3 Fundamentos e Conceitos Técnicos

3.1 Campos, Registros e Tabelas

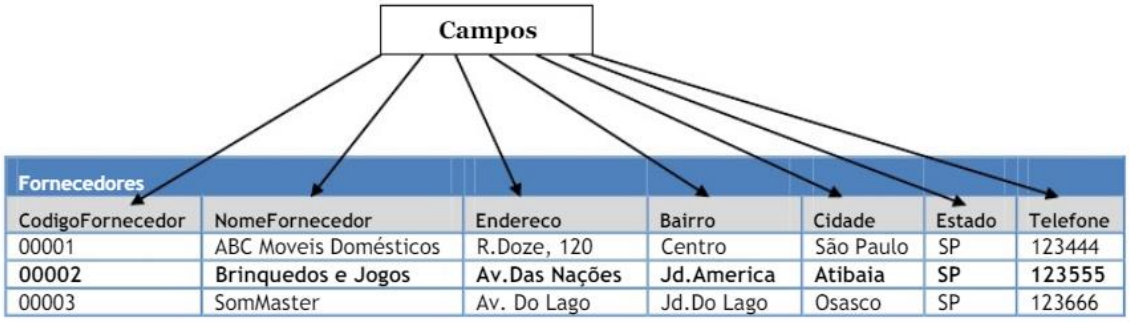
Os bancos de dados possuem seu próprio vocabulário de termos técnicos. Os termos se referem aos objetos que são os fundamentos dos bancos de dados.

3.1.1 Campos de um Registro

O campo é a menor unidade destinada ao armazenamento de valores existente em um arquivo ou tabela de um banco de dados e pode ser referenciado por um programa aplicativo. Isso significa que os dados armazenados são separados em pequenos fragmentos. Cada campo somente pode conter um tipo de dado. Tomemos como exemplo a seguinte informação:

```
Brinquedos & Jogos Educar
Av. das Nações, 280
Jd. América
Atibaia
SP
```

Listada dessa forma, não faz muito sentido do ponto de vista de um banco de dados. Embora forneça uma informação completa sobre uma empresa (no caso um fabricante de brinquedos), para ser armazenada num banco de dados é preciso separá-la em diversas partes. Podemos assumir que cada linha é uma fração da informação como um todo. Imagine-a distribuída numa folha quadriculada em que cada item/linha ocupa uma coluna (como nas planilhas eletrônicas).



Durante a estruturação do banco de dados, uma das principais tarefas do projetista do sistema é definir os campos que comporão as tabelas. Cada campo recebe um nome de identificação, a especificação do tipo de dado que será capaz de armazenar e o tamanho para armazenamento,entre outras informações.

Os tipos de dados básicos que podemos atribuir aos campos são: caractere, numérico (com ou sem casas decimais), data, hora e lógico (valores do tipo verdadeiro ou falso). Nem todos os sistemas de banco de dados suportam o tipo de dado lógico e alguns possuem um tipo especial denominando auto-incremento. Com esse tipo o valor do campo (que é numérico sem casa decimal) é incrementado pelo próprio sistema quando um novo registro é adicionado a tabela. Isso é muito útil quando precisamos gerar um valor seqüencial único para cada registro. Geralmente um campo desse tipo é usado na definição de uma chave primaria.

Há sistemas de banco de dados, como é o caso dos servidores SQL, que permitem a criação de campos calculados. Esse tipo de campo é fornecido por uma expressão matemática, que envolve outros campos da própria tabela ou mesmo constantes numéricas. Seu valor é o resultado obtido por meio dessa expressão. Por exemplo, suponhamos que você tenha uma tabela de pedidos de cliente e nessa tabela constem campos para entrada da quantidade do produto e o preço unitário. O preço total do item poderia simplesmente ser obtido multiplicando o valor do campo de quantidade pelo valor do campo de preço unitário. Em termos simples, a especificação do campo seria parecida com a seguinte instrução fictícia:

DEFINIR "VALORTOTAL" COMO CALCULO ("QUANTIDADE" * "PRECOUNITARIO")

Em alguns sistemas também é possível definir um valor-padrão para o campo, o que significa que se o usuário não fornecer um valor, será assumido o padrão.

Os sistemas mais recentes incluem um novo tipo de dado chamado BLOB (Binary Large Objects– Objetos Binários Grandes) e permitem o armazenamento de dados não-estruturados, como imagens, sons, vídeos, etc. Geralmente os valores de campos BLOB são armazenados separadamente dos respectivos registros no arquivo de dados, e um ponteiro no registro faz referencia a esses valores.

Diversos sistemas permitem que campos do tipo caractere (string) sejam definidos com tamanho fixo ou variável. No caso de tamanhos fixos, se o usuário entrar com dados cujo comprimento seja menor que o especificado para o campo, os bytes extras são preenchidos com espaços em branco.Já com campos de tamanho variável, somente os caracteres fornecidos pelo usuário serão armazenados.

A definição de nome e atributos (tipos de dados e tamanhos) dos campos constitui o que chamamos de **formato de registro**
.

3.1.2 Registros (ou Linhas)

Um registro é o conjunto de campos valorizados de uma tabela. É a unidade básica para o armazenamento e recuperação de dados e que identifica a entrada de um único item de informação em particular numa tabela do banco de dados. São também chamados de tuplas ou n-uplas. Numa tabela cujos registros são formados por cinco campos, cada registro é denominado de 5-upla (ou quíntupla). Também podemos chamar os registros de linhas de uma tabela, pois durante sua visualização os dados dos campos são todos listados numa única linha. Se uma tabela de cadastro de clientes possuir 20.000 linhas ou registros, então ela estará armazenado dados de 20.000 clientes. Os registros de uma tabela de dados são do mesmo tipo, ou seja, permitem o armazenamento dos mesmos tipos de dados. No entanto, seus campos podem ser de tipos e tamanho variável, o tamanho do próprio registro pode também variar conforme os dados que se encontram

armazenados nele. Por exemplo, numa tabela de cadastro de produtos, os registros são utilizados para guardar os dados referentes a produtos, não sendo possível armazenar qualquer outro tipo de dado (como de clientes ou de funcionários).

É durante a estruturação das tabelas do banco de dados que definimos o formato (ou layout) dos registros. Como já mencionado, esse formato de registro envolve a disposição dos seus campos com nomes e atributos.

Voltando ao exemplo do tópico anterior, cada linha da tabela representa um registro. Assim a tabela como um todo se resume a um agrupamento de linhas (registros) que são divididos em colunas (campos).

3.1.3 Tabelas de Dados

Uma tabela nada mais é do que um conjunto ordenado de registros/linhas. Cada registro possui o mesmo numero de colunas (campos). Um banco de dados pode ser formado por uma ou mais tabelas, e cada uma deve ser definida de tal forma que somente possa conter um tipo de informação. Por exemplo, uma tabela para armazenar dados dos clientes, uma para os fornecedores, uma para os produtos, etc. Falando em termos de modelagem de dados, elas representam as entidades do modelo conceitual.

Alguns sistemas de banco de dados, como MySQL, criam um arquivo para cada tabela. Outros, como o Interbase, possuem um único arquivo, dentro do qual estão todas as tabelas (alem de outros recursos, como rotinas autocontidas e índices).

Cada tabela deve ser identificada por um nome único dentro do banco de dados. São as tabelas que possuem toda a estrutura/composição dos registros, como nomes, tipos de dados e tamanhos. Uma aplicação de banco de dados somente pode acessar um determinado registro se referenciar a tabela na qual ele esta definido. Na linguagem SQL também é necessário especificar o nome da tabela a ser utilizada num comando de consulta ou atualização de dados.

As aplicações normalmente utilizam varias tabelas do banco de dados para consolidar e retornar informações nas quais o usuário tem interesse (como na geração de um relatório de vendas no mês ou o boletim escolar dos alunos).

3.2 Índices, Chave Primaria, Chaves Estrangeiras, Chaves Candidata e Domínios

3.2.1 Índices

Quando precisamos procurar um determinado assunto num livro ou em uma enciclopédia, geralmente recorremos ao índice para saber em que volume e pagina se encontra a informação. Os índices nos bancos de dados têm a mesma funcionalidade, ou seja, permitem que os registros com dados sejam encontrados com extrema rapidez. Eles também oferecem uma maneira de acesso alternativo aos registros sem que sua posição física dentro do banco de dados seja modificada. Por exemplo, se desejarmos listar na impressora todos os clientes em ordem alfabética de nomes,podemos utilizar um índice com base nesse campo.

Um índice pode ser simples (quando é formado por um só campo) ou composto (formado por vários campos da tabela). Denominamos os campos que são utilizados na definição de índices como **campos de indexação**. Os índices não contem dados propriamente ditos, mas apenas o valor do campo de indexação e "ponteiros" que direcionam para o registro adequado dentro da tabela. A informação contida neles é automaticamente atualizada, ou seja, se inserirmos ou excluirmos um registro, o índice também será modificado.

Alguns sistemas de bancos de dados (notadamente os servidores SQL) fazem uso de índices automaticamente para agilizar as consultas (quando um índice tiver sido criado para o campo envolvido no comando de consulta). É o que se costuma chamar de "plano de consulta".

Uma característica interessante que diversos sistemas oferecem com relação aos índices é a capacidade de ordenação dos registros de forma crescente ou decrescente. Outro recurso muito útil é a possibilidade de definir índices únicos, ou seja, que não permitem duplicidade de dados.Esse tipo de índice é independente da chave primaria. Vamos supor como exemplo uma tabela de cadastro de fornecedores, em que cada registro possui como chave primaria o código do fornecedor. Podemos definir um índice com base no campo CNPJ que não permite duplicidade,evitando dessa forma que um mesmo fornecedor seja cadastrado duas vezes.

Para localizar um registro utilizando índice, o banco de dados efetua uma pesquisa binária no índice e ao encontrar um ponteiro para o valor a ser pesquisado, posiciona no registro correspondente da tabela.

3.2.2 Chave Primaria

Uma chave primaria é um atributo (campo) da tabela que permite a identificação de forma única dos seus registros. Ela tem por função ainda aplicar uma ordenação automática aos registros, uma vez que seu funcionamento é similar ao de um índice. Isso significa que a ordem física dos registros é determinada por ela. Como ocorre com os índices, uma chave primaria pode ser formada por um ou mais campos. No ultimo caso, elas são chamadas de chaves compostas.

Uma chave primaria evita que tenhamos registros duplicados, ou seja, não é possível ter dois ou mais registros contendo os mesmos valores nos campos que a compõem.

As chaves primárias permitem que uma pesquisa seja realizada com maior velocidade. Suponhamos que se deseja encontrar o registro em um banco de dados de clientes, cujo campo RG seja "10.101.101". Se tivermos definido esse campo como chave primaria, a pesquisa dar-se-á com maior rapidez.

Há basicamente dois critérios importantíssimos na hora de escolher os campos que vão definir uma chave primaria. Em primeiro lugar devemos escolher campos cujo tamanho não seja muito grande,assim as atualizações e consultas da chave primaria serão mais rápida. Pelo mesmo motivo, quando tivermos chave composta, devemos procurar incluir o menor numero possível de campos.

O segundo critério esta relacionado com a capacidade de alteração do valor armazenado no campo da chave primaria. Esse valor deve ser estável, ou seja, não pode ser modificado (pelo menos deforma tão freqüente). Isso se deve ao fato de os bancos de dados relacionais utilizarem a chave primaria no relacionamento de uma tabela com outras, como veremos a seguir.

Há situações, no entanto, que mesmo sendo parte de chave primaria, um campo deve permitir que seu valor seja alterado. Vamos supor o caso de uma tabela de cadastro de produtos, na qual o campo de código do produto é utilizado para armazenar o código de barras impresso na embalagem. Esse campo deve estar habilitado para permitir modificação, uma vez que o fabricante pode, por um motivo qualquer, alterar o código de barras de um produto que já se encontrava em fabricação.

Para que isso não afete de forma desastrosa toda a integridade dos dados do banco, os sistemas relacionais oferecem recursos para atualização em cadeia dos valores nas tabelas envolvidas nos relacionamentos. Isso, no entanto, acarreta uma ligeira queda no desempenho do sistema,principalmente quando existirem muitos registros a serem atualizados.

Preferencialmente, para campos de chaves primarias, devemos utilizar valores que são calculados de forma seqüencial pelo próprio sistema, seja por meio de recursos SGDB ou por rotinas do aplicativo que faz uso do banco de dados.

Campos que formam chaves primárias são de preenchimento obrigatório.

3.2.3 Chaves Candidatas

Chaves candidatas são campos que poderiam ser utilizados como chaves primárias, mas não são. Por exemplo, numa tabela de cadastro de clientes, a chave primaria é definida como o campo de código do cliente, cujo valor é seqüencial e gerado automaticamente pelo próprio sistema. No entanto, há o campo RG do cliente que, por ser de valor único (não deve repetir dentro do banco de dados), também poderia ser utilizado como chave primária (é uma chave candidata).

No entanto, devido ao fato de ser um campo com tamanho demasiadamente grande, e principalmente por não ser ideal para identificar unicamente um cliente, uma vez que no Brasil uma pessoa pode ter mais de um RG, em estados diferentes, ele não é utilizado na definição da chave primaria. Sendo preferível um campo de codificação para cada registro de cliente.
É um campo de preenchimento obrigatório também.

3.2.4 Chaves Estrangeiras

As chaves estrangeiras permitem que os registros de uma tabela sejam relacionados com os de outra tabela pela sua chave primaria. Para exemplificar, vamos supor duas tabelas com as seguintes características:

- Uma tabela de cadastro de categorias de produtos que possua um campo de código de categoria (CodigoCategoria) que é chave primária.
- Uma tabela de cadastro de produtos que possui um campo de código da categoria a que pertence (CodigoCategoria).

Podemos notar que o campo denominado **CodigoCategoria** existe nas duas tabelas, e na de cadastro de produtos não é a chave primária. Como ela é utilizada para criar um relacionamento com a tabela de categorias, em que o campo é chave primária, dizemos que ela é chave estrangeira.

Nesse caso, o campo **CodigoCategoria** é chave primária na tabela **Categorias** e chave estrangeira na tabela **Produtos**. O vinculo entre as duas tabelas é efetuado por meio desses dois campos. Da mesma maneira, o campo **CodigoFornecedor** é chave primária em **Fornecedores** e chave estrangeira em **Produtos**.

Um outro uso muito comum de chave estrangeira, além da ligação entre tabelas, é no fornecimento de valores de outra tabela (a que contem a chave primaria). Por exemplo, poderíamos utilizar uma caixa de listagem para escolha da categoria de produtos numa tela de cadastro de produtos,evitando assim que o usuário entre com informações indevidas, como código de categoria inexistente.

Apesar de normalmente o nome do campo (ou conjunto de campos) que define a chave primária e a chave estrangeira ser o mesmo, é perfeitamente possível nomeá-los de forma distinta.

3.2.5 Domínios

O domínio é um tipo de conceito ate certo ponto difícil de demonstrar. A melhor maneira de entendê-lo é com exemplos concretos. Os domínios se prestam a dois objetivos básicos: definição de tipos de dados e especificação de valores que podem ser aceitos pelos campos. Nem todos os sistemas de bancos de dados suportam a criação e uso de domínios. Em geral, todos os sistemas padrão SQL dão suporte total.

Uma das maiores vantagens de utilizar domínios na construção de banco de dados é o fato de podermos padronizar os atributos de campos que existem em varias tabelas do banco de dados.Vamos supor que precisamos definir cinco campos para armazenar o endereço, bairro, cidade,estado e CEP. O formato desses campos é apresentado na seguinte tabela:

Campo	Tipo de dado	Tamanho
Endereco	Caractere	50 caracteres
Bairro	Caractere	35 caracteres
Cidade	Caractere	35 caracteres
Estado	Caractere	2 caracteres
CEP	Caractere	9 caracteres

*Apesar de a informação do CEP ser composta apenas por números, devemos definir o campo como sendo caractere para que seja possível armazenar zeros apresentados a esquerda e o traço de separação, como, por exemplo, “00123-456”.

Continuando com o exemplo, esses campos serão criados em três tabelas: cadastro de funcionários,cadastro de clientes, cadastro de fornecedores. A probabilidade de definirmos um ou mais campos com tamanho diferente é grande.

Com os domínios essa probabilidade de erro pode ser evitada. Para isso devemos criá-los,especificando os atributos necessários a cada campo, e depois empregá-los, especificando os atributos necessários a cada campo, e depois empregá-los na definição dos campos. Vamos utilizar uma linguagem imaginaria para demonstrar a criação de domínios para esse campo:

```
DEFINIR DOMINIO "DM_ENDERECO" COM TIPO CARACTERE E TAMANHO 50;
DEFINIR DOMINIO "DM_BAIRRO" COM TIPO CARACTERE E TAMANHO 35;
DEFINIR DOMINIO "DM_CIDADE" COM TIPO CARACTERE E TAMANHO 35;
DEFINIR DOMINIO "DM_ESTADO" COM TIPO CARACTERE E  TAMANHO 2;
DEFINIR DOMINIO "DM_CEP" COM TIPO CARACTERE E TAMANHO 9;
```

Na criação da tabela de dados, devemos fazer referencia a esses domínios da seguinte maneira(linguagem imaginária):

```
CRIAR TABELA "FUNCIONARIOS" COM FORMATO
(CAMPO ENDERECO COM ATRIBUTO "DM ENDERECO",
 CAMPO BAIRRO COM ATRIBUTO "DM_BAIRRO",
 CAMPO CIDADE COM ATRIBUTO "DM_CIDADE",
 CAMPO ESTADO COM ATRIBUTO "DM_ESTADO",
 CAMPO CEP COM ATRIBUTO "DM_CEP")
```

O mesmo método seria empregado para as outras duas tabelas de dados. Com isso não há como esquecermos as propriedades para um determinado campo, com, por exemplo, seu tamanho em caracteres.

O segundo uso dos domínios, como já mencionado, é na especificação de valores limites ou faixa de dados validos. Por exemplo, podemos criar um domínio para não permitir que o usuário tente entrar com um valor menor ou igual a zero no campo de salário do funcionário. Utilizando nossa linguagem imaginaria, teríamos:

```
DEFINIR DOMINIO "DM_SALARIO" COM TIPO DECIMAL E VALOR MAIOR QUE 0;
```

A regra principal por trás dos domínios é similar a da programação orientada a objetos: defina uma vez, use varias vezes.

3.3 Integridade dos Bancos de Dados

Uma das maiores preocupações de qualquer desenvolvedor ou projetista de banco de dados é encontrar uma forma de garantir a integridade dos dados que se encontram armazenados. Essa preocupação se deve ao fato de que, se houver algum dado crucial armazenado de forma incorreta,o resultado pode ser catastrófico, pois o banco de dados pode apresentar informações imprecisas ou mesmo totalmente errôneas.

Imagine a seguinte situação: uma aplicação de contas a receber tem em seu banco de dados vinte e dois registros de pagamentos em aberto, referentes a um determinado cliente. A ligação entre essas informações (o cadastro do cliente e o pagamento em aberto) é feita pelo campo de código do cliente, e por isso o valor desse campo não pode ser alterado de forma nenhuma. Agora digamos que um usuário ou o próprio administrador do banco de dados tenha, fora da aplicação, aberto o banco de dados e alterado o valor do campo do código do cliente na tabela de cadastro de clientes.Logicamente o vinculo será quebrado, pois os campos utilizados no relacionamento agora possuem valores distintos. Da para ter uma idéia do enorme problema que isso causaria.

Se o próprio sistema de banco de dados oferecerem formas de restringir ou mesmo impossibilitar a quebra dessa integridade, o desenvolvedor terá menos trabalho, pois não será necessário escrever código que faca essa tarefa dentro do aplicativo. Alem disso, como no exemplo, acessando o banco de dados fora da aplicação, às informações estarão vulneráveis.

Como solução para este problema, a grande maioria dos sistemas hoje existentes possui recursos capazes de gerenciar essa integridade de dados. A seguir relacionamos algumas regras básicas que um sistema de banco de dados deve contemplar para garantir a integridade.

3.3.1 Integridade de Entidades

A regra de integridade define que as chaves primárias de uma tabela não podem ser nulas, ou seja,sempre devem conter um valor. No caso de chaves primárias compostas (formadas por mais de um campo), todos os campos que a formam devem ser preenchidos.Essa exigência se deve ao fato de a chave primária ser um atributo da tabela que identifica um registro único, alem de determinar a ordem física do registro e também ser utilizada no relacionamento com outras tabelas. Devemos ressaltar que um valor nulo é diferente de 0 ou um espaço em branco. Com campo do tipo caractere, um espaço em branco é considerado um valor valido, enquanto um valor nulo seria realmente a ausência de qualquer caractere (mesmo espaços).Já com campos do tipo numérico, seja inteiro ou decimal, o valor 0 também é valido. O valor nulo para esse campo também seria a ausência de qualquer dado (mesmo o 0).

3.3.4 Integridade de Referencial

De forma bastante resumida, podemos dizer que a regra de integridade referencial estabelecer restrições ou bloqueios de algumas operações (notadamente exclusão ou alteração) nos dados de campos das chaves primarias utilizadas no relacionamento de uma tabela com outra.Com um exemplo pratico fica mais fácil de entender. Vamos pegar o exemplo de tabelas de **Categorias** e **Produtos**, mostradas anteriormente no tópico sobre chaves estrangeiras. A titulo de recordação, temos um campo denominado **CodigoCategoria** que é chave primaria da tabela **Categorias** e chave estrangeira na tabela **Produtos**. A categoria cujo código "00001" se relaciona com um ou mais produtos, como, por exemplo, o de código "123456". Se o usuário,acidentalmente,

excluir o registro da categoria “00001”, os produtos que pertencem a essa categoria ficarão “órfãos”. Teríamos o mesmo resultado se o código da categoria fosse alterado.

Nesse caso, alem de registro órfão na tabela de produtos, teríamos também registros pais sem nenhum vinculo na tabela de categorias.O que a integridade referencial faz é proibir esse tipo de inconsistência, impedindo assim que a relação entre a chave primaria e a chave estrangeira seja quebrada. Quando o próprio sistema de banco de dados se encarrega de mantê-las, o desenvolvedor novamente se vê livre da codificação de rotinas que devem ser incluídas em seus aplicativos para executar essas tarefas de “vigilância”.

O sistema também deve permitir que o desenvolvedor especifique a forma de tratamento dessas ocorrências, escolhendo uma das duas opções: proibição ou execução em cascata. A proibição de uma operação não permite que o usuário exclua ou altere os dados de uma chave primária que é chave estrangeira em outra tabela (a menos que essa ultima não possua nenhum registro relacionado com a tabela da chave primaria). Já na execução em cascata, quando um usuário efetua uma exclusão ou alteração, essa ação é refletida automaticamente nas tabelas relacionadas.No exemplo citado anteriormente, os produtos que pertencem à categoria “00001” serão excluídos junto com a própria categoria. Se houvesse uma alteração no código da categoria, ela seria passada aos registros correspondentes na tabela de produtos.

3.3.4 Integridade de Domínios

A integridade de domínios estabelece restrições e regras na definição dos domínios em si, em vez de diretamente nos campos. Isso ajuda a economizar tempo e trabalho, pois podemos ter vários campos que são formatados a partir de domínios, e com a regra de integridade definida neles, esses campos automaticamente herdam essas restrições.Este é o caso do domínio definido como exemplo anteriormente (em nossa linguagem fictícia):

DEFINIR DOMINIO “DM_SALARIO” COM TIPO DECIMAL E VALOR MAIOR QUE 0;

3.3.4 Integridade de Campos

Dentro da integridade de campos podemos mencionar a validação de dados. Essa regra possibilita que o próprio sistema de banco de dados faca uma verificação quanto aos valores entrados pelo usuário serem validos ou não. No exemplo de cadastro de funcionários, cada funcionário tem um cargo, e seu salário é definido em função do cargo que ocupa. Sendo assim,mesmo que o valor dessa informação esteja dentro da regra definida para o campo (no caso entre R$ 400,00 e R$ 2.500,00), deve ser efetuada uma validação para certificar-se de que o valor fornecido esta dentro do padrão para o cargo do funcionário.

Essa validação, sendo efetuada pelo próprio ambiente, também isola qualquer aplicativo que faca uso do banco de dados.

Um terceiro tipo de integridade de campo diz respeito ao formato dos dados. Por exemplo, para o campo que armazena o numero de CNPJ, as informações devem ter o formato “XX.XXX.XXX/XXXX-XX”. Se pudermos especificar esse formato para entrada de dados do campo diretamente no sistema, não precisaremos nos preocupar em codificar o aplicativo pra fazer alguma conversão. O Microsoft Access permite que sejam especificadas “mascaras de entrada” para cada campo do banco de dados.

Há ainda regras de integridade que podem ser definidas através de rotinas criadas pelo desenvolvedor, como é o caso das **Stored Procedures** e **Triggers** do padrão SQL