de unidade de medidas indicada explicitamente em metros, centímetros ou outra.

Numeros_telefone_nacional
É o conjunto de números de telefone com dez ou onze dígitos válidos no Brasil (o número do DDD com dois algarismos e demais número do telefone com oito ou nove dígitos, no formato "(dd) ddddd-dddd"). Neste contexto, "(00) 00000001" não é um valor que pertence a esse domínio, pois o número de telefone que esse valor representa não é um número de telefone válido. Já o valor "(61) 34030000" parece pertencer ao domínio, o prefixo 61 representa código DDD do Distrito Federal e os caracteres restantes, 34030000, parece ser um número de telefone válido (na verdade é o telefone da Faculdade AIEC).

Numeros_telefone_local
É o conjunto de números de telefone de oito ou nove dígitos válidos dentro de um código de área em particular no Brasil. Semelhante ao exemplo anterior, 10000000 não parece ser um número de telefone local válido, já 34030000 parece ser um número válido.

Cadastro_pessoa_fisica
É o conjunto de números do CPF com onze dígitos. Esse é um identificador exclusivo atribuído a cada pessoa no Brasil para fins de emprego, imposto e benefícios. Novamente, "000000000-01" não é um número de CPF válido, já "321445321-84" parece ser.

Nomes
É o conjunto de cadeia de caracteres que representam nomes de pessoas. Como exemplo, "João da Costa Barros" é um nome válido para uma pessoa, já "joao@gmail.com" não é, na verdade, parece ser um endereço de e-mail.

Medias_nota
São possíveis valores para calcular a média das notas de um aluno; cada um deve ser um número real (ponto flutuante) entre O e 10, com apenas uma casa decimal. Dessa forma: "4,5", "9" e "2,5" são notas válidas, já "-4", "12,3" e "7,44356" não são valores válidos.

`Idade_aluno`
Representa as idades possíveis dos alunos da escola, cuja regra (hipotética para uma escola qualquer) é que cada valor deve ser um número inteiro entre 5 e 18.

04

Um **esquema relacional** representa a forma textual de descrever uma relação (tabela) e seus respectivos atributos (campos).

O padrão para representar um esquema relacional é: `nome_da_relação (atributo 1, atributo 2, atributo 3, ..., último atributo)`. Dessa forma, o esquema relacional do modelo da escola descrito no início desse módulo seria representado por:

- `TURMA (Numero, Serie).`
- `ALUNO (Matricula, Nome, Efetivo, Turma_atual, Serie_atual).`
- `HISTORICO_ESCOLAR (Matricula, Disciplina, Ano, Serie, Nota).`
- `DISCIPLINA (Nome).`

Não há padrão ou regra universal sobre a utilização de letras em maiúsculas ou uso do caractere de sublinhado, "_", para representar espaços entre palavras. Cada empresa, equipe ou projeto define e utiliza os padrões de nomenclatura que achar apropriado. Portanto, dizer que o nome físico da entidade que representa o histórico escolar dos alunos será "HistoricoEscolar", "historico_escolar", "HISTORICO_ESCOLAR", "TblHistoricoEscolar" ou qualquer outra possibilidade é apenas um padrão a ser adotado, não há formas certas, erradas, piores ou melhores, são apenas padrões. O que quase todos nós evitamos (embora seja possível) é utilizar letras acentuadas ou espaços em branco, portanto, nomes como "Histórico_Escolar" e "Historico Escolar" devem ser evitados. A mesma regra é válida para atributos e demais objetos do modelo relacional (triggers, índices, relacionamentos, procedimentos armazenados etc.).

05

Utiliza-se o termo **grau** para descrever a quantidade de atributos de uma relação.

No exemplo anterior, a relação turma tem grau dois `((Numero, Serie,` enquanto que a relação aluno tem grau cinco `(Matricula, Nome, Efetivo, Turma_atual, Serie_atual).`

Outra forma de descrever um esquema relacional é incluir o **tipo de dado** de cada atributo. Isso facilita a futura especificação e desenho do modelo relacional. Os tipos básicos são:

- **string** (para qualquer tipo de caractere, como letras, números e símbolos),
- **integer** (para números inteiros, positivos e negefetivos),
- **real** (para números fracionários),
- **boolean** ou **bit** (para verdadeiro ou falso).

O tipo de dado é descrito logo após o nome do atributo, no formato `Nome_do_atributo: Tipo_de_dados`. Dessa forma, teríamos a seguinte especificação para a evolução do nosso esquema relacional:

- `TURMA (Numero: string, Serie: integer)`
- `ALUNO (Matricula: string, Nome: string, Efetivo: boolean, Turma_atual: string, Serie_atual: integer)`
- `HISTORICO_ESCOLAR (Matricula: string, Disciplina: string, Ano: integer, Serie: integer, Nota: real)`
- `DISCIPLINA (Nome: string)`

06

O conteúdo de informações que registramos no ambiente conceitual de uma relação é denominado **tupla**. Cada tupla representa um conjunto completo de informações acerca de cada registro.

Dessa forma, se você observar a tabela no início desse módulo, para a relação ALUNO, temos quatro tuplas, sendo a primeira: t1 (14562/2, Thiago Ferreira Borges, Sim, 5-1, 5).

ALUNO				
Matrícula	**Nome**	**Efetivo**	**Turma Atual**	**Série Atual**
14562/2	Thiago Ferreira Borges	Sim	5-1	5
432/2	Isadora Luccas Fernandes	Sim	7-1	7
332/5	Marcelo Correia Luz	Não		
4539/1	Mariana Gonçalves Coelho	Sim	6-1	6

Cada **tupla** na relação representa uma entidade de aluno em particular (ou objeto). Apresentamos a relação como uma tabela, onde cada tupla aparece como uma linha e cada **atributo** corresponde a um cabeçalho de coluna, indicando um papel ou interpretação dos valores nesta coluna. Valores NULL representam atributos cujos valores são desconhecidos ou não existem para alguma tupla. Exemplo, quando um aluno não está matriculado em nenhuma turma, o atributo turma_aluno possui valor NULL.

Lembre-se de que estamos tratando das mesmas informações sobre o tema "bancos de dados" em ambientes distintos; utilizamos uma nomenclatura para o **ambiente conceitual** (que é mais acadêmico e anterior à própria existência de "bancos de dados") e outra para o **ambiente físico**, quando falamos especificamente de SGBD. Portanto, enquanto a **tupla** se refere ao ambiente conceitual, o **registro** é algo localizado fisicamente dentro do ambiente de banco de dados, em um arquivo de dados, num disco rígido.

07

1.2 - Características das relações

A definição dada de relações implica certas características que tornam uma relação diferente de um arquivo ou uma tabela. Agora iremos tratar sobre essas relações.

1.2.1 - Ordenação de tuplas em uma relação

Uma relação é definida como um conjunto de tuplas. Matematicamente, os elementos de um conjunto não possuem ordem entre eles; logo, as tuplas em uma relação não possuem nenhuma ordem em particular. Em outras palavras, uma relação não é sensível à ordenação das tuplas. Porém, em um arquivo, os registros estão fisicamente armazenados no disco (ou na memória), de modo que sempre existe uma ordem entre eles (mesmo que aleatória). Essa ordenação indica o primeiro, segundo, os próximos, até o último registro no arquivo. De modo semelhante, quando exibimos uma relação como uma tabela, as linhas são exibidas em certa ordem.

A ordenação da tupla não faz parte da definição da relação porque uma relação tenta representar fatos em um nível lógico ou abstrato. Muitas ordens de tupla podem ser especificadas na mesma relação. Por exemplo, as tuplas na relação ALUNO poderiam ser ordenadas pelos valores de Matrícula, Nome, Turma, Série, ou até mesmo se é um aluno efetivo ou ex-aluno. A definição de uma relação não especifica ordem alguma: não existe preferência para ordenação de qualquer outra relação, Logo, uma relação ordenada alfabeticamente pelo nome do aluno é considerada idêntica à outra ordenada pelo número da matrícula.

Quando uma relação é implementada como um arquivo ou exibida como uma tabela, uma ordenação em particular pode ser especificada sobre os registros do arquivo ou das linhas da tabela.

08

1.2.2 - Valores e NULL nas tuplas

Já falamos anteriormente que cada valor em uma tupla é um valor atômico, ou seja, ele não é divisível em componentes dentro da estrutura do modelo relacional básica. Logo, atributos compostos ou

multivalorados (como vetores e objetos) não são permitidos. Esse modelo às vezes é chamado de **modelo relacional plano**.

Grande parte da teoria por trás do modelo relacional foi desenvolvida com essa suposição em mente, que é chamada pressuposto da **primeira forma normal** (falaremos sobre normalização em outro momento no nosso curso). Assim, atributos multivalorados precisam ser representados por relações separadas, e os atributos compostos são representados apenas por seus atributos de componentes simples no modelo relacional básico. Como você pode observar, por exemplo, a tabela de histórico escolar apresenta uma linha para cada nota de cada aluno. Não é possível ter na mesma linha todas as notas que ele tirou.

Um conceito importante é o dos **valores nulos**, que são usados para representar os valores de atributos que podem ser desconhecidos ou não se aplicam a uma tupla. Um valor especial, chamado NULL, é usado nesses casos.

Há uma diferença de **semântica SQL** entre valor em branco (“”) e valor nulo (`NULL`). Mesmo que ambos sejam interpretados da mesma forma, eles são diferentes para o controle do SGBD. Ao realizar uma pesquisa no banco de dados é necessário que você inclua ambas possibilidades para localizar esses atributos. Por exemplo, suponha que você tenha uma tabela denominada ALUNO e você queira identificar todos aqueles que não tem uma turma atribuída. A consulta SQL que representa essa consulta é:

```
SELECT * FROM ALUNO WHERE turma ISNULL OR turma = “”
```

Observe no exemplo acima que foram utilizadas duas expressões, a **`ISNULL`**, para atributos nulos e a **`“”`**, para atributos em branco.

09

O significado exato de um valor `NULL` determina como ele será aplicado durante agregações aritméticas ou comparações com outros valores. Por exemplo, uma comparação de dois valores `NULL` leva a ambiguidades — se os Clientes A e B têm endereços `NULL`, isso não significa que eles têm o mesmo endereço. Durante o projeto do banco de dados, é melhor evitar ao máximo valores `NULL`. Discutiremos isso melhor em outro momento no nosso estudo.

Outro problema comum do uso de nulo é a atribuição dada às variáveis durante a programação dos sistemas de informação. Os programadores geralmente usam variáveis do tipo numérico ou textual para receber os valores dos campos lidos de um banco de dados. Porém, essas variáveis não aceitam receber o valor nulo. Dessa forma, se o programador escreve uma linha de comando parecida com **`“`**`variável = valor_lido_do_banco`**`”,`** ele está sujeito a gerar um erro no programa caso o valor atributo no banco seja nulo.

Para resolver essa questão, os programadores precisam fazer inicialmente um teste de nulidade na variável, e só depois atribuir o valor a ela caso não seja nulo, ou atribuindo um espaço em branco ou um

zero, caso seja. Veja os exemplos abaixo de lógica de programação para você entender melhor como seria isso:

```
'Teste lógico para saber se o valor da variável no banco de dados é
nulo.
          Se Atributo1BancoDeDados é nulo
          Entao VariavelTexto = ""          'No caso de a variável ser
do tipo texto.
          Se não VariavelTexto = AtributoBancoDeDados

          Se Atributo2BancoDeDados é nulo
          Entao VariavelNumero = 0          'No caso de a variável ser
do tipo número.
          Se não VariavelNumero = AtributoBancoDeDados
```

Tenha atenção especial a atributos do tipo booleano, que nessa situação poderia apresentar três estados diferentes: verdadeiro, falso e nulo (nulo é diferente de falso). Daí a importância de modelar bem o projeto e evitar valores nulos, melhor é utilizar valores padrão como falso, 0 e em branco para atributos do tipo booleano, numérico (qualquer tipo) e textual (qualquer tipo).

10

1.2.3 - Interpretação (significado) de uma relação

O esquema de relação pode ser interpretado como uma declaração ou um tipo de afirmação. Por exemplo, o esquema da relação ALUNO visto no início deste módulo afirma que, em geral, uma entidade de aluno tem uma matrícula, um nome, se é efetivo ou ex-aluno, uma turma atual e uma série atual. Cada tupla na relação pode então ser interpretada como um **fato** ou uma **instância** em particular da afirmação. Por exemplo, a primeira tupla afirma que existe um aluno cuja matrícula é 14562/2, de nome Thiago Ferreira Borges, que é um aluno efetivo, que está na turma 5-1, cursando a 5ª série.

ALUNO				
Matrícula	**Nome**	**Efetivo**	**Turma Atual**	**Série Atual**
14562/2	Thiago Ferreira Borges	Sim	5-1	5

Observe que algumas relações podem representar fatos sobre **entidades**, enquanto outras podem representar fatos sobre **relacionamentos** entre entidades. Por exemplo, as relações ALUNO, DISCIPLINA e TURMA são entidades, pois existem por si só, porque representam objetos que existem naturalmente e isoladamente. Já o HISTORICO_ESCOLAR afirma que os alunos cursaram disciplinas escolares. Uma tupla nessa relação relaciona um aluno e disciplina cursada, dessa forma, essa relação é do tipo relacionamento, ela existe pela reunião de informações de entidades.

Logo, o modelo relacional representa **fatos** sobre entidades e relacionamentos uniformemente como relações. Isso às vezes compromete a compreensão, pois é preciso descobrir se uma relação representa

um tipo de entidade ou um tipo de relacionamento. Apresentaremos o modelo Entidade-Relacionamento (ER) com detalhes em um módulo mais à frente no nosso curso, no qual os conceitos de entidade e relacionamento serão descritos minuciosamente.

11

2 - Restrições em modelo relacional

Até aqui, discutimos as características de relações isoladas. No banco de dados relacional, normalmente haverá muitas relações, e as tuplas nessas relações costumam estar relacionadas de várias maneiras. O estado do banco de dados inteiro corresponderá aos estados de todas as suas relações em determinado ponto no tempo. Em geral, existem muitas **restrições** (ou constraints) sobre os valores reais em um estado do banco de dados. Essas restrições são derivadas das regras no minimundo que o banco de dados representa, conforme discutimos anteriormente.

Neste momento, discutiremos as diversas restrições sobre os dados que podem ser especificadas em um banco de dados relacional na forma de restrições. As restrições nos bancos de dados geralmente podem ser divididas em três categorias principais:

Restrições inerentes baseadas no modelo ou restrições implícitas	**Restrições baseadas em esquema** ou restrições explícitas.	**Restrições baseadas na aplicação** ou restrições semânticas ou regras de negócios
• Restrições que são inerentes no modelo de dados.	• Restrições que podem ser expressas diretamente nos esquemas do modelo de dados, em geral especificando-as na DDL (linguagem de definição de dados).	• Restrições que não podem ser expressas diretamente nos esquemas do modelo de dados, e, portanto, devem ser expressas e impostas pelos programas de aplicação.

12

As características das relações que discutimos no início deste módulo são as restrições inerentes ao modelo relacional e pertencem à **primeira categoria**. Por exemplo, a restrição de que uma relação não pode ter tuplas duplicadas é uma restrição inerente (não pode haver duas disciplinas com mesmo nome, por exemplo).

As restrições que discutimos nesta seção são da **segunda categoria**, a saber, restrições que podem ser expressas no esquema do modelo relacional por meio da DLL.

As restrições da **terceira categoria** são mais gerais, relacionam-se ao significado e também ao comportamento dos atributos, e são difíceis de expressar e impor dentro do modelo de dados, de modo que normalmente são verificadas nos sistemas de informação que realizam as atualizações no banco de dados. Como por exemplo: a data em que a nota de um aluno pode ser lançada no histórico escolar deve ser somente após o término do período de provas e antes do final do ano letivo.

Outra categoria importante de restrições é a de **dependências de dados**, que incluem dependências funcionais e dependências multivaloradas. Elas são usadas principalmente para testar a "qualidade" do projeto de um banco de dados relacional e em um processo chamado **normalização**, que será discutido em módulos futuros.

As **restrições baseadas em esquema** incluem:

- restrições de domínio,
- restrições de chave,
- restrições sobre nulidade,
- restrições de integridade de entidade e
- restrições de integridade referencial.

Veremos cada uma a seguir.

13

2.1 - Restrições de domínio

As restrições de domínio especificam que, dentro de cada tupla, o valor de cada atributo deve ser um valor indivisível do domínio ao qual ele pertence.

Os tipos de dados associados aos domínios normalmente incluem os tipos de dados numéricos padrão para inteiros (como short integer, integer e long integer) e números reais (float e double). Caracteres, booleanos, cadeia de caracteres de tamanho fixo (char) e cadeia de caracteres de tamanho variável (varchar) também estão disponíveis, assim como data, hora, marcador de tempo (timestamp), moeda ou outros tipos de dados especiais (binary, bit etc.). A especificação e as diferenças entre esses padrões serão discutidas futuramente, em outro módulo.

Outros domínios possíveis podem ser descritos por um subintervalo dos valores de um tipo de dados (exemplo: números inteiros de 1 a 100) ou como um tipo de dado enumerado (exemplo: Sim ou Não, Masculino ou Feminino etc.), em que todos os valores possíveis são explicitamente listados.

14

2.2 - Restrições de chave e restrições sobre valores nulos

No modelo relacional formal, uma relação é definida como um conjunto de tuplas. Por definição, todos os elementos de um conjunto são distintos; logo, todas as tuplas em uma relação também precisam ser distintas. Isso significa que duas tuplas não podem ter a mesma combinação de valores para todos os seus atributos. Normalmente, existem outros subconjuntos de atributos de um esquema de relação com a propriedade de que duas tuplas em qualquer estado de relação não deverão ter a mesma combinação de valores para esses atributos.

Levando em consideração de que cada conjunto de atributos que forma uma tupla é distrito dos demais conjuntos, a união de todos esses elementos é denominada de **superchave** do esquema da relação. Por meio destas informações é possível identificar exclusivamente uma única tupla dentre todos aqueles pertencentes à relação.

Vamos ver um exemplo: observe a relação ALUNO abaixo, identifique a tupla “432/2, Isadora Luccas Fernandes, Sim, 7-1, 7”, observe que ela é exclusiva, não existem duas tuplas com esses dados. Também é possível localizar qualquer outra tupla, caso saibamos seus atributos.

ALUNO				
Matrícula	**Nome**	**Efetivo**	**Turma Atual**	**Série Atual**
14562/2	Thiago Ferreira Borges	Sim	5-1	5
432/2	Isadora Luccas Fernandes	Sim	7-1	7
332/5	Marcelo Correia Luz	Não		
4539/1	Mariana Gonçalves Coelho	Sim	6-1	6

Entidade ALUNO com suas respectivas tuplas

Cada relação tem pelo menos uma **superchave padrão** — o conjunto de todos os seus atributos. Contudo, uma superchave pode ter atributos redundantes, de modo que um conceito mais útil é o de uma **chave**, que não tem redundância. Uma **chave** de uma relação é um atributo que é exclusivo. Um atributo que não há possibilidade de se repetir, e portanto, é capaz também de identificar, exclusivamente uma tupla em uma relação.

Por exemplo, examinando a tabela acima, localize a tupla cuja matrícula é igual a “332/5”. Observe que não existem dois alunos com a mesma matrícula. Desta forma, um único atributo (chave) é capaz de identificar a tupla da mesma forma que todos os seus atributos juntos (superchave) também o são.

15

Muitos bancos de dados utilizam identificadores de exclusividade que nos é comum, como número de CPF, número de cadastro/matrícula, sigla da Unidade da Federação, número da placa de um veículo, entre outros. Esses valores funcionam como chave, pois são capazes de identificar um único registro dentro de uma tabela.

Por outro lado, há tabelas que não possuem esse tipo de identificador, e para elas é necessário criar um **código identificador**. Nesses casos, normalmente cria-se um código numérico sequencial (1, 2, 3, ...), onde cada registro recebe um código que o identifica exclusivamente. Aos olhos dos usuários dos

sistemas, muitas vezes esses códigos passam despercebidos, mas para os programadores e administradores de dados, eles sabem que existem e o utilizam dentro dos programas para referência e localização.

Por exemplo, suponha que uma tabela de “unidades de medidas” possua dois campos: o **código** e o **nome da unidade de medida**. Suponha que a unidade de medida “quilo” possua código 3. Agora pense hipoteticamente que o usuário gostaria de alterar no nome da unidade “quilo” para “quilograma”. O comando SQL que faz essa operação é o seguinte:

```
UPDATE UNIDADE_MEDIDA SET NOME = “Quilograma” WHERE CODIGO = 3.
```

A tradução desta operação é “Altere a tabela UNIDADE_MEDIDA, atualize o valor de NOME para Quilograma do registro **cujo CODIGO é igual a 3**”. Observe que a parte final da instrução (em negrito) restringe e identifica exclusivamente um único registro. Não é para alterar todos os registros, apenas aquele cujo código é igual a três.

Em geral, um esquema de relação pode ter mais de uma chave. Nesse caso, cada uma das chaves é chamada de **chave candidata**. Por exemplo, se a relação ALUNO além da matrícula contivesse o CPF do aluno, esses dois campos seriam chaves candidatas, pois ambos são capazes de identificar exclusivamente um único aluno. É comum designar uma das chaves candidatas como a chave principal da relação. A essa chave principal dá-se o nome de **chave primária** da relação.

16

Quando um esquema de relação tem várias chaves candidatas, a escolha de uma para se tornar a chave primária é um tanto quanto arbitrária; porém, normalmente é melhor escolher uma chave primária com um único atributo ou um pequeno número de atributos.

As outras chaves candidatas são designadas como **chaves únicas** (unique keys). Ao tornar um atributo único, ele passa a não permitir valores duplicados. Por exemplo: se você disser que o atributo nome da entidade Aluno é único, então não será possível cadastrar dois alunos com mesmo nome.

Outra restrição sobre os atributos especifica se valores nulos são permitidos ou não. Por exemplo, se cada tupla de ALUNO precisar ter um valor válido, diferente de nulo, para o atributo Nome, então Nome de ALUNO é restrito a ser NOT NULL.

Independentemente de muitas relações possuírem chaves candidatas, é comum os projetistas de bancos de dados sempre criarem um campo de identificação sequencial em todas as tabelas que representem entidades (a exceção são as tabelas que representam associações, cuja chave é a composição das chaves das entidades que elas representam). O objetivo das chaves numéricas sequenciais é facilitar a programação e a manutenção dos dados pelos administradores de dados. Geralmente esse campo recebe o nome de **ID** ou **Código**.

17

3 - Bancos de dados relacionais e esquemas de banco de dados relacional

Um **esquema de banco de dados relacional** é um conjunto de relações e um conjunto de restrições de integridade.

Após a modelagem e implementação de um banco de dados, com suas respectivas restrições, só é possível inserir ou alterar informações que obedeçam às **regras de integridade**.

Exemplo: levando em consideração que poderia haver uma associação entre HISTORICO_ESCOLAR, ALUNO, TURMA e DISCIPLINA, só seria possível inserir um registro em histórico escolar caso os dados obedeçam às seguintes regras:

a) O aluno deve existir na tabela de alunos.
b) A turma e série devem existir na tabela de turmas.
c) A disciplina deve existir na tabela de disciplina.
d) A nota deve ser um número entre 0 e 10, com uma casa decimal.

Caso a informação a ser cadastrada seja contraditória a qualquer uma dessas regras, a sua inserção na tabela não será permitida pelo SGBD. Da mesma forma, se tentássemos altera um registro já cadastrado em histórico escolar, mudando a disciplina para uma que não exista na tabela de disciplinas, essa mudança também será impedida de acontecer pelo SGBD.

18

3.1 - Integridade, integridade referencial e chaves estrangeiras

Já aprendemos que para que uma tupla possa ser identificada em uma relação é necessário que seja exista um identificador para essa tupla. Esse identificador pode ser uma chave ou superchave. À chave escolhida com principal identificador damos o nome de **chave primária**.

A **restrição de integridade de entidade** afirma que nenhum valor de chave primária pode ser nulo.

Levando em consideração que o valor da chave primária é usado para identificar tuplas individuais em uma relação, possuir valores nulos para a chave primária implica que não podemos identificar essas tuplas. Por exemplo, se duas ou mais tuplas tivessem `NULL` para suas chaves primárias, não conseguiríamos distingui-las ao tentar referenciá-las por outras relações.

As restrições de chave e as restrições de integridade de entidade são especificadas sobre relações individuais.

A **restrição de integridade referencial** é especificada entre duas relações e usada para manter a consistência entre tuplas nas duas relações. Informalmente, a restrição de integridade referencial afirma que uma tupla em uma relação que referencia outra relação precisa se referir a uma tupla existente nessa relação.

19

Vamos ver o exemplo a seguir. Observe a relação CIDADE, que possui dois atributos: id_cidade (que representa a chave primária) e nome. A outra entidade, PONTO_TURISTICO, possui três atributos: id_ponto_turistico (que representa a chave primária), id_cidade (que informa em qual cidade está o ponto turístico), e nome.

CIDADE	
Id_cidade	**Nome**
1	Rio de Janeiro
2	Foz do Iguaçu
3	São Paulo
4	Rio Grande do Sul

Relação de Cidades

PONTO_TURISTICO		
Id_ponto_turistico	**Id_cidade**	**Nome**
1	1	Pão de Açúcar
2	1	Cristo Redentor
3	2	Cataratas do Iguaçu
4	4	Cânion Itaimbezinho

Relação de Pontos Turísticos

Observe como o identificador das cidades em cada ponto turístico serve para identificar cada cidade de localização. A chave primária de cada relação, quando utilizada para referência em outra relação recebe o nome de chave estrangeira. Portanto, para o exemplo acima, “Id_Cidade” em “PONTO_TURISTICO” é uma chave estrangeira da relação “CIDADE”.

20

A integridade referencial diz que em “Id_Cidade” da relação PONTO_TURISTICO só podem ser utilizados valores que existam previamente na relação CIDADE. Não é possível, por exemplo, inserir uma tupla em PONTO_TURISTICO com Id_cidade igual a 5. Caso uma tupla seja excluída da relação CIDADE, pode haver alguns **problemas** ou **controles** que mantenham a integridade do banco, por exemplo:

- Se tentarmos excluir a cidade de São Paulo de CIDADE, a exclusão ocorrerá sem problema algum, pois essa cidade não é utilizada em PONTO_TURISTICO.

- Se tentarmos excluir a cidade de Rio de Janeiro de CIDADE, temos duas opções que podem acontecer, de acordo de como o modelo foi projetado (veremos como fazer isso em outro momento):

    1. Se a opção “deletar em cascata” tiver sido configurada, todos os pontos turísticos onde aparecem a cidade do Rio de Janeiro serão excluídos da relação de PONTOS_TURISTICOS, e depois a cidade do Rio de Janeiro será excluída da relação de CIDADE.

    2. Se a opção “deletar em cascata” não tiver sido configurada, então a exclusão da cidade do Rio de Janeiro será impedida, pois o sistema não permitirá deixar pontos turísticos sem a referência da cidade onde eles estão localizados. O SGBD gerará um erro de exclusão.

3.2 - Uso de chaves estrangeiras para otimizar o modelo de dados

O uso da estratégia de chaves primárias e estrangeiras (para referência) beneficia dois aspectos do modelo de dados: controle de redundância de informações e otimização do uso do espaço em disco. Sem o uso dessa técnica, teríamos apenas uma relação de pontos turísticos dessa forma:

PONTO_TURISTICO		
Id_ponto_turistico	**Cidade**	**Nome**
1	Rio de Janeiro	Pão de Açúcar
2	Rio de Janeiro	Cristo Redentor
3	Foz do Iguaçu	Cataratas do Iguaçu
4	Cataratas do Iguaçu	Cânion Itaimbezinho

Relação de Pontos Turísticos não otimizada

Desta segunda forma, os dados repetidos, além de redundantes, podem gerar alguns problemas de qualidade da informação. Por exemplo, se alguém cadastrar um ponto turístico utilizando como cidade “R. de Janeiro” ou “RJ”, gerará uma informação no banco de dados onde uma pergunta do usuário do tipo “Quais são os pontos turísticos do Rio de Janeiro?” não será capaz de localizar “R. de Janeiro” e “RJ”, pois esses valores são diferentes.

Em um banco de dados de muitas relações, normalmente existem muitas restrições de integridade referencial. Para especificar essas restrições, primeiro devemos ter um conhecimento claro do significado ou papel que cada atributo, ou conjunto de atributos, que fazem parte nos diversos esquemas de relação do banco de dados.

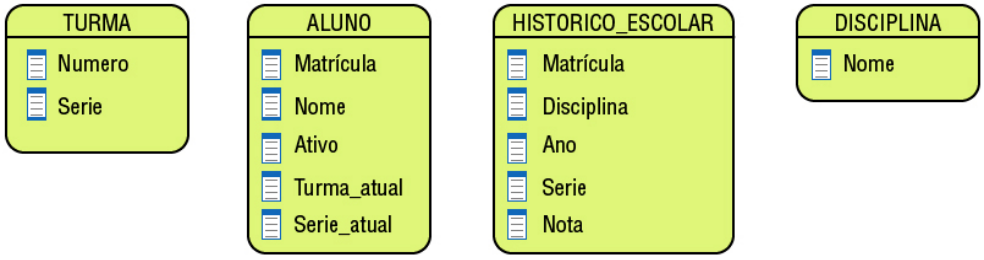
As restrições de integridade referencial surgem com frequência dos relacionamentos entre as entidades representadas pelos esquemas de relação, pois a maioria das entidades relaciona-se entre si, é raro um esquema onde entidades residem isoladamente.

22

3.3 - Criando referências para otimizar um esquema

Você se lembra daquele esquema exemplificativo da escola que criamos? Vamos trabalhar nele agora para criar chaves primárias e referências.

O modelo-base que temos para iniciar é o mesmo que já vimos anteriormente:



Modelo conceitual inicial

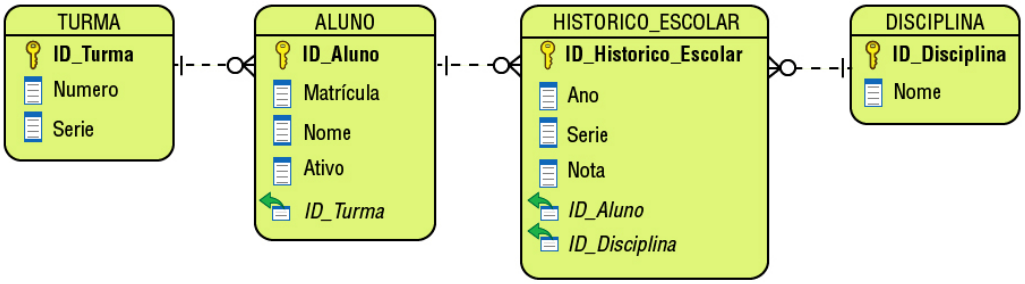
Nosso primeiro passo é criar chaves primárias para cada relação. Note que mesmo que a relação ALUNO tenha uma chave candidata boa, Matrícula, vamos inserir um identificador sequencial para padronizar todas as relações. Iremos, também, declarar que os identificadores criados serão chaves primárias.

Normalmente a simbologia que as ferramentas de modelagem utilizam é o símbolo de uma **chave**. Dessa forma, nosso novo modelo é o seguinte:

Modelo conceitual com inclusão de identificadores

23

O próximo passo é substituir o conteúdo (texto) nas relações referenciadas pelos identificadores das relações de referência. Para isso substituiremos turma_atual e serie_atual em ALUNO por ID_Turma, Matrícula em HISTORICO_ESCOLAR por ID_Aluno e Disciplina por ID_Disciplina. Finalmente, na ferramenta de modelagem, indicaremos que as tabelas se relacionam entre si pelos atributos criados. Note que símbolos como este "-|- - - -<" foram diagramados para dizer que existe um relacionamento entre as tabelas (essa parte explicaremos oportunamente). Outras ferramentas utilizam setas para indicar a relação. O resultado final é o que se segue:



Modelo conceitual com relações

Mais à frente estudaremos a parte de diagramação e de normalização. Por enquanto, compreenda apenas que criamos relacionamentos entre as relações. Note ainda que a ferramenta de diagramação criou um símbolo especial para dizer que aquele atributo é uma chave estrangeira. O símbolo utilizado é "  ".

Todas as restrições de integridade deverão ser especificadas no esquema de banco de dados relacional (ou seja, definidas como parte de sua definição) se quisermos impor essas restrições sobre os estados do banco de dados. Logo, a DDL inclui meios para especificar os diversos tipos de restrições de modo que o SGBD possa impô-las automaticamente. A maioria dos SGBDs relacionais admite restrições de chave, integridade de entidade e integridade referencial. Essas restrições são especificadas como uma parte da definição de dados na DDL.

24

Resumo

Neste módulo, aprendemos que:

a) O modelo de dados relacional é padrão mais difundido e utilizado globalmente. Suas principais vantagens são a simplicidade de uso, maior performance, uso de teorias simples como a teoria de conjuntos da matemática, localização eficiente de registros, uso otimizado do espaço em disco.

b) O modelo relacional representa o banco de dados como uma coleção de relações. Cada relação é um assunto ou tema a ser abordado.

c) Cada entidade (relação) representa uma coleção de valores de dados relacionados. Ou seja, atributos de uma tabela.

d) Uma relação representa uma entidade ou uma tabela em um modelo de dados.

e) Uma relação é composta por atributos. Cada atributo representa uma informação específica.

f) As relações tanto podem representar fatos sobre entidades quando relacionamentos entre elas.

g) Entidades são relações bases que existem por si só, são completas e geralmente seus dados são utilizados por outras relações.

h) Relacionamentos são relações que só existem em um contexto. Esse contexto geralmente é a representação de entidades que se associam entre si.

i) Cada cadastro em uma entidade, ou seja, o conteúdo de uma relação é denominado de tupla.

j) Um domínio representa o conjunto de valores válidos para um determinado atributo.

k) Um esquema relacional é uma forma textual de descrever uma relação e seus atributos, esse padrão é definido pelo nome da relação e, entre parêntesis, separados por vírgulas, os respectivos atributos.

l) Grau é o termo utilizado para definir o número de atributos de uma relação.

m) Em uma relação, não é possível representar uma ordenação lógica para as tuplas que ela contém.

n) Restrições são regras que restringem valores ou relacionamentos entre os elementos do banco de dados.

o) Restrição de domínio refere-se a quais valores são considerados válidos para um atributo.

p) Restrição de chave refere-se a um determinado atributo só aceitar valores presentes em um determinado campo de outra tabela, na maioria das vezes, chaves primárias de outras tabelas.

q) Restrição de nulidade refere-se a um determinado atributo aceitar ou não o valor nulo. Quando não aceita valores nulos é comumente chamado de campos obrigatórios.

r) Superchave é a denominação do conjunto de atributos que forma uma tupla distinta de todas as demais de uma relação.

s) Uma chave de uma relação é um atributo que é exclusivo.

t) Chave candidata é o nome que se dá a um atributo exclusivo que é capaz de identificar uma tupla exclusivamente.

u) Chave primária é o atributo escolhido pelo projetista para ser o identificador principal de uma tabela.

v) Quando um atributo é declarado como único (unique key), ele passa a não permitir valores duplicados.

w) Chaves primárias não podem ter seus valores nulos. A nulidade de uma chave primária fere o princípio da integridade de entidade, gera ambiguidades e impede relacionamentos.

x) A integridade referencial refere-se à relação entre duas relações, onde a chave primária de uma relação é utilizada como chave estrangeira (para referência) na outra relação.

y) O uso de chaves estrangeiras traz vários benefícios para o modelo de dados, como melhor eficiência, eliminação de redundâncias, uso menor de espaço em disco e garantia de consistência de dados.

UNIDADE 2 – MODELO DE DADOS RELACIONAL E SQL
MÓDULO 2 – INTRODUÇÃO A SQL

01

1 - OPERAÇÕES DE ATUALIZAÇÃO, TRANSAÇÕES E TRATAMENTO DE VIOLAÇÕES DE RESTRIÇÃO

Olá, seja bem-vindo a mais uma etapa do nosso estudo. Neste módulo, iremos começar a tratar sobre os comandos SQL.

As operações do modelo relacional podem ser categorizadas em **recuperações** e **modificações**.

Uma expressão da álgebra relacional de **recuperação** forma uma nova relação após a aplicação de uma série de operadores algébricos a um conjunto existente de relações; seu uso principal é **consultar um banco de dados** a fim de consultar informações.

O usuário formula uma consulta que especifica os dados de interesse, e uma nova relação é formada aplicando operadores relacionais para recuperar esses dados. Esta relação resultado torna-se a resposta para a (ou resultado da) consulta do usuário.

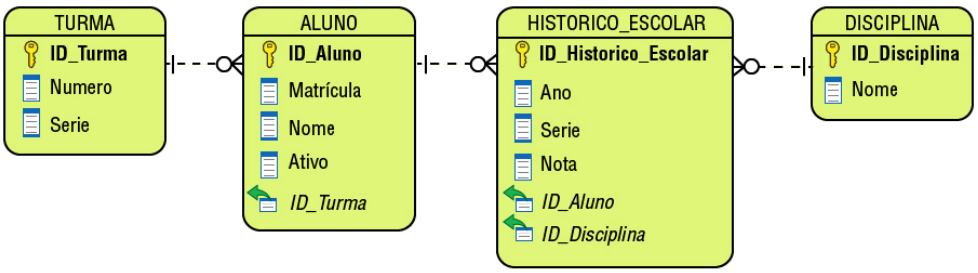
Em outras palavras, tenha em mente que um banco de dados é um minimundo onde todas as informações cadastradas estão lá. Cada usuário tem interesse por uma parte dessas informações (e nunca todo o conjunto de uma vez). Dessa forma, cada consulta que um usuário realiza no banco de dados é uma extração de um subconjunto desse conjunto maior. As informações que serão extraídas, o formato dessas informações, as ordenações e filtros desejados formam a técnica que é utilizada para extrair esse subconjunto.

02

Outro tipo de operação básica é representado pela **inserção, alteração e exclusão** de dados, operações estas que **modificam** o conteúdo do banco de dados. Elas inserem novos dados, excluem dados antigos ou modificam registros de dados existentes.

`INSERT` é o nome do comando SQL usado para inserir uma ou várias novas tuplas em uma relação;
`DELETE` é o comando SQL usado para excluir tuplas, e
`UPDATE` é o comando SQL usado para alterar os valores de alguns atributos nas tuplas existentes.

Sempre que essas operações são aplicadas, as restrições de integridade especificadas sobre o esquema de banco de dados relacional não devem ser violadas. Nesta seção, discutimos os tipos de restrições que podem ser violadas por cada uma dessas operações e os tipos de ações que podem ser tomados se uma operação causar uma violação. Usamos o banco de dados da escola para os exemplos e discutimos apenas as restrições de chave, restrições de integridade de entidade e as restrições de integridade referencial.



Modelo do banco de dados da escola que utilizaremos para exemplificar este módulo

TURMA		
ID_Turma	**Número**	**Série**
1	5-1	5ª
2	5-2	5ª
3	5-3	5ª
4	6-1	6ª
5	6-2	6ª
6	7-1	7ª

Registros atuais de turmas e séries

ALUNO				
ID_Aluno	**Matrícula**	**Nome**	**Ativo**	**ID_Turma**
1	14562/2	Thiago Ferreira Borges	Sim	1
2	432/2	Isadora Luccas Fernandes	Sim	6
3	332/5	Marcelo Correia Luz	Não	
4	4539/1	Mariana Gonçalves Coelho	Sim	4

Registros dos alunos e ex-alunos.

Obs.: Utilizaremos apenas essas duas relações para nossos exemplos.

03

1.1 - A operação Inserir

A operação **Inserir** oferece uma lista de valores de atributo para que uma nova tupla possa ser inserida em uma relação.

Ela pode violar qualquer um dos quatro tipos de restrições discutidos anteriormente (gerando erros ou alertas). As restrições de domínio podem ser violadas se for dado um valor de atributo que não aparece

no domínio correspondente ou não é do tipo de dado apropriado. As restrições de chave podem ser violadas se um valor de chave na nova tupla já existir em outra tupla na relação. A integridade de entidade pode ser violada se qualquer parte da chave primária da nova tupla for nula. A integridade referencial pode ser violada se o valor de qualquer chave estrangeira em se referir a uma tupla que não existe na relação referenciada. Aqui estão alguns exemplos para ilustrar essa discussão.

Operação	Resultado
1- Operação: INSERIR (NULL, ‘8-1’, ‘8ª.’) em TURMA.	Resultado: esta inserção viola a restrição de integridade de entidade (NULL para a chave primária ID_Turma), de modo que é rejeitada.
2- Operação: INSERIR (6, ‘8-1’, ‘8ª.’) em TURMA.	Resultado: Esta inserção viola a restrição de chave porque outra tupla com o mesmo valor de ID_Turma (6) já existe na relação TURMA, e, portanto, é rejeitada.
3- Operação: INSERIR (5, ‘2660/0’, ‘Marcos Valério Santiago’, ‘Sim’, 7) em ALUNO.	Resultado: Esta inserção viola a restrição de integridade referencial especificada sobre ID_Turma em ALUNO porque não existe uma tupla referenciada correspondente em TURMA com ID_Turma igual a 7.
4- Operação: INSERIR (5, ‘2660/0’, ‘Marcos Valério Santiago’, ‘Sim’, 6) em ALUNO.	Resultado: Esta inserção satisfaz todas as restrições, de modo que é aceitável.

Se uma inserção violar uma ou mais restrições, a opção padrão é **rejeitar a inserção**. Nesse caso, o SGBD geralmente oferece um motivo ao usuário sobre a rejeição da inserção, ou seja, uma informação do porquê ocorreu o erro. Outra opção é tentar corrigir o motivo da rejeição da inserção, mas isso normalmente não é usado para violações causadas pela operação Inserir; em vez disso, é usado com mais frequência na correção de violações das operações Excluir e Atualizar. Saiba+

Saiba+
Na primeira operação, o SGBD poderia pedir ao usuário para oferecer um valor para ID_Turma ou mesmo identificar o próximo ID_Turma da sequência (que no caso seria o número 7), e poderia então aceitar a inserção. Na operação 3, o SGBD poderia pedir que o usuário mudasse o valor de ID_Turma para algum valor válido (ou defini-lo como NULL), ou poderia pedir ao usuário para inserir uma tupla em TURMA com o ID_Turma igual a 7 e poderia aceitar a inserção original somente depois que uma operação fosse aceita.

04

1.2 - A operação Excluir

A operação **Excluir** pode violar apenas a integridade referencial. Isso ocorre se a tupla que está sendo excluída for referenciada por chaves estrangeiras de outras tuplas no banco de dados.

Para especificar a exclusão, uma condição sobre os atributos da relação seleciona a tupla (ou tuplas) a ser(em) excluída(s). Aqui estão alguns exemplos.

Operação	Resultado
1- Operação: Excluir a tupla em TURMA com ID_Turma = 2.	Resultado: Esta exclusão é aceitável e exclui exatamente uma tupla.
2- Operação: Excluir a tupla em TURMA com ID_Turma = 1.	Resultado: Esta exclusão não é aceitável, pois existem tuplas em ALUNO que se referenciam a esta tupla. Logo, se a tupla em TURMA for excluída, haverá violações de integridade referencial.

Várias opções estão disponíveis se uma operação de exclusão causar uma violação. A primeira opção, chamada **restrict**, é rejeitar a exclusão. A segunda opção, chamada **cascade**, é tentar propagar (ou gerar em cascata) a exclusão excluindo tuplas que referenciam aquela que está sendo excluída. Por exemplo, nesta operação, o SGBD poderia excluir automaticamente a primeira tupla da relação ALUNO (que é referenciada pelo comando) juntamente com a tupla da relação TURMA. Uma terceira opção, chamada **set null** ou **set default**, é modificar os valores de atributo que referenciam a causa da violação; cada valor desse tipo é definido para `NULL` ou alterado para referenciar um valor padrão válido. Observe que, se um atributo referenciando que causa uma violação faz parte da chave primária, ele não pode ser definido como nulo; caso contrário, ele violaria a integridade de entidade.

Em geral, quando uma restrição de integridade referencial é especificada na DDL, o SGBD permitirá que o projetista de banco de dados especifique qual das três opções se aplica no caso de uma violação da restrição. Cada tabela pode ter um tipo de configuração de restrição distinta.

05

1.3 - A operação Atualizar

A operação **Atualizar** é usada para alterar os valores de um ou mais atributos em uma tupla (ou tuplas) de alguma relação.

É necessário especificar uma condição sobre os atributos da relação para selecionar a tupla ou tuplas a serem modificadas. Aqui estão alguns exemplos.

Operação	Resultado
1. Operação: Atualizar ALUNO, alterando ID_Turma para 2 onde o ID_Aluno = 1.	Resultado: Aceitável, irá mudar a turma do anulo ‘Thiago’ de ‘5-1’ para ‘5-2’ da ‘5ª série’.
2. Operação: Atualizar ALUNO, alterando o ID_Turma para NULL e Ativo para ‘Não’ onde o ID_Aluno = 4.	Resultado: Aceitável, irá transformar a aluna ‘Mariana’ em ex-aluna.
3. Operação: Atualizar ALUNO, alterando	Resultado: Inaceitável, pois não existe aluno com esse ID.

ID_Turma para 2 onde o ID_Aluno = 5.	
4. Operação: Atualizar ALUNO, alterando ID_Turma para 7 onde o ID_Aluno = 4.	Resultado: Inaceitável, pois viola a integridade referencial, não existe turma com ID_Turma igual a 7.
5. Operação: Atualizar ALUNO, alterando ID_Aluno para 4 onde o ID_Aluno = 3.	Resultado: Inaceitável, pois viola a restrição de chave primária, repetindo um valor que já existe como chave primária na tupla.
6. Operação: Atualizar ALUNO, alterando ID_Turma para '7-1' onde o ID_Aluno = 3.	Resultado: Inaceitável, pois o valor '7-1' é inválido para o tipo de dado, além de ferir a regra de integridade referencial.

Atualizar um atributo que nem faz parte de uma **chave primária** nem de uma **chave estrangeira** em geral não causa problemas; o SGBD só precisa verificar para confirmar se o novo valor é do tipo de dado e domínio corretos. **Modificar um valor de chave primária é semelhante a excluir uma tupla e inserir outra em seu lugar**, pois usamos a chave primária para identificar tuplas. Logo, as questões discutidas anteriormente nos itens "Operação Inserir" e "Operação Excluir" entram em cena. Saiba+

Saiba+
Se um atributo de chave estrangeira for modificado, o SGBD deverá garantir que o novo valor referencia uma tupla existente na relação referenciada (ou que seja definido como NULL). Existem opções semelhantes para lidar com as violações de integridade referencial causadas pela operação Atualizar, como as opções discutidas para a operação Excluir. De fato, quando uma restrição de integridade referencial for especificada na DLL, o SGBD permitirá que o usuário escolha opções separadas para lidar com uma violação causada pela operação Excluir e uma violação causada pela operação Atualizar.

06

2 - A LINGUAGEM SQL

A linguagem SQL pode ser considerada um dos principais motivos para o sucesso dos bancos de dados relacionais comerciais. Como ela se tornou um padrão para esse tipo de bancos de dados, os usuários ficaram menos preocupados com a migração de suas aplicações de outros tipos de sistemas de banco de dados — por exemplo, sistemas de rede e hierárquicos — para sistemas relacionais. Isso aconteceu porque mesmo que os usuários estivessem insatisfeitos com o produto de um SGBD relacional em particular que estavam usando, a conversão para outro produto de SGBD relacional não seria tão cara ou demorada, pois os dois sistemas seguiam os mesmos padrões de linguagem.

Na prática, é óbvio, existem muitas diferenças entre diversos fabricantes de SGBD relacionais comerciais. Porém, se o usuário for cuidadoso em usar apenas dos recursos que fazem parte do padrão, e se os dois sistemas relacionais admitirem fielmente o padrão, então a conversão entre ambos deverá ser bastante simplificada. Outra vantagem de ter esse padrão é que os usuários podem escrever

comandos em um programa de aplicação de banco de dados que pode acessar dados armazenados em dois ou mais SGBDs relacionais sem ter de mudar a linguagem de banco de dados (SQL) se os sistemas admitirem o padrão SQL.

Apresentaremos agora os principais **recursos do padrão SQL** para SGBDs relacionais comerciais, enquanto o item anterior apresentou os conceitos mais importantes por trás do modelo de dados relacional formal. Futuramente, discutiremos as operações da álgebra relacional, que são muito importantes para entender os tipos de solicitações que podem ser especificadas em um banco de dados relacional.

A linguagem SQL oferece uma interface de linguagem declarativa de nível mais alto, de modo que o usuário apenas especifica qual deve ser o resultado, deixando a otimização real e as decisões sobre como executar a consulta para o SGBD. Embora a SQL inclua alguns recursos da álgebra relacional, ela é baseada em grande parte no **cálculo relacional de tupla**, que estudaremos em momentos futuros. Porém, a sintaxe SQL é mais fácil de ser utilizada do que qualquer uma das duas linguagens formais.

Álgebra relacional
Elas também são importantes para processamento e otimização de consulta em um SGBD relacional. No entanto, as operações da álgebra relacional são consideradas muito técnicas para a maioria dos usuários de SGBD comercial, pois uma consulta em álgebra relacional é escrita como uma sequência de operações que, quando executadas, produz o resultado exigido. Logo, o usuário precisa especificar como — ou seja, em que ordem — executar as operações de consulta.

07

2.1 - Um pouco sobre a história da SQL

O nome SQL vem do termo em inglês Structured Query Language (**Linguagem de Consulta Estruturada**). Originalmente, SQL era chamada de SEQUEL (Structured English QUEry Language) e foi criada e implementada na IBM Research como a interface para um sistema de banco de dados relacional experimental, chamado SYSTEM R.

A SQL agora é a linguagem padrão para SGBDs relacionais comerciais. Tentaremos abordar a última versão da SQL ao máximo possível.

A SQL é uma linguagem de banco de dados abrangente: tem instruções para definição de dados, consultas e atualizações. Logo, ela é uma DDL (Data Definition Language - trabalha com objetos, por exemplo, tabelas) e uma DML (Data Manipulation Language - trabalha com linhas). Além disso, ela tem facilidades para definir visões sobre o banco de dados, para especificar segurança e autorização, para definir restrições de integridade e para especificar controles de transação. Ela também possui regras para embutir instruções SQL em uma linguagem de programação de uso geral, como Java, COBOL ou C.

Os padrões SQL mais recentes (começando com SQL:1999) são divididos em uma especificação núcleo mais extensões especializadas. O núcleo deve ser implementado por todos os fornecedores de SGBDR que sejam compatíveis com SQL. As extensões podem ser implementadas como módulos opcionais a serem adquiridos independentemente para aplicações de banco de dados específicas, como mineração de dados, dados espaciais, dados temporais, data warehousing, processamento analítico on-line (OLAP), dados de multimídia e assim por diante.

Como a SQL é muito importante (e muito grande), dedicamos alguns módulos para explicar seus recursos.

Padrão
Um esforço conjunto entre o American National Standards Institute (ANSI) e a International Standards Organization (ISO) levou a uma versão-padrão da SQL (ANSI, 1986), chamada SQL.86 ou SQL1. Um padrão revisado e bastante expandido, denominado SQL-92 (também conhecido como SQL2) foi desenvolvido mais tarde. O próximo padrão reconhecido foi SQL:1999, que começou como SQL3. Duas atualizações posteriores ao padrão são SQL:2003 e SQL.2006, que acrescentaram recursos de XML entre outras atualizações para a linguagem. Outra atualização em 2008 incorporou mais recursos de banco de dados de objeto na SQL.

08

2.2 - Conceito de transação

Um programa de aplicação de banco de dados que funciona com um banco de dados relacional normalmente executa uma ou mais transações.

Uma **transação** é um programa em execução que inclui **duas ou mais operações sequenciais e relacionadas entre si de modificação de dados**, como aplicar inserções, exclusões ou atualizações a ele.

Uma única transação pode envolver qualquer número de **operações de recuperação** e qualquer número de **operações de atualização** (sendo que a lógica é que existam duas ou mais). Essas recuperações e atualizações juntas formarão uma unidade atômica de trabalho no banco de dados. Por exemplo, uma transação para aplicar uma transferência bancária costuma ler o registro da conta do correntista de origem, verificar se existe saldo suficiente, debitar o valor da sua conta e depois creditar o valor na conta do destinatário.

Um grande número de aplicações comerciais, que utilizam bancos de dados relacionais em sistemas de processamento de transação on-line (OLTP — Online Transaction Processing), executa transações que atingem taxas velocidade de centenas ou milhares de operações por segundo.

Os conceitos de processamento de transação, execução concorrente de transações e recuperação de falhas serão discutidos em outro momento.

09

Para se **iniciar uma transação**, utiliza-se o comando **`BEGIN TRANSACTION`** ou `BEGIN TRANS` no início do programa que contém os comandos da transação.

Ao final da transação, ela precisa deixar o banco de dados em um estado válido ou coerente, que satisfaça todas as restrições especificadas no esquema do banco de dados. O comando que confirma o encerramento da transação é denominado **`COMMIT TRANSATION`** ou `COMMIT TRANS` ou apenas `COMMIT`.

Caso uma das operações da transação falhe, todas as modificações realizadas serão desfeitas, voltando aos dados anteriores à transação. Essa operação é denominada `ROLL BACK`. Geralmente os programas possuem um controle de erros, caso a operação ocorra com sucesso é aplicado o `COMMIT`, caso ocorra erro é aplicado o `ROLL BACK`.

A lógica de programação que organiza esses comandos é exemplificada a seguir:

```
BEGIN TRANS
      Operação 1;
      Operação 2;
      Operação 3;
SE houver erro ENTÃO
      ROLL BACK;
SENÃO COMMIT;
```

10

3 - Definições e tipos de dados em SQL

A SQL usa os termos **tabela**, **linha** e **coluna (ou campo)** para os termos do modelo relacional formal **relação**, **tupla** e **atributo**, respectivamente. Usaremos os termos correspondentes para indicar a mesma coisa.

O principal comando SQL para a definição de dados é o CREATE, que pode ser usado para criar esquemas (espaços para bancos de dados), tabelas (relações) e domínios (valores válidos para um atributo), bem como outras construções como views, assertions e triggers.

Como a especificação SQL é muito grande, oferecemos uma descrição dos recursos mais importantes. Outros detalhes poderão ser encontrados em diversos documentos dos padrões SQL (sempre consulte no Google para saber mais).

11

3.1 - Conceito de Esquema / Banco de Dados

As primeiras versões da SQL não incluíam o conceito de um esquema de banco de dados relacional; todas as tabelas (relações) eram consideradas parte do mesmo esquema. O conceito de um esquema SQL foi incorporado inicialmente com SQL2 a fim de agrupar tabelas e outras construções que pertencem à mesma aplicação de banco de dados.

Um **esquema** SQL é identificado por um **nome de esquema**, e inclui um **identificador de autorização** para indicar o usuário ou conta proprietária do esquema, bem como descritores para cada elemento.

Esses elementos incluem tabelas, restrições, visões, domínios e outras construções (como concessões — grants — de autorização) que descrevem o esquema que é criado por meio da instrução CREATE SCHEMA. Esta pode incluir todas as definições dos elementos do esquema. Como alternativa, o esquema pode receber um identificador de nome e autorização, e os elementos podem ser definidos mais tarde. Por exemplo, a instrução a seguir cria um esquema chamado ESCOLA, pertencente ao usuário com identificador de autorização 'JoseSilva'. Observe que cada instrução em SQL termina com um ponto e vírgula.

```
CREATE SCHEMA ESCOLA AUTHORIZATION ‘JoseSilva’;
```

Observação: O uso de aspas simples ou duplas também depende do SGBD utilizado, alguns aceitam ambas opções para definir textos e datas. Números e palavras-chave não usam aspas.

12

Muitos bancos de dados utilizam o termo **DATABASE** (banco de dados) como sinônimo de esquema. Nesses casos, criar um DATABASE é a mesma funcionalidade de se criar um SCHEMA. A única diferença clara entre os dois é que o banco de dados não possui um usuário padrão controlado pela cláusula AUTHORIZATION (que só existe em esquemas).

Dessa forma, o comando SQL que cria um novo banco de dados é:

```
CREATE DATABASE ESCOLA;
```

Em geral, os usuários **não** estão autorizados a criar esquemas, bancos de dados, tabelas e demais elementos do banco de dados. O privilégio para criar esses elementos deve ser concedido explicitamente às contas de usuário relevantes pelo administrador do sistema, administrador de dados (AD) ou ao administrador de banco de dados (DBA).

Numa organização é comum que seja criado um esquema (ou um banco de dados) para cada aplicação ou cada grande assunto, função ou departamento, de forma a criar um espaço para todas as tabelas que de alguma forma são inter-relacionadas. Por exemplo, é comum vermos em organizações esquemas ou banco de dados com nomes que remetem à "Vendas", "DepartamentoPessoal", "Recursos Humanos", "Estoque", "SistemaX", "SistemaY" etc.

13

3.2 - Conceito de Catálogo de Banco de Dados

Além do conceito de um esquema, a SQL usa o conceito de um **catálogo**.

Catálogo é uma coleção nomeada de esquemas em um ambiente SQL.

Um ambiente SQL é basicamente uma instalação de um SGBDR compatível com SQL em um sistema de computador. Um catálogo sempre contém um esquema especial, chamado **INFORMATION SCHEMA**, que oferece informações sobre todos os esquemas no catálogo e todos os seus descritores de elemento.

As restrições de integridade, como a integridade referencial, podem ser definidas entre as relações somente se existirem nos esquemas dentro do mesmo catálogo. Os esquemas dentro do mesmo catálogo também podem compartilhar certos elementos, como definições de domínio.

14

3.3 - O comando CREATE TABLE em SQL

O comando **CREATE TABLE** é usado para especificar uma nova relação (ou seja, uma tabela), dando-lhe um nome e especificando seus atributos e restrições iniciais. Os atributos são especificados primeiro, e cada um deles recebe um nome, um tipo de dado para especificar seu domínio de valores e quaisquer restrições de atributo, como NOT NULL, para indicar que o campo é obrigatório.

As **restrições de chave**, **integridade de entidade** e **integridade referencial** podem ser especificadas na instrução CREATE TABLE, depois que os atributos forem declarados, ou acrescentadas depois, usando o comando ALTER TABLE (veremos mais à frente, juntamente com o comando DROP TABLE que exclui uma tabela do SGBD). Em geral, o esquema SQL em que as relações são declaradas é especificado implicitamente no ambiente em que as instruções CREATE TABLE são executadas. Para o nosso exemplo da escola, o comando que cria a tabela de turma é CREATE TABLE TURMA. Como alternativa, podemos conectar explicitamente o nome do esquema (ou do banco de dados) ao nome da relação, separados por um ponto. Por exemplo, escrevendo CREATE TABLE ESCOLA.TURMA.

As relações declaradas por meio das instruções CREATE TABLE são chamadas de tabelas da base (ou relações da base); isso significa que a relação e suas tuplas são realmente criadas e armazenadas como um arquivo pelo SGBD. As relações da base são distintas das relações virtuais, criadas por meio da instrução CREATE VIEW, que podem ou não corresponder a um arquivo físico real. Em SQL, os atributos em uma tabela da base são considerados ordenados na sequência em que são especificados no comando CREATE TABLE. No entanto, as linhas (tuplas) não são consideradas ordenadas dentro de uma relação.

O comando CREATE TABLE permite criar a tabela, definir seus campos, tipificar esses campos, criar chaves primárias, índices e realizar relacionamentos, tudo ao mesmo tempo e em um único comando! Entretanto, ao criar um novo modelo com várias tabelas, é importante observar que, caso você não observe atentamente a ordem de criação das tabelas, podem ocorrer erros de relacionamento, pois você pode estar criando um relacionamento cuja tabela de origem ainda não existe. Veja um exemplo.

Exemplo
Vamos a um exemplo prático: Se no modelo da escola você criar primeiramente a tabela ALUNO e disser que o campo ID_Turma está relacionando com o ID_Turma da tabela TURMA, ocorrerá um erro, pois a tabela TURMA ainda não foi criada.

15

Dessa forma, há duas possibilidades para a criação de um novo modelo:

a) Criar primeiro as tabelas que não recebem chaves estrangeiras e depois criar as tabelas que recebem as chaves estrangeiras.	**b) Criar todas as tabelas sem criar relacionamentos, e depois alterar a estrutura das tabelas adicionando os relacionamentos** (usando o comando ALTER TABLE).
• Dessa forma, uma possível ordem de criação de tabelas para nosso exemplo seria a seguinte sequência: TURMA → DISCIPLINA → ALUNO →HISTORICO_ESCOLAR.	• Dessa forma, não é necessário obedecer nenhuma ordem de criação.

Utilizando a opção “a” acima, o comando SQL que cria todo o nosso modelo de escola é apresentado abaixo e discutido cada parte do comando nas próximas seções.

```
CREATE TABLE 'TURMA' (
  'ID_Turma' int NOT NULL,
  'Numero' varchar(5) NOT NULL UNIQUE,
  'Serie' varchar(3) NOT NULL,
  PRIMARY KEY ('ID_Turma'));

CREATE TABLE 'DISCIPLINA' (
  'ID_Disciplina' int NOT NULL,
```

```
  'Nome' varchar(50) NOT NULL,
  PRIMARY KEY ('ID_Disciplina'));

CREATE TABLE 'ALUNO' (
  'ID_Aluno' int NOT NULL,
  'Matricula' varchar(10) NOT NULL UNIQUE,
  'Nome' varchar(250) NOT NULL,
  'Ativo' boolean NOT NULL,
  'ID_Turma' int NULL,
  PRIMARY KEY ('ID_Aluno')),
  CONSTRAINT 'turma_fk' FOREIGN KEY ('ID_Turma') REFERENCES 'TURMA'
('ID_Turma');

CREATE TABLE 'HISTORICO_ESCOLAR' (
  'ID_Historico_Escolar' int NOT NULL,
  'Ano' int NOT NULL,
  'Serie' varchar(3) NOT NULL,
  'Nota' decimal (2,1) NOT NULL,
  'ID_Aluno' int NOT NULL,
  'ID_Disciplina' int NOT NULL,
  PRIMARY KEY ('ID_Historico_Escolar')),
  CONSTRAINT 'aluno_fk' FOREIGN KEY ('ID_aluno') REFERENCES 'ALUNO'
('ID_Aluno'),
  CONSTRAINT 'disciplina_fk' FOREIGN KEY ('ID_Disciplina') REFERENCES
'DISCIPLINA' ('ID_Disciplina');
```

16

3.4 - Tipos de dados de atributo e domínios em SQL

Os tipos de dados básicos disponíveis para atributos são:

- booleano,
- numérico,
- cadeia ou sequência de caracteres,
- cadeia ou sequência de bits,
- data e hora.

Vejamos cada tipo de dado a seguir.

3.4.1 - Tipo de dado booleano

O tipo de dado booleano é o tipo de dado mais simples que existe. Ele é formado por apenas um bit que pode ser sinalizado como 0, significando FALSO ou como 1, significando VERDADEIRO.

Também pode receber o valor NULL, cuja interpretação é um terceiro estado denominado valor desconhecido.

17

3.4.2 - Tipo de dados numérico

Os tipos de dados **numérico** incluem números inteiros de vários tamanhos (INTEGER ou INT, BIG INT, SMALLINT e TINYINT) e números de ponto flutuante (reais) de várias precisões (FLOAT ou REAL e DOUBLE PRECISION). O formato dos números pode ser declarado usando DECIMAL (i, j) — ou NUMERIC (i, j) — onde i, a precisão, é o número total de dígitos decimais e j, a escala, é o número de dígitos após o ponto decimal. O valor padrão para a escala é zero, e para a precisão, é definido pela implementação. Vejamos alguns detalhes técnicos:

Padrões para números inteiros:

BIGINT	Representa um número inteiro que vai de -2^63 (-9.223.372.036.854.775.808) a 2^63-1 (9.223.372.036.854.775.807), ocupa 4 bytes de armazenamento.
INT	Representa um número inteiro que vai de -2^31 (-2.147.483.648) a 2^31-1 (2.147.483.647), ocupa 4 bytes de armazenamento.
SMALLINT	Representa um número inteiro que vai de 2^15 (-32,768) a 2^15-1 (32,767), ocupa 2 bytes de armazenamento.
TINYINT	Representa um número inteiro que vai de 0 a 255, ocupa 1 byte de armazenamento.

Veja que na prática, para representar a idade de uma pessoa, basta declararmos o valor como TINYINT, pois ninguém possuirá mais de 255 anos de idade. Essas opções de escolha permitem que o modelador do banco de dados escolha aquela que é mais viável para o banco de dados. Por outro lado, a economia de espaço gerada no banco de dados é mínima ao se utilizar a opção TINYINT ao invés da INT, que é a padrão, por isso, na grande maioria das aplicações, não é necessário que o projetista fique preocupado em fazer a melhor escolha, a opção padrão satisfaz a quase todos os casos.

18

Padrões para números de ponto flutuante:

Float	Representa números no gigantesco intervalo que vai de - 1,79E+308 a -2,23E-308, 0 e 2,23E-308 a 1,79E+308, ou seja, permite armazenar qualquer número necessário em qualquer tipo de sistema.
Real	Representa números no intervalo - 3,40E + 38 a -1,18E - 38, 0 e 1,18E - 38 a 3,40E + 38.
Decimal (i,j) e **Numeric** (i,j)	Representam números que dependem de como são configurados. Normalmente utilizamos esses tipificadores para definir atributos relacionados a moeda ou quando precisamos definir o número exato de casas decimais, como no exemplo da nota do aluno que vai de 0 a 10 com uma casa decimal. A diferença entre decimal e numeric é que em numeric os valores dos decimais sempre são preenchidos com zeros à direita. Exemplo, o número 123,45 para um campo NUMERO1 decimal (10,5) é representado por 123,45; já para o campo NUMERO2 numeric (10,5) é representado por 123,45000.

19

3.4.3 - Tipo de dados de cadeia de caracteres

Os tipos de dados de cadeia de caracteres são de tamanho fixo ou variável. Utilizamos respectivamente os tipos CHAR(n), e VARCHAR(n), onde n é o número de caracteres.

Ao atribuir um valor literal para uma cadeia de caracteres, deve-se colocar o valor entre aspas simples (apóstrofos), e é **case sensitive** (diferencia maiúsculas de minúsculas). Dessa forma, os valores para nome ‘Marcelo’ e ‘marcelo’ são diferentes entre si.

Para cadeias de caracteres de tamanho fixo, uma cadeia mais curta é preenchida com caracteres em branco **à direita**. Por exemplo, se o valor 'Silva' for atribuído a um atributo do tipo CHAR(10), ele é preenchido com cinco caracteres em branco, torando-se **‘Silva ’**.

Esses espaços em branco podem diferenciar os valores em alguns SGBDs, onde eles consideram ‘Silva’ diferente de ‘Silva ’. Muitos SGBDs permitem configurar se o SGBD deve considerar os valores diferentes ou não.

Para fins de comparação, as cadeias de caracteres são consideradas ordenadas em ordem alfabética; se uma cadeia de caracteres aparece antes de outra cadeia em ordem alfabética, então a primeira é considerada menor que segunda. Dessa forma, “Antonio” é considerado menor que “João” que é menor que “Pedro”. Essa comparação matemática é usada para mecanismos de ordenação de valores.

Outro tipo de dado de cadeia de caracteres de tamanho variável, chamado TEXT ou CHARACTER LARGE OBJECT ou CLOB, que é utilizamos para cadeias de caracteres imensas, como por exemplo, todo o texto de um livro.

20

3.4.4 - Tipos de dados de cadeia de bits

O tipo de dado bit é o tipo mais simples que existe, podendo armazenar apenas os números 0 ou 1. Ele é equivalente ao tipo de dado booleano (V ou F).

Há um outro tipo de dado denominado BLOB (do inglês Binary Large Object), que pode conter um tamanho imenso de informações no formato binário. Esse tipo de dado é comumente utilizado para armazenar arquivos em banco de dados, como por exemplo, fotos, vídeos, documentos digitalizados e outros.

3.4.5 - Tipo de dado para datas

O tipo de dados **DATE** possui dez posições, e seus componentes são **DAY** (dia), **MONTH** (mês) e **YEAR** (ano) na forma DD-MM-YYYY. O tipo de dado TIME (tempo) tem pelo menos oito posições, com os componentes HOUR (hora), MINUTE (minuto) e SECOND (segundo) na forma HH:MM:SS. Somente datas e horas válidas devem ser permitidas pela implementação SQL. Isso implica que os meses devem estar entre 1 e 12 e os dias devem estar entre 1 e 31; além disso, uma data deve ser uma data válida para o mês correspondente.

As comparações com < (menor que) e > (maior que) podem ser usadas com datas ou horas — uma data anterior é considerada menor que uma data posterior, e da mesma forma com o tempo. Os valores literais são representados por cadeias com apóstrofos; por exemplo, '27-09-2015' ou '09:12:47'.

Um valor de data pode incluir o horário também, como por exemplo '27-09-2015 09:12:47'. Quando não especificamos o horário, é assumido 00:00:00. Portanto '27-09-2015 00:00:00' é idêntico a '27-09-2015'.

21

3.4.6 - Criando um tipo de dado novo por meio de um domínio

Um domínio, como já vimos, pode ser criado para definir um tipo especial de dado. Por exemplo, ao criar um domínio denominado TIPO_CPF e dizendo que esse domínio é equivalente a CHAR(11), estamos dizendo que qualquer agora há um novo tipo de dado disponível que será de 11 posições de caracteres.

Um exemplo do comando que cria um domínio é:

```
CREATE DOMAIN TIPO_CPF AS CHAR (11).
```

A partir deste domínio criado, é possível agora criar atributos baseado nesse domínio. Por exemplo, na construção de uma tabela hipotética de funcionários, poderíamos criar um atributo denominado CPF_FUNCIONÁRIO dessa forma:

```
CREATE TABLE 'FUNCIONARIOS' (
     'ID_Funcionario' int NOT NULL,
      'CPF_Funcionario' TIPO_CPF NOT NULL, ...
```

Observe que nem todos os SGBDs implementam a funcionalidade de criação de domínios.

22

3.5 - Definindo um valor inicial

Para que um atributo nunca receba um valor nulo, além de definirmos ele como **NOT NULL**, podemos especificar um valor padrão. Isso é feito por meio da DEFAULT, seguido do valor que queremos informar.

É comum em projetos de bancos de dados definirmos valores como zero para números, falso para booleano e branco (‘’) para caracteres. Dessa forma, para especificarmos que o atributo ‘ativo’ da tabela ALUNO seja por padrão igual a FALSO, acrescentaríamos a cláusula DEFAULT da seguinte forma:

```
'Ativo' boolean NOT NULL DEFAULT FALSE,
```

Dessa forma, sempre que uma tupla fosse cadastrada e o valor de ativo não fosse especificado, automaticamente seria atribuído FALSO para esse campo. Ou seja, uma operação do tipo INSERT que não contivesse o valor do atributo ‘ativo’ seria automaticamente interpretado como FALSE e este valor inserido no banco de dados.

23

3.6 - Especificando restrições de chave primária

Já aprendemos que uma chave primária de uma tabela deve ser criada a fim de que possamos identificar exclusivamente um único registro daquela tabela. Isso é feito na implementação do banco de dados por meio da cláusula **PRIMARY KEY**, que normalmente é definida durante a criação da tabela.

Como você pode observar no trecho do nosso modelo de exemplo, após a definição dos atributos da tabela incluímos a especificação PRIMARY_KEY e, entre parêntesis, o(s) campo(s) que define(m) a chave primária daquela tabela.

Veja o trecho abaixo a definição da chave primária destacada em amarelo:

```
CREATE TABLE 'TURMA' (
  'ID_Turma' int NOT NULL,
  'Numero' varchar(5) NOT NULL,
  'Serie' varchar(3) NOT NULL,
  PRIMARY KEY ('ID_Turma'));
```

Há tabelas que são resultado da associação de duas ou mais tabelas. Por exemplo, em um sistema de venda de produtos, uma tabela ITENS_DE_VENDA representa o que o cliente está comprando e é formada pela chave primária da tabela da VENDA com a chave primária de um PRODUTO, além de valores de quantidade, custo unitário e desconto (hipoteticamente).

Nesse caso, a chave primária dessa tabela é a composição das chaves primárias da tabela VENDA e da tabela PRODUTO, criando algo que poderia ser parecido com:

```
PRIMARY KEY ('ID_Venda', 'ID_Produto').
```

24

3.7 - Especificando valores únicos

Em algumas implementações de banco de dados, é importante que alguns valores não possam ser repetidos, o que acontece no caso de identificadores como CPF, Matrícula, Código cadastral, e outros. Para tal restrição, utilizamos a cláusula **UNIQUE** para definir que aquele campo não aceita valores duplicados.

Observe no exemplo do modelo da escola que os campos Numero em TURMA e Matricula em ALUNO não devem aceitar valores duplicados e por isso possuem a cláusula UNIQUE conforme o trecho abaixo, destacado em amarelo:

```
CREATE TABLE 'ALUNO' (
  'ID_Aluno' int NOT NULL,
  'Matricula' varchar(10) NOT NULL UNIQUE,
  'Nome' varchar(250) NOT NULL,...
```

25

3.8 - Definindo um relacionamento de chave estrangeira

Uma chave estrangeira é outro tipo de restrição, denominada **integridade referencial**.

Para definirmos uma regra de relacionamento, precisamos utilizar uma cláusula que possui três parâmetros:

- o nome da relação,
- o campo na tabela que se refere a uma chave estrangeira, e
- localização da chave primária e da tabela de onde a chave estrangeira se refere.

O padrão para esta cláusula é `CONSTRAINT 'NomeQualquerParaARelação' FOREIGN KEY ('CampoNaTabelaQueÉAChaveEstrangeira') REFERENCES 'NomeDaTabelaReferenciada' ('ChavePrimáriaDaTabelaReferenciada');`

Observe no trecho abaixo, destacado em amarelo, como foi especificada a relação da tabela ALUNO com a tabela TURMA, onde a chave primária da tabela TURMA está relacionada com uma chave estrangeira na tabela ALUNO.

```
CREATE TABLE 'ALUNO' (
  'ID_Aluno' int NOT NULL,
  'Matricula' varchar(10) NOT NULL UNIQUE,
  'Nome' varchar(250) NOT NULL,
  'Ativo' boolean NOT NULL,
  'ID_Turma' int NULL,
  PRIMARY KEY ('ID_Turma')),
  CONSTRAINT 'turma_fk' FOREIGN KEY ('ID_Turma') REFERENCES 'TURMA'
('ID_Turma');
```

26

RESUMO

Neste módulo, aprendemos que:

a) As operações que realizamos em modelo de dados são classificadas em recuperações e modificações. As recuperações são feitas mediante o operador SELECT, as modificações são aplicadas por meio dos operadores INSERT, UPDATE e DELETE, que executam, respectivamente, as operações de inserir tuplas, atualizar campos e excluir tuplas.

b) As operações de modificação podem afetar um ou vários registros ao mesmo tempo.

c) Ao executar uma operação de modificação, o SGBD sempre verifica se alguma violação de restrição impede ou afeta o comando de ser executado, gerando alertas e erros nesses casos.

d) Há três configurações possíveis para controlar uma regra de integridade referencial em uma operação de excluir:

   a. Restrict, que é rejeitar uma operação inválida.

   b. Cascade, que significa propagar a exclusão em todas as tabelas afetadas (referenciadas) pela operação.

   c. Set null, que é tornar os valores referenciados como nulos.

e) Uma transação representa um controle de duas ou mais operações de modificação de dados, onde caso uma das operações falhe, todas as demais serão desfeitas. Os três comandos que controlam uma transação são: Begin Trans (inicia uma transação), Commit (confirma a transação) e Roll Back (cancela uma transação).

f) A SQL é uma linguagem própria para operar bancos de dados. Permite a definição, consulta e manipulação dos dados.

g) Um esquema de banco de dados é uma área designada para conter várias tabelas inter-relacionadas. Alguns SGBDs implementam a denominação banco de dados (database) para o mesmo propósito.

h) Um catálogo é um banco de dados especial que controla informações utilizadas pelo próprio SGBD para seu gerenciamento. O information squema é o nome deste banco de dados.

i) O comando CREATE TABLE cria uma nova tabela no banco de dados.

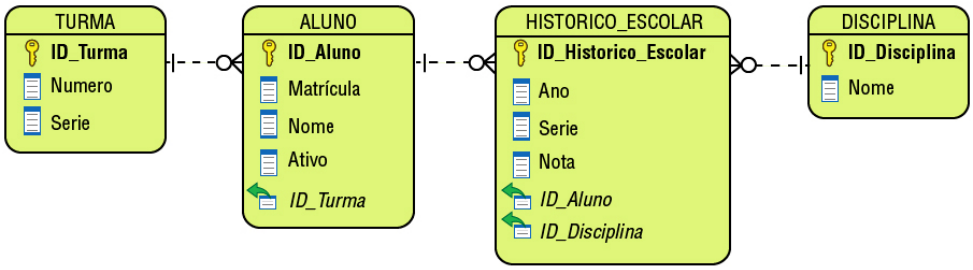
j) Há vários tipos de informação possíveis de serem armazenadas em um banco de dados, em cada tipo há uma série de tipificadores de atributos apropriados. Texto, número, data, booleano e binário são os tipos mais comuns.

UNIDADE 2 – MODELO DE DADOS RELACIONAL E SQL
MÓDULO 3 – INSTRUÇÃO SELECT (PARTE 1)
01

1 - CONSULTAS DE RECUPERAÇÃO BÁSICA EM SQL

Olá, seja bem-vindo a mais uma etapa do nosso estudo. Neste módulo, continuaremos a tratar sobre os comandos SQL, e iniciaremos o estudo sobre o mais complexo (e poderoso) dos comandos SQL: a instrução **SELECT**.

A SQL tem uma instrução básica para recuperar informações de um banco de dados denominada SELECT. A instrução SELECT é o mesmo que a operação SELECT da álgebra relacional. Existem muitas opções e tipos de instrução SELECT em SQL, de modo que introduziremos seus recursos gradualmente (usaremos três módulos para isso). Na maioria dos exemplos que apresentarmos, elas serão baseadas no esquema da escola que temos falado desde o início do curso.



Modelo do banco de dados da escola, que utilizaremos na maioria dos exemplos deste módulo

Esse banco de dados está populado com as seguintes informações hipotéticas para nossos exemplos:

ID_Turma	Numero	Serie
1	5-1	5a
2	5-2	5a
3	5-3	5a
4	6-1	6a
5	6-2	6a
6	7-1	7a

Registros atuais de turmas e séries

ID_Aluno	Matricula	Nome	Ativo	ID_Turma
1	14562/2	Thiago Ferreira Borges	1	1
2	432/2	Isadora Luccas Fernandes	1	6
3	332/5	Marcelo Correia Luz	0	0
4	4539/1	Mariana Gonçalves Coelho	1	4

Registros dos alunos e ex-alunos.

02

Os registros que estão armazenados na tabela de histórico escolar representam os seguintes valores:

HISTÓRICO ESCOLAR				
Matrícula	**Disciplina**	**Ano**	**Série**	**Nota**
14562/2	Matemática	2015	5ª	6,8
14562/2	Português	2015	5ª	8,9

14562/2	Ciências	2015	5ª	10,0
432/2	Matemática	2015	7ª	8,2
432/2	Português	2015	7ª	8,9
432/2	Geografia	2015	7ª	9,4
432/2	História	2015	7ª	9,0
332/5	Matemática	2010	5ª	8,8
332/5	Português	2010	5ª	8,9
332/5	Ciências	2010	5ª	6,4

Nosso modelo de dados incorpora identificadores (chaves estrangeiras) que armazenam os dados desta maneira:

ID_Historico_Escolar	Ano	Serie	Nota	ID_Aluno	ID_Disciplina
1	2015	5a	6.0	1	1
2	2015	5a	8.9	1	2
3	2015	5a	10.0	1	3
4	2015	7a	8.2	2	1
5	2015	7a	8.9	2	2
6	2015	7a	9.4	2	4
7	2015	7a	9.0	2	5
8	2015	5a	8.8	3	1
9	2015	5a	8.9	3	2
10	2015	5a	6.4	3	3

Registros do histórico escolar de todos os alunos

03

Neste capítulo, apresentamos os recursos da SQL para consultas de recuperação simples. Os recursos da SQL para especificar consultas de recuperação mais complexas serão apresentados oportunamente.

Antes de prosseguir, devemos apontar uma distinção importante entre a SQL e o modelo relacional formal.

A SQL permite que uma tabela (relação) tenha duas ou mais tuplas que são idênticas em todos os seus valores de atributo. Assim, em geral, **uma tabela SQL não é um conjunto de tuplas**, pois um conjunto não permite dois membros idênticos.

Algumas relações SQL são restritas a serem conjuntos porque uma restrição de chave foi declarada ou porque a opção DISTINCT foi usada com a instrução SELECT (descrita mais adiante neste módulo). Precisamos estar cientes dessa distinção à medida que discutirmos os exemplos.

Este banco de dados será o que utilizaremos neste e nos próximos módulos, portanto você verá esta parte introdutória repetida nos próximos módulos.

04

2 - A ESTRUTURA SELECT-FROM-WHERE DAS CONSULTAS SQL BÁSICAS

As consultas em SQL podem ser muito complexas. Começaremos com consultas simples, e depois passaremos progressivamente para as mais complexas. A forma básica do comando SELECT é composta pelas três cláusulas SELECT, FROM e WHERE, e tem a seguinte forma:

```
SELECT «lista de atributos»
FROM «lista de tabelas»
WHERE «condição»
```

Onde:

- **«lista atributos»** é uma lista de nomes de atributo cujos valores devem ser recuperados pela consulta. Representa os campos que formarão as colunas da tabela-resposta da consulta.
- **«lista tabelas»** é uma lista dos nomes de relação exigidos para processar a consulta. Representa as tabelas ondes os atributos estão localizados.
- **«condição»** é uma expressão condicional que identifica as tuplas a serem recuperadas pela consulta. Funciona como um "filtro" para não mostrar todos os registros, mas somente aqueles desejados. Também é usado para expressar a relação entre duas tabelas (geralmente associando uma chave primária a uma chave estrangeira). Este parâmetro é opcional.

Em SQL, os operadores básicos de comparação lógicos para comparar valores de atributo entre si e com constantes literais são =, <, <=, >, >= e <>. Estes correspondem aos mesmos operadores da álgebra relacional e da maioria das linguagens de programação. A principal diferença sintática é o operador diferente, nas linguagens de programação geralmente utiliza-se "**!=**" para simbolizar "diferente de". A SQL possui operadores de comparação adicional que apresentaremos de maneira gradual.

05

2.1 - Formato básico de uma consulta SQL

Ilustraremos a instrução SELECT básica em SQL com alguns exemplos de consultas. As consultas são rotuladas aqui números como C1, C2, C3 e assim por diante, de forma a podermos criar referências. Vamos ao primeiro exemplo:

C1 - *Recuperar o nome e a matrícula de todos os alunos.*

```
SELECT Nome, Matricula FROM ALUNO
```

Esta consulta envolve apenas a relação ALUNO listada na cláusula FROM. A consulta seleciona as tuplas individuais de ALUNO que satisfazem a condição da cláusula WHERE, depois projeta o resultado nos atributos Nome e Matricula listados na cláusula SELECT. A cláusula SELECT da SQL especifica os atributos cujos valores devem ser recuperados, que são chamados atributos de projeção.

Observe que neste exemplo não aparece a cláusula WHERE, desta forma, como todos os alunos do banco de dados satisfazem à consulta, todos eles serão apresentados como resposta. Entretanto, a tabela ALUNO possui cinco campos, e apenas dois foram especificados na consulta. Dessa forma, apenas os campos Nome e Matrícula serão apresentados como resposta pelo SGBD. Note também que a sequência dos campos (nome e matrícula) não são os mesmos da tabela, o SGBD se encarrega de representar na sequência desejada. O resultado esperado é:

Nome	Matricula
Thiago Ferreira Borges	14562/2
Isadora Luccas Fernandes	432/2
Marcelo Correia Luz	332/5
Mariana Gonçalves Coelho	4539/1

Resultado da consulta C1.

06

C2 - *Vamos agora fazer uma pequena variação, vamos listar apenas alunos cuja matrícula seja igual a "332/5":*

```
SELECT Nome, Matricula FROM ALUNO WHERE Matricula = '332/5'
```

Perceba agora a presença da cláusula WHERE, que especifica uma condição booleana que deve ser verdadeira para qualquer tupla recuperada, também conhecida como condição de seleção. Para realizar tal consulta, o SGBD irá percorrer todas as tuplas do banco de dados e apresentar apenas aquelas que satisfazem a condição imposta pela cláusula WHERE, eliminando aquelas que não satisfazem.

No nosso exemplo, só há um único aluno que satisfaz essa condição, dessa forma, o SGBD irá responder a esta consulta assim:

Nome	Matricula
Marcelo Correia Luz	332/5

Resultado da consulta C2

Os valores cadastrados como número de matrícula, a exemplo '322/5' e os demais, por conter o símbolo de barra ("/"), não são considerados um tipo numérico, mas sim texto. Para que seja possível armazenar o símbolo de barra, o SGBD precisa interpretar e armazenar esse tipo de valor como char ou varchar, por exemplo. Portanto, a cláusula WHERE requerer que o valor esteja representado entre aspas simples. Se no exemplo acima você não usasse as aspas, o SGBD interpretaria 322/5 como a divisão do número 322 pelo número 5, ou seja, o número: 64,4. O mesmo valeria para o símbolo de hífen, por exemplo, 322-5 sem aspas para o SGBD é a subtração de dois números, que tem como resultado 317.

07

2.2 - Procurando por valores nulos

Na instrução SELECT, usamos o operador WHERE x = y para filtrar e localizar itens. Entretanto, não podemos escrever algo como "campo = NULL" para localizar valores nulos, pois não é possível comparar valores nulos. O que a SQL oferece é o formato "campo IS NULL". Veja o exemplo abaixo:

C3 *- Localizar os alunos que não estão lotados em uma turma.*

```
SELECT NOME FROM ALUNOS WHERE ID_Turma IS NULL
```

O resultado dessa consulta seria apresentar o nome do aluno "Marcelo Correia Luz", que é o único que não tem uma turma associada a ele.

08

2.3 - Relacionando duas ou mais tabelas

Em SQL, o mesmo nome pode ser usado para dois (ou mais) atributos, desde que estes estejam em relações diferentes. Se isso acontecer, e uma consulta em múltiplas tabelas se referir a dois ou mais atributos com o mesmo nome, é preciso qualificar o nome do atributo com o nome da relação, para evitar ambiguidade. Isso é feito prefixando o nome da relação ao nome do atributo e separando os dois por um ponto, ou seja, no formato `NOME_DA_TABELA.NOME_DO_CAMPO`.

Por exemplo, note que as tabelas MATRÍCULA e ALUNO possuem o campo ID_Turma, caso uma operação SQL envolvesse ambos os campos, seria necessário identifica-los por meio do nome da tabela

dessa forma: `MATRICULA.ID_Turma e ALUNO.ID_Aluno`. Assim, o SGBD saberia interpretar de qual ID_Turma você está falando ao construir a sua instrução SQL.

Sempre que houver campos com mesmo nome, precisaremos informar o nome da tabela. Vamos ver isso na prática.

Nosso próximo exemplo envolve duas tabelas, iremos apresentar a matrícula dos alunos, os nomes, o número da turma e a série. Para tal consulta, precisaremos especificar as tabelas e os campos a serem apresentados bem como a cláusula de relação de interdependência entre essas tabelas (determinada pelas chaves primárias e estrangeiras da relação).

Há duas formas de fazer isso, utilizando a cláusula WHERE ou a cláusula JOIN (veremos o uso da cláusula JOIN no próximo módulo).

09

Nosso exemplo seria representado da seguinte forma:

C4 - *Recuperar a matrícula, o nome, a turma e a série de todos os alunos.*

```
SELECT Matricula, Nome, Numero, Serie
FROM ALUNO, TURMA
WHERE ALUNO.ID_TURMA = TURMA.ID_TURMA
```

Neste exemplo, o uso da condição ALUNO.ID_Turma = TURMA.ID_Turma faz com que as tabelas ALUNO e TURMA sejam inter-relacionadas e o resultado comum a elas apresentado. Neste caso, a cláusula WHERE não teve um efeito “filtragem e seleção” de registros, mas sim de relacionar duas tabelas. O SGBD irá processar essa consulta identificando o ID_TURMA de cada aluno e depois percorrendo a tabela TURMA para localizar o número e a série dos alunos. O resultado esperado para essa consulta é:

Matricula	Nome	Numero	Serie
14562/2	Thiago Ferreira Borges	5-1	5a
432/2	Isadora Luccas Fernandes	7-1	7a
4539/1	Mariana Gonçalves Coelho	6-1	6a

Resultado da consulta C4

Perceba um detalhe muito importante da resposta desse processamento: o aluno “Marcelo Correia Luz” por não possuir nenhuma turma, foi excluído da resposta da consulta. Isso acontece porque não há uma expressão verdadeira para a relação dele com as tuplas presentes em turma, a relação dele com turma é nula. Dessa forma, a relação dele é caracterizada como FALSA e, portanto, excluída da resposta da relação. Para resolver essa questão será necessário utilizar a cláusula LEFT JOIN que veremos futuramente.

2.4 - Nomes de atributos ambíguos, apelido e renomeação de atributos

Quando uma consulta utiliza campos de mesmo nome, podemos utilizar um “apelido” (alias) para renomear as colunas. Por exemplo, suponha que uma consulta relacione as tabelas HISTORICO_ESCOLAR e DISCIPLINA, onde os campos Nome de ambas as tabelas sejam apresentadas como resultado. Para não haver confusão, podemos renomear a informação consultada como, por exemplo, para Aluno e Disciplina ou “Nome do aluno” e “Nome da Disciplina”.

Para tal funcionalidade, usamos o operador AS <novo nome> logo após o nome do campo na instrução de SELECT. Dessa forma a instrução: SELECT ALUNO.Nome AS ‘Nome do Aluno’, DISCIPLINA.Nome AS ‘Nome da Disciplina’ FROM ... substitui os nomes originais dos campos pelos novos nomes.

Quando o novo nome é composto de apenas uma palavra, como NomeDoAluno ou Aluno, onde não há espaços em branco, não é necessário utilizar aspas para delimitar o novo nome, já quando você deseja utilizar mais de uma palavra, e consequentemente haverá espaços em branco entre elas, obrigatoriamente será necessário colocar o novo termo entre aspas simples. Observe que isso foi feito no exemplo acima. Dessa forma, observe os seguintes exemplos abaixo:

- `SELECT ALUNO.Nome AS Aluno` ➔ é válido, pois ‘aluno’ é uma palavra sem espaços em branco.
- `SELECT ALUNO.Nome AS NomeDoAluno` ➔ é válido, pois ‘NomeDoAluno’ não contém espaços.
- `SELECT ALUNO.Nome AS Nome_do_aluno` ➔é válido, pois ‘Nome_do_Aluno’ não contém espaços, o sublinhado substitui o espaço em branco com um caractere válido.
- `SELECT ALUNO.Nome AS Nome do aluno` ➔é inválido, pois ‘Nome do Aluno’ contém espaços em branco.
- `SELECT ALUNO.Nome AS ‘Nome do aluno’` ➔é válido, pois foi utilizado aspas para delimitar o atributo.
- `SELECT ALUNO.Nome AS ‘Aluno’` ➔ é válido, pois mesmo não precisando, foi utilizado aspas para delimitar o atributo.
- `SELECT ‘ALUNO’.‘Nome’ AS ‘Aluno’` ➔ é válido, pois mesmo não precisando, foi utilizado aspas para delimitar o atributo e o nome do campo original.
- `SELECT ‘ALUNO.Nome’ AS ‘Aluno’` ➔ é inválido, pois cada componente deve ser individualmente delimitado por aspas, não podemos juntar o nome da tabela e o campo entre aspas.

Observação: é possível nomear uma tabela como “TABELA DE ALUNOS” ou mesmo um campo como “Nome do aluno”, ou seja, utilizando espaços em branco. Isso não é uma boa prática, mas é possível. Ao fazer isso, as instruções SQL sempre precisarão utilizar aspas simples para identificar essas tabelas e campos. Exemplo: `SELECT ‘Nome do aluno’ FROM ‘Tabela de alunos’`. Evite ao máximo utilizar espaços em branco nos nomes dos componentes de um banco de dados. Também jamais utilize caracteres acentuados ou cedilha para nomear tabelas, campos, índices etc.

11

2.5 - Relacionando tabelas e aplicando filtros de pesquisa

Uma consulta que envolve apenas condições de seleção e junção mais atributos de projeção é conhecida como uma **consulta seleção – projeção - junção**. O próximo exemplo é uma consulta seleção – projeção - junção com duas condições de junção e duas condições de filtro. Para aplicar mais de uma condição utilizamos o operador AND que representa a junção de várias condições. Vamos ao exemplo:

C5 - *Recuperar o nome do aluno, o histórico escolar para o ano de 2015 e o nome das disciplinas, do aluno de matrícula "432/2". Observe que precisaremos relacionar as tabelas ALUNO, HISTORICO_ESCOLAR e DISCIPLINA, e aplicar dois filtros, um para o ano de 2015 e outro para a matrícula "432/2".*

```
SELECT ALUNO.Nome AS Aluno, DISCIPLINA.Nome AS DISCIPLINA,
HISTORICO_ESCOLAR.Ano, HISTORICO_ESCOLAR.Serie, HISTORICO_ESCOLAR.Nota
FROM ALUNO, HISTORICO_ESCOLAR, DISCIPLINA
WHERE ALUNO.ID_Aluno = HISTORICO_ESCOLAR.ID_Aluno
   AND DISCIPLINA.ID_Disciplina = HISTORICO_ESCOLAR.ID_Disciplina
   AND Ano =2015
   AND Matricula = '432/2'
```

Aluno	DISCIPLINA	Ano	Serie	Nota
Isadora Luccas Fernandes	Matemática	2015	7a	8.2
Isadora Luccas Fernandes	Português	2015	7a	8.9
Isadora Luccas Fernandes	Geografia	2015	7a	9.4
Isadora Luccas Fernandes	História	2015	7a	9.0

Resultado da consulta C5

12

O SGBD permite que você separe a instrução SQL em várias linhas, para ele, a quebra de linha não faz diferença, e para nós, a quebra de linha nos ajuda a organizar melhor as consultas mais complexas. Observe:

```
SELECT ALUNO.Nome AS Aluno, DISCIPLINA.Nome AS DISCIPLINA,
HISTORICO_ESCOLAR.Ano, HISTORICO_ESCOLAR.Serie, HISTORICO_ESCOLAR.Nota
FROM ALUNO, HISTORICO_ESCOLAR, DISCIPLINA
WHERE ALUNO.ID_Aluno = HISTORICO_ESCOLAR.ID_Aluno
   AND DISCIPLINA.ID_Disciplina = HISTORICO_ESCOLAR.ID_Disciplina
   AND Ano =2015
   AND Matricula = '432/2'
```

- Na consulta acima, a primeira linha contempla o tipo de operação (SELECT) e o nome dos atributos a serem apresentados. Observe o uso de apelidos para nome do aluno e nome da disciplina para diferenciá-los.
- A terceira linha, indicada pela cláusula FROM, informa de quais tabelas o SGBD precisa consultar as informações solicitadas.
- A quarta linha, com a cláusula WHERE, inicia a descrever o relacionamento entre as tabelas, informando que a chave primária ID_Aluno da tabela ALUNO corresponde à chave estrangeira ID_Aluno da tabela HISTORICO_ESCOLAR.
- A quinta linha, por meio do operador AND, continua a adicionar parâmetros de relacionamento, no caso, relacionando as tabelas DISCIPLINA e HISTORICO_ESCOLAR.
- A sexta linha representa a adição de um parâmetro de filtragem de seleção: não se quer todos os registros de histórico escolar, mas só aqueles do ano de 2015, que é um dos parâmetros de filtragem desejado.
- A sétima e última linha agrega o último parâmetro de filtragem: não se quer o histórico de todos os alunos, mas somente daquele aluno cuja matrícula é '432/2'.

Da mesma forma que o exemplo anterior, precisamos informar quais tabelas os atributos "Nome" se referem, visto que tanto em ALUNO quanto em DISCIPLINA aparece o atributo Nome, mas com significados e valores totalmente diferentes. O resultado dessa consulta é uma combinação das três tabelas.

Cada tupla no resultado será uma combinação de um aluno, um histórico escolar e uma disciplina, que satisfaz as condições de junção e filtragem. Os atributos de projeção são usados para escolher os atributos a serem exibidos com base em cada tupla combinada.

13

2.6 - Renomeando tabelas

O mesmo conceito do uso do operador AS, que renomeia atributos, pode ser utilizado para renomear tabelas. Isso é especialmente útil para quando nossas consultas SQL possuem muitas tabelas com valores ambíguos, pois podemos economizar em digitação ao renomear tabelas. Para renomear uma tabela, usamos o mesmo padrão já visto: AS <novo nome da tabela>.

Por exemplo, reveja a consulta C5, nós utilizamos os identificadores de aluno, histórico escolar e disciplina várias vezes no texto. Poderíamos simplificar esta consulta reescrevendo-a da seguinte forma:

```
SELECT A.Nome AS Aluno, D.Nome AS DISCIPLINA, Ano, Serie, Nota
FROM ALUNO AS A, HISTORICO_ESCOLAR AS H, DISCIPLINA AS D
```

```
WHERE A.ID_Aluno = H.ID_Aluno
   AND D.ID_Disciplina = H.ID_Disciplina
   AND Ano = 2015
   AND Matricula = '432/2'
```

Note que ao renomear as tabelas de aluno, histórico escolar e disciplina para, respectivamente, A, H e D, todas as vezes que precisamos referenciar às estas tabelas, não precisamos mais utilizar seu nome completo, mas sim utilizaremos apenas seu "apelido", que corresponde às letras indicadas. Perceba como fica esteticamente mais simples e fácil de analisar a instrução SQL utilizando novos nomes para as tabelas.

14

2.7 - Uso do asterisco para representar todos os campos de uma tabela

Há algumas situações em que precisamos criar uma consulta rápida em todos os atributos de uma tabela. Agora, imagine que você esteja trabalhando com uma tabela com 30 campos. Seria terrivelmente moroso ter de digitar todos esses 30 campos.

Para tal questão, a SQL provê uma forma simples para você informar que deseja ver todos os campos da tabela. Isso é feito substituindo o nome de todos os campos por um único asterisco. Vamos ver um exemplo.

C6 - *Listar todos os campos da tabela HISTORICO_ESCOLAR.*

A forma tradicional que sabemos até agora seria escrever essa consulta SQL da seguinte forma:

```
SELECT ID_Historico_Escolar, Ano, Serie, Nota, ID_Aluno, ID_Disciplina
FROM HISTORICO_ESCOLAR
```

Com o uso do asterisco, a nossa consulta se resumiria para o seguinte:

```
SELECT * FROM HISTORICO_ESCOLAR
```

Muito mais simples, correto? Observe que ambas as consultas produzem resultados idênticos.

O uso do asterisco é especialmente útil para descobrirmos quais são os campos de uma tabela. Quando não temos acesso ao modelo de dados ou simplesmente queremos recordar quais campos formam aquela tabela, basta digitar `SELECT * FROM <Nome_da_Tabela>`. O resultado produzido pelo SGBD conterá o nome dos campos na primeira linha.

Há ainda outra técnica muito simples para o SGBD retornar apenas os nomes dos campos, sem incluir registro algum. Saiba qual é.

Saiba qual é
Outra técnica é especialmente útil para quando queremos apenas identificar os campos, sem nos importar o conteúdo da tabela. Para isso, basta escrever uma cláusula que seja FALSA para todos os registros do banco, assim, o SGBD retornará um conjunto vazio de dados, e apenas o nome dos campos. Uma forma de se fazer isso é usar a cláusula `WHERE 1=2`. Como um é diferente de dois, o SGBD não irá mostrar registro algum, somente os nomes dos campos. Infelizmente, nem todo SGBD executa essa instrução dessa maneira, alguns apenas informam que o resultado é um conjunto vazio de dados, sem mostrar o nome dos campos.

15

Resumo

Neste módulo, aprendemos que:

a) A instrução SELECT da SQL é a instrução mais complexa e poderosa de todas, admite uma série de operadores e funções a fim de obter resultados nos mais diversos formatos exigidos pelos usuários dos bancos de dados.

b) A instrução SELECT básica possui três cláusulas: SELECT, FROM e WHERE, com a seguinte forma:

   a. SELECT «lista de atributos»

   b. FROM «lista de tabelas»

   c. WHERE «condição»

c) Os operadores básicos de comparação em SQL são =, <, <=, >, >= e <>.

d) A SQL permite organizar os campos da consulta em qualquer ordem que desejarmos.

e) Apenas valores que representam dado booleano e números podem ser representados sem aspas nas consultas. Todos os demais (textos, datas, binários, etc) devem vir entre aspas.

f) O operador IS NULL é utilizado para localizar valores nulos.

g) Uma das formas se relacionar duas tabelas em uma consulta SQL é utilizar o campo WHERE para representar as relações entre as tabelas. Normalmente a relação é feita comparando a chave primária de uma tabela com a chave estrangeira de outra.

h) Usa-se a referência no padrão NOME_DA_TABELA.NOME_DO_CAMPO quando uma relação pode apresentar ambiguidade de nomes, ou seja, quando ambas tabelas possuem o mesmo nome de campo.

i) Apelidos (alias) são criados para alterar os nomes das tabelas e campos de forma a representar a expressão SELECT de forma esteticamente mais limpa.

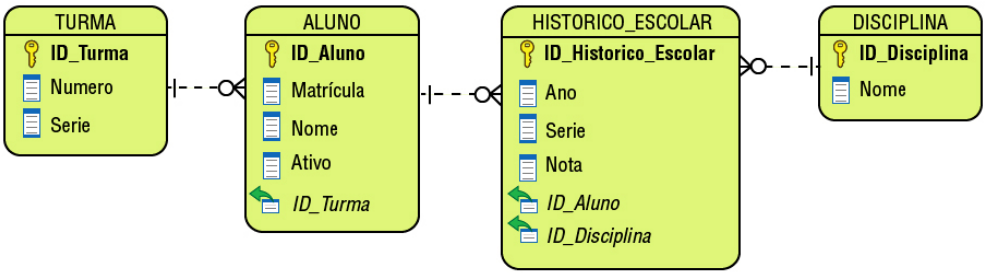
j) O uso do asterisco na cláusula SELECT faz com que todos os campos da tabela sejam apresentados.

UNIDADE 2 – MODELO DE DADOS RELACIONAL E SQL
MÓDULO 4 – INSTRUÇÃO SELECT (PARTE 2)
01

1 - MODELO E DADOS HIPOTÉTICOS USADOS NA MAIORIA DOS EXEMPLOS

Olá, seja bem-vindo a mais uma etapa do nosso estudo. Neste módulo, continuaremos a tratar sobre os comandos SQL.

Conforme explicamos na unidade anterior, usaremos a estrutura e o conteúdo abaixo na maioria dos nossos exemplos.



Modelo do banco de dados da escola que utilizaremos para exemplificar este módulo.

ID_Turma	Numero	Serie
1	5-1	5a
2	5-2	5a
3	5-3	5a
4	6-1	6a
5	6-2	6a
6	7-1	7a

Registros atuais de turmas e séries

ID_Aluno	Matricula	Nome	Ativo	ID_Turma
1	14562/2	Thiago Ferreira Borges	1	1
2	432/2	Isadora Luccas Fernandes	1	6
3	332/5	Marcelo Correia Luz	0	0
4	4539/1	Mariana Gonçalves Coelho	1	4

Registros dos alunos e ex-alunos

02

Os registros que estão armazenados na tabela de histórico escolar representam os seguintes valores:

HISTÓRICO ESCOLAR				
Matrícula	**Disciplina**	**Ano**	**Série**	**Nota**
14562/2	Matemática	2015	5ª	6,8
14562/2	Português	2015	5ª	8,9
14562/2	Ciências	2015	5ª	10,0
432/2	Matemática	2015	7ª	8,2
432/2	Português	2015	7ª	8,9
432/2	Geografia	2015	7ª	9,4
432/2	História	2015	7ª	9,0
332/5	Matemática	2010	5ª	8,8
332/5	Português	2010	5ª	8,9
332/5	Ciências	2010	5ª	6,4

Nosso modelo de dados incorpora identificadores (chaves estrangeiras) que armazenam os dados desta maneira:

ID_Historico_Escolar	Ano	Serie	Nota	ID_Aluno	ID_Disciplina
1	2015	5a	6.0	1	1
2	2015	5a	8.9	1	2
3	2015	5a	10.0	1	3
4	2015	7a	8.2	2	1
5	2015	7a	8.9	2	2
6	2015	7a	9.4	2	4
7	2015	7a	9.0	2	5
8	2015	5a	8.8	3	1
9	2015	5a	8.9	3	2
10	2015	5a	6.4	3	3

Registros do histórico escolar de todos os alunos

03

2 - Continuação do estudo sobre a instrução SELECT

Vimos no módulo passado várias configurações e funcionalidades possíveis para criar uma instrução SELECT. Neste e no próximo módulo continuaremos a apresentar outras funcionalidades.

2.1 - Relacionando tabelas por meio do operador JOIN

Além do operador WHERE, a SQL oferece o operador JOIN para fazer o relacionamento entre duas ou mais tabelas. O operador JOIN é muito mais poderoso que o WHERE e permite selecionar que queremos incluir itens que não possuem relacionamentos ou não.

Imagine que existam as tabelas ALUNOS e HISTORICO_ESCOLAR, e que você queira uma lista de todos os alunos e seu respectivo histórico escolar. Usando a cláusula WHERE, caso um aluno não tivesse ao menos um registro na tabela de histórico escolar, ele não apareceria na consulta, pois não existe uma tupla viável para esse aluno entre essas duas tabelas. É nesse momento que a cláusula JOIN entra para resolver a questão. Com ela você pode informar ao SGBD para incluir todos os itens que não possuam relação de uma ou outra tabela. O padrão para a criação de uma relação do tipo JOIN é a seguinte:

```
SELECT <<campos>>
FROM <<Tabela1>> <<tipo_de_join>> JOIN <<Tabela2>>
ON <<Tabela1.chave>> = <<Tabela2.chave>>
```

Os três tipos de relação JOIN são: **INNER JOIN**, **LEFT JOIN** e **RIGHT JOIN**. Vamos ver as características dessas relações e exemplos para ficar bem claro.

04

Para ilustrar os três tipos de relação JOIN, vamos usar duas tabelas relacionadas, uma que apresenta nome de pessoas e outra que apresenta nomes de animais domésticos. Há pessoas que não possuem animais, e há animais que não possuem donos. Veja o conteúdo das tabelas abaixo:

	Nome
1	João
2	Pedro
3	Cláudio

Tabela de Pessoas

ID	Nome	Dono
1	Bolinha	1
2	Ventania	1
3	Lulu	2
4	Peludo	NULL

Tabela de Animais

As duas tabelas acima refletem o seguinte cenário:

- João tem dois bichinhos de estimação, o Bolinha e o Ventania.
- Pedro tem um animal de estimação, a Lulu.
- Cláudio não tem animais de estimação.
- O animal "Peludo" não possui um dono.

05

2.1.1 - INNER JOIN

O operador INNER JOIN tem comportamento semelhante à cláusula WHERE. Ele elimina os casos em que na relação não pode ser confirmada, ou seja, onde o elemento de uma tabela não se relaciona com o da outra.

C1 - *Relação entre donos de animais e animais usando o operador INNER JOIN.*

```
SELECT P.NOME AS DONO, A.NOME AS ANIMAL
FROM PESSOAS AS P
INNER JOIN ANIMAIS AS A
ON P.ID = A.DONO
```

Essa consulta relaciona ambas as tabelas pelo vínculo entre o identificador (chave primária) da tabela de pessoas com o identificador do dono (chave estrangeira) na tabela de animais, porém eliminando aquelas tuplas que não completam a relação (pessoas que não possuem animais e animais que não tem donos).

O resultado de C6 seria:

Dono	Animal
João	Bolinha
João	Ventania
Pedro	Lulu

Resultado da consulta C7

06

2.1.2 - LEFT JOIN

O operador LEFT JOIN incorpora na seleção todos os itens da tabela da esquerda da consulta SQL que não possuem tuplas correspondentes. Ou seja, todos os itens da tabela da esquerda aparecerão no resultado, mas da tabela da direita somente aqueles que têm relação com a primeira.

C2 - *Relação entre donos de animais e animais usando o operador LEFT JOIN.*

```
SELECT P.NOME AS DONO, A.NOME AS ANIMAL
FROM PESSOAS AS P
LEFT JOIN ANIMAIS AS A
ON P.ID = A.DONO
```

A diferença entre a consulta C8 e a C7 é que a C8 irá apresentar todas as pessoas, inclusive aquelas que não possuem animais. Observe que o Cláudio aparece nessa consulta, porém sem animais (NULL):

Dono	Animal
João	Bolinha
João	Ventania
Pedro	Lulu
Cláudio	NULL

Resultado da consulta C8

07

2.1.3 - RIGHT JOIN

O operador RIGHT JOIN é o oposto da LEFT JOIN. Ele incorpora na seleção todos os itens da tabela da direita da consulta SQL que não possuem tuplas correspondentes. Ou seja, todos os itens da tabela da direita aparecerão no resultado, mas da tabela da esquerda somente aqueles que têm relação com a primeira.

C3 - *Relação entre donos de animais e animais usando o operador RIGHT JOIN.*

```
SELECT P.NOME AS DONO, A.NOME AS ANIMAL
FROM PESSOAS AS P
RIGHT JOIN ANIMAIS AS A
```

```
ON P.ID = A.DONO
```

Observe agora que o Cláudio não aparece na consulta, mas o animal “Peludo” é apresentado sem um dono:

Dono	Animal
João	Bolinha
João	Ventania
Pedro	Lulu
NULL	Peludo

Resultado da consulta C9

08

2.1.4 - OUTER JOIN

O operador OUTER JOIN é junção dos operadores LEFT e RIGHT JOIN. Ele incorpora na seleção todos os itens das tabelas da relação, incluindo os registros que não possuem relação na tupla.

C4 - *Relação entre donos de animais e animais usando o operador OUTER JOIN.*

```
SELECT P.NOME AS DONO, A.NOME AS ANIMAL
FROM PESSOAS AS P
OUTER JOIN ANIMAIS AS A
ON P.ID = A.DONO
```

Observe agora que Cláudio e “Peludo” aparecem na consulta, ambos com relações nulas para seus pares:

Dono	Animal
João	Bolinha
João	Ventania
Pedro	Lulu
Cláudio	NULL
NULL	Peludo

Resultado da consulta C10

09

2.2 - Tratando itens duplicados

Conforme já dissemos, a SQL normalmente trata uma tabela não como um conjunto, mas como um multiconjunto; tuplas duplicadas podem aparecer mais de uma vez em uma tabela, e no resultado de

uma consulta. A SQL **não elimina automaticamente tuplas duplicadas** nos resultados das consultas, pelos seguintes motivos:

- A eliminação de duplicatas é uma operação dispendiosa. Um modo de implementá-la é classificar as tuplas primeiro e depois eliminar as duplicatas.
- O usuário pode querer ver as tuplas duplicadas no resultado de uma consulta.
- Quando uma função agregada é aplicada às tuplas (discutiremos isso em outro módulo), na maioria dos casos não queremos eliminar duplicatas.

Uma tabela SQL com uma chave é restrita a ser um conjunto, uma vez que o valor de chave precisa ser distinto em cada tupla. Se quisermos eliminar tuplas duplicadas do resultado de uma consulta SQL, usamos a palavra-chave DISTINCT na cláusula SELECT, significando que apenas as tuplas distintas deverão permanecer no resultado.

Em geral, uma consulta Com SELECT DISTINCT elimina duplicatas, enquanto uma consulta com SELECT ALL não elimina. A especificação de SELECT sem ALL é equivalente a SELECT ALL.

10

C5 - *Listar o nome de todos os alunos que possuem histórico escolar:*

```
SELECT A.Nome AS Aluno
FROM ALUNO AS A, HISTORICO_ESCOLAR AS H
WHERE A.ID_Aluno = H.ID_Aluno
```

Entretanto, para cada registro de histórico escolar, o SGBD listaria o nome dos alunos, produzindo um resultado com vários nomes repetidos, dessa forma:

Aluno
Thiago Ferreira Borges
Thiago Ferreira Borges
Thiago Ferreira Borges
Isadora Luccas Fernandes
Isadora Luccas Fernandes
Isadora Luccas Fernandes
Isadora Luccas Fernandes
Marcelo Correia Luz
Marcelo Correia Luz
Marcelo Correia Luz

Resultado da consulta C11

Como o aluno Thiago tem três registros em histórico escolar, o nome dele aparece três vezes. A mesma lógica vale para os demais alunos. Diante desse problema, o uso da cláusula `DISCTINCT` iria eliminar os valores repetidos.

11

C6 - *Listar o nome de todos os alunos que possuem histórico escolar, porém sem repetição de nomes:*

```
SELECT DISTINCT A.Nome AS Aluno
FROM ALUNO AS A, HISTORICO_ESCOLAR AS H
WHERE A.ID_Aluno = H.ID_Aluno
```

Para essa nova consulta, o resultado será:

Aluno
Thiago Ferreira Borges
Isadora Luccas Fernandes
Marcelo Correia Luz

Resultado da consulta C12

12

2.3 - Ordenando campos

Uma poderosa e muito utilizada funcionalidade é a capacidade de ordenar itens da consulta SQL. A ordenação pode ser baseada em datas, números e/ou caracteres (alfabética). A SQL permite ordenar vários campos de uma só vez, bem como escolher se deseja a ordem normal (do menor para o maior) ou a ordem inversa.

A cláusula que informa a ordenação é denominada ORDER BY. Para indicarmos que desejamos ordem inversa utilizamos DESC no final da expressão.

Vamos ver alguns exemplos de como funciona. Para não gerar muito texto, omitiremos as tabelas resultado.

C7 - *Realizar a mesma pesquisa C12, entretanto, ordenando os nomes dos alunos alfabeticamente.*

```
SELECT DISTINCT A.Nome AS Aluno
FROM ALUNO AS A, HISTORICO_ESCOLAR AS H
WHERE A.ID_Aluno = H.ID_Aluno
ORDER BY A.Nome
```

Observe a expressão `ORDER BY` no final da instrução SQL, é nesse local que ela deve aparecer (depois da cláusula `WHERE` se ela existir).

13

C8 - *Realizar a mesma consulta 13, porém, ordenando os nomes na ordem alfabética inversa.*

```
SELECT DISTINCT A.Nome AS Aluno
FROM ALUNO AS A, HISTORICO_ESCOLAR AS H
WHERE A.ID_Aluno = H.ID_Aluno
ORDER BY A.Nome DESC
```

Note agora o uso do operador `DESC` para indicar a ordem alfabética invertida.

C9 - *Listar as turmas, ordenando primeiro por série (na ordem crescente) e depois por turma (na ordem decrescente).*

```
SELECT Numero, Serie
FROM TURMA
ORDER BY Serie, Numero DESC
```

Nesse exemplo, primeiro o SGBD irá ordenar por séries, posteriormente, ordenará por número na ordem decrescente.

C10 - *Realizar a mesma consulta C15, porém ordenando a série decrescentemente e a turma na ordem crescente.*

```
SELECT Numero, Serie
FROM TURMA
ORDER BY Serie DESC, Numero
```

Para C16, basta mudar a posição do operador `DESC`.

14

2.4 - Operador LIKE

Quanto tratamos de número e datas fica fácil sempre usar o valor exato para encontrar um dado. Por exemplo, se quisermos mostrar todos os valores iguais a dez, usamos algo como `WHERE <campo_numerico> = 10`. Quando queremos uma data posterior a 01/01/2015, usamos, `WHERE <campo_data> >= '01-01-2015'`.

Agora, para encontrar um texto não é tão fácil assim na prática, pois somente se colocarmos a informação exatamente como está cadastrada será possível que o SGBD localize a informação. Por

exemplo, para exibirmos a matrícula do aluno Thiago temos que escrever a seguinte expressão usando o seu nome completo:

```
SELECT Matricula FROM ALUNO WHERE Nome = 'Thiago Ferreira Borges'.
```

Se não soubermos seu nome completo ou se errarmos na digitação em algum caractere, certamente o SGBD não irá encontrá-lo.

Para facilitar a localização de registro de texto é muito comum utilizarmos apenas um trecho do texto para pesquisa. Por exemplo: Qual é a matrícula do aluno Thiago? Sem saber o nome completo, é possível usar o operador LIKE para encontrar algo que "se pareça com" o que estamos informando na pesquisa.

15

Além de usar a expressão LIKE precisamos usar um símbolo para identificar se queremos pesquisar tudo que 'comece com', que 'contenha' ou que 'termine com' determinada sequência de caracteres. O símbolo que define essa opção é o de percentual (%).

% no **início** da expressão	% no **final** da expressão	% no **início** e no **final** da expressão
•Ao utilizarmos no **início da expressão** de pesquisa, significa que qualquer informação que termine com o texto informado será considerada um resultado válido. Exemplo: LIKE "%Barros" teria como valores válidos "Marcelo Barros" e "Pedro Barros". Mas, "Marcelo Barros Costa" seria considerado inválido, visto que a expressão deve terminar com "Barros".	•Já se utilizarmos o percentual no **final da expressão**, significa que o texto pesquisado deve começar com a expressão informada. Exemplo: LIKE "Paulo%" teria como valores válidos "Paulo Barros" e "Paulo Cesar Barros". Mas, "Marcus Paulo Costa" seria considerado inválido, visto que a expressão deve começar com "Paulo".	•Se utilizarmos um símbolo de percentual no **início e outro no final**, significará que qualquer registro que contenha aquele trecho de informação, seja no início, no meio ou no final, será mostrado como resultado na pesquisa. Exemplo: LIKE "%Paula%" teria como valores válidos "Paula Barros", "Paula Cesar Barros", "Maria Paula Costa" e "Francisco Lima de Paula", visto que a expressão "Paula" deve aparecer em qualquer parte do texto.

O operador LIKE geralmente não diferencia letras maiúsculas de minúsculas. Portanto, uma operação do tipo LIKE "%Marcelo%", LIKE "%marcelo%", LIKE "%MARCELO%" ou até mesmo LIKE "%mArCeLo%" são idênticas e trazem o mesmo resultado. Já numa cláusula WHERE cada um desses valores seria diferente entre si.

16

Vamos ver alguns exemplos para ficar bem claro o uso do `LIKE` e do %:

C11 - *Mostrar todos os alunos que comecem com "Marcelo".*

```
SELECT * FROM ALUNOS WHERE Nome like 'Marcelo%'
```

Usando o operador no final, significa que o nome do aluno deve começar com Marcelo. No nosso caso iria apresentar todos os dados do aluno Marcelo Correia Luz.

C12 - *Mostrar todos os alunos que contenham o nome "Marcelo".*

```
SELECT * FROM ALUNOS WHERE Nome like '%Marcelo%'
```

Se alguém se chamasse "Augusto Marcelo Ferreira", também seria apresentado como resultado para essa consulta, pois com o percentual no início da expressão, não importa a localização do trecho pesquisado.

C13 - *Mostrar todos os alunos que contenham o nome "Mar".*

```
SELECT * FROM ALUNOS WHERE Nome like '&Mar%'
```

De acordo com esse exemplo, os alunos Marcelo Correia Luz e Mariana Gonçalves Coelho seriam apresentados.

C14 - *Mostrar todos os alunos que terminem o nome com "es".*

```
SELECT * FROM ALUNOS WHERE Nome like '%es'
```

RESUMO

Neste módulo, aprendemos que:

a) O operador JOIN permite relacionar duas ou mais tabelas. Ele possui três partes:

   a. Na cláusula SELECT aparecerão os campos a serem apresentados.

   b. Na cláusula INNER JOIN aparecerão as tabelas a serem relacionadas.

   c. Na cláusula ON aparecerão os campos referenciados que relacionam as tabelas.

b) Há quatro principais formas de se usar o operador JOIN:

a. LEFT JOIN – inclui todos os registros da tabela da esquerda e todos o da tabela da direita que possuem relação.

b. RIGHT JOIN – inclui todos os registros da tabela da direita e todos o da tabela da esquerda que possuem relação.

c. INNER JOIN – inclui somente os registros das tabelas que possuem relação, os valores nulos são excluídos.

d. OUTER JOIN – inclui todos os registros de ambas as tabelas, incluindo os nulos.

c) O operador DISTINCT elimina itens duplicados em uma consulta.

d) A cláusula ORDER BY permite ordenar pesquisa por um ou mais campos. Quando a ordem inversa é desejada, devemos usar a expressão DESC logo após o nome do campo de ordenação. Múltiplas ordenações são permitidas, usando o sinal de vírgula para separar os campos de ordenação.

e) O operador LIKE permite a busca por parte de um registro de texto. Utiliza-se o valor % para informar se a expressão pesquisada está no início, meio ou final do texto.