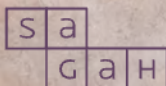


ADMINISTRAÇÃO DE BANCO DE DADOS

Maurício de Oliveira Saraiva



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS



Refinamento de banco de dados

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Reconhecer índices e sua importância para o bom desempenho de consultas.
- Identificar restrições na utilização do banco de dados.
- Aplicar índices para o refinamento do banco de dados.

Introdução

Gravar e consultar informações são atividades que fazem parte das funcionalidades de qualquer sistema de informação que utiliza bancos de dados. Contudo, é preciso garantir que as informações sejam salvas com integridade, bem como facilitar a sua extração por meio de rotinas ágeis.

Para garantir a integridade das informações, sistemas gerenciadores de bancos de dados (SGBDs) aplicam restrições que podem impedir valores indesejados. Esses sistemas também implementam índices que agilizam a execução de consultas nas tabelas.

Neste capítulo, você vai estudar os índices e verificar a sua importância para o bom desempenho de consultas. Além disso, você vai conhecer as restrições que garantem a integridade dos dados.

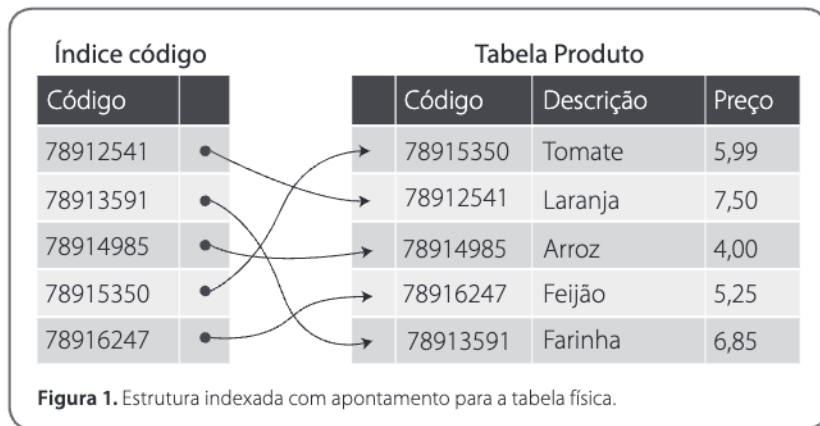
A importância dos índices para o desempenho de consultas

Bancos de dados são utilizados por sistemas de informação para guardar dados a respeito de determinados contextos. Um supermercado, por exemplo, necessita manter dados sobre seus produtos, como código, descrição, preço e registros de venda. O armazenamento desses dados é realizado em arquivos

organizados de tal modo que possibilitem a gravação e a leitura de informações. No entanto, uma tabela que armazena o registro de vendas de um grande supermercado pode conter milhões de registros feitos ao longo dos anos.

Imagine uma consulta que pretende selecionar os produtos vendidos em determinado mês e ano. Para garantir a seleção correta, a instrução teria de percorrer todos os registros da tabela, desde o primeiro até o último, para trazer os dados desejados. Dependendo do tamanho dessa tabela, uma varredura sequencial (*full table scan*), que acessa todos os registros, pode ser muito demorada. Para melhorar o desempenho de consultas que filtram dados, SGBDs implementam índices que auxiliam na seleção de registros por meio de indexações. É o caso do índice de uma agenda classificada em ordem alfabética.

De acordo com Ramakrishnan e Gehrke (2013, p. 228), “um índice é uma estrutura de dados que organiza registros de dados em disco para otimizar determinados tipos de operações de recuperação”. Isso significa que cada índice armazena seu(s) campo(s) de maneira ordenada crescente ou decrescente. Cada linha do índice contém o endereço físico do registro na tabela. Na Figura 1, você pode ver um exemplo de índice de uma tabela nomeada **Produto**.



Como você pode ver na Figura 1, a tabela de produtos não está ordenada por nenhum campo, de modo que todo registro novo é adicionado no final. Não havendo índice em nenhum campo dessa tabela, a varredura ocorrerá sequencialmente para qualquer tipo de pesquisa em seus campos.

Com a definição de um índice para o campo de código da tabela **Produto**, uma nova estrutura (índice **código**) será criada contendo apenas o código e um endereçamento para o seu respectivo registro na tabela. Desse modo, ao executar uma consulta que seleciona determinado código, o SGBD identifica que existe um índice e faz a busca na estrutura indexada por meio de algum algoritmo de seleção em uma lista ordenada. É o que acontece no caso da busca binária, por exemplo.

A busca binária realiza sucessivas divisões na estrutura indexada por meio do elemento central. A cada divisão, um novo elemento central é identificado. Assim, a busca verifica se o elemento procurado está posicionado antes ou após o elemento central atual. Com isso, a metade dos registros restantes são descartados a cada divisão, até que o elemento procurado seja encontrado com poucas comparações (MANNINO, 2008).

Após encontrar o código procurado na estrutura indexada, ocorre a captura do endereço físico desse código. Isso torna possível acessá-lo diretamente na tabela física por meio de um cálculo de endereçamento. Desse modo, o registro procurado é selecionado com poucas comparações, sem a necessidade de uma varredura em todos os registros da tabela.



Saiba mais

É possível criar vários índices para uma mesma tabela. Esses índices podem ser identificados por apenas um campo (índices simples) ou por mais de um campo (índices compostos). Nos índices compostos, a ordenação ocorre primeiro pelo campo mais à esquerda, seguindo a ordem dos demais campos.

Parece tentador criar índices para todos os campos de uma tabela, porém, você deve considerar que a criação de índices desnecessários, além de desperdiçar espaço para o armazenamento das estruturas indexadas, pode fazer com que o SGBD consuma tempo para determinar quais índices utilizar a cada consulta. Desse modo, é preciso ter cautela na criação dos índices. Uma boa prática é identificar os campos mais pesquisados nas tabelas do banco de dados. Em um supermercado, por exemplo, os códigos dos produtos são os mais utilizados, pois as leituras para a verificação de preços e o lançamento de vendas são realizadas por meio deles.

Em tabelas com poucos registros, não é necessário criar índices, uma vez que a varredura completa se torna mais vantajosa. Também pode ser melhor realizar a busca sequencial em tabelas cujos campos possuem muitos valores repetidos, visto que a constante pesquisa da estrutura indexada na tabela física pode consumir mais tempo do que a varredura *full table scan* na própria tabela.

Existem diferentes tipos de índices implementados nos diversos SGBDs disponíveis no mercado. No MySQL, por exemplo, os tipos de índices existentes são: *primary*, *unique*, *index* e *fulltext*. Cada um desses tipos atende a um determinado critério de aplicação, com base nas vantagens e desvantagens do contexto de atuação (MYSQL, 2019a). A seguir, você pode conhecer melhor cada um deles.

- Índice *primary*: é um índice criado automaticamente pela chave primária de uma tabela, podendo conter um ou mais campos. Não permite valores nulos (NULL).
- Índice *unique*: define a restrição de que uma ou mais colunas não podem conter valores repetidos, com a exceção de valores nulos que são permitidos.
- Índice *index*: representa um índice normal de um ou mais campos de uma tabela.
- Índice *fulltext*: otimiza a busca de palavras ou frases em campos de texto longos, proporcionando, inclusive, busca por proximidade de palavras.

Colunas de chave primária podem ser armazenadas de maneira ordenada na tabela física. Essa opção é conhecida como “índice clusterizado” ou “índice interno”. Ela permite a busca indexada diretamente na tabela, sem o uso de estrutura de índice adicional — como ocorre nos demais tipos de índice. Esse índice possui a vantagem de realizar consultas de maneira mais otimizada, pois não utiliza estruturas indexadas para localizar os registros pesquisados.



Fique atento

Por padrão, os índices são armazenados em ordem crescente. No entanto, alguns SGBDs também permitem que os índices sejam criados em ordem decrescente.

Índices também são fundamentais no relacionamento entre tabelas, cujas chaves estrangeiras são indexadas nas tabelas relacionadas para garantir a rápida seleção de registros associados por meio de instruções *Structured Query Language* (SQL) nos bancos de dados relacionais, como o MySQL.

A definição dos tipos dos índices e dos campos que receberão índices depende de diversos fatores que devem ser avaliados pelo projetista de *software*. Apesar de os índices atuarem no desempenho de consultas, a inserção e a exclusão de dados em tabelas indexadas são mais custosas, pois o SGBD precisa atualizar as estruturas indexadas, além de salvar os dados nas tabelas físicas.



Saiba mais

Existem recursos de bancos de dados que auxiliam na identificação de gargalos em consultas. Esses recursos são conhecidos como “planos de avaliação de consultas”. Eles permitem identificar quais índices são selecionados pelo SGBD na execução de instruções SQL de seleção de dados. Leia o capítulo 12 de Ramakrishnan e Gehrke (2013) para saber mais sobre otimização de consultas.

Restrições em bancos de dados

Armazenar informações íntegras e consistentes nos bancos de dados é um fator importante para toda organização. Nesse contexto, diversos SGBDs fornecem recursos para manipular dados que visam a garantir a confiabilidade e a integridade dos dados por meio de restrições (*constraints*) em bancos de dados (MYSQL, 2019b).

As restrições podem ser definidas no momento da criação das tabelas do banco de dados ou posteriormente, por meio de instruções de *Data Definition Language* (DDL), como ALTER TABLE, executadas por um administrador de banco de dados (*DataBase Administrator* [DBA]). A seguir, você vai conhecer as principais restrições de integridade de banco de dados conforme detalhado em MySQL (2019c).

Restrição de chave primária

Uma chave primária (*primary key* [PK]) compreende a identificação de uma ou mais colunas de uma tabela de forma única. Os valores dessas colunas possuem a restrição de não conter dados nulos. As chaves primárias atuam principalmente nas atividades listadas a seguir.

- Identificação dos registros: as chaves primárias identificam cada registro (linha/tupla) das tabelas de forma única, sem repetição de dados.
- Indexação dos registros: as linhas com chave primária são armazenadas de maneira ordenada na tabela — indexação interna.
- Relacionamentos entre as tabelas: as chaves primárias atuam nos relacionamentos e nas relações de integridade entre as tabelas.

Uma chave primária é dita simples quando compreende apenas uma coluna. Ela é composta quando envolve duas ou mais colunas. Nesse caso, a ordenação interna se dá pela ordem das colunas na tabela. Um índice primário é todo índice definido automaticamente pela chave primária de uma tabela, podendo compreender um ou mais campos. O desempenho de pesquisas em campos que pertencem às chaves primárias é ainda mais otimizado devido à restrição NOT NULL.

Em tabelas que não possuem um campo próprio para a identificação da chave primária, uma boa prática é a indicação de um campo numérico com valores definidos por autoincremental (*Auto-Increment* [AI]). Esses campos com identificação exclusiva podem servir para realizar a associação entre tabelas de maneira bastante otimizada.

Restrição de valores nulos

Esse tipo de restrição atua na garantia de não aceitar valores nulos para determinadas colunas de uma tabela. Quando essa restrição é definida, as operações de inclusão e alteração de dados que tentarem definir valores nulos para as colunas indicadas serão descartadas pelo SGBD, retornando um erro de violação de restrição.

Restrição de coluna única

Uma restrição *unique index* permite que uma coluna seja identificada unicamente em uma tabela, ou que um conjunto de colunas receba essa mesma garantia de forma agrupada — combinando os valores de todas as colunas. Diferentemente da restrição de chave primária, essa restrição permite várias *unique index* em uma mesma tabela. Além disso, permite que as colunas identificadas com essa restrição recebam valores nulos, pois apenas os campos com valores diferentes de nulo são impedidos de receber dados repetidos.

Como boa prática, usa-se essa restrição em colunas que não pertencem à chave primária de uma tabela, mas que devem possuir valores únicos, como os números dos documentos de uma pessoa — RG, CPF, título de eleitor, etc. Em um cadastro de aluno de uma instituição de ensino, por exemplo, a chave primária pode ser identificada como um número sequencial chamado Registro de Aluno. No entanto, se houver uma restrição *unique index* em determinados campos, tal como o CPF, o SGBD impedirá que o mesmo aluno seja incluído duas vezes na base de dados.

Restrição de *check*

A restrição de *check* (CK) define um conjunto de valores que podem se aceitar por determinada coluna de uma tabela do banco de dados. É possível criar uma *check* composta de vários critérios para um mesmo campo ou várias *checks* para campos diferentes.



Link

Bancos de dados MySQL anteriores à versão 8.0.16 não suportam a instrução *check*, apesar de aceitar o comando na definição das tabelas. Como medida contingencial, as *checks* podem ser implementadas por meio de *triggers* ou *views*. Saiba mais sobre CK no MySQL por meio dos seguintes *links*:

<https://qrgo.page.link/iC1g>

<https://qrgo.page.link/ANfU>

Uma situação de restrição *check* pode ser implementada em uma coluna em que é preciso armazenar o sexo de uma pessoa. Nesse caso, uma CK pode exigir que apenas os valores **masculino** e **feminino** sejam armazenados no campo, garantindo a integridade das informações.

Restrição de chave estrangeira

Uma chave estrangeira (*foreign key*) é uma coluna (ou um conjunto de colunas) de uma tabela utilizada para impor integridade no relacionamento entre duas tabelas. Na relação entre duas tabelas, a chave primária de uma tabela se torna a chave estrangeira da outra.

Na Figura 2, você pode ver um relacionamento entre as tabelas **Produto** e **Categoria**. Nesse relacionamento, a coluna **cat_codigo**, que é chave primária na tabela **Categoria**, é incluída como chave estrangeira na tabela **Produto**. Dessa forma, a coluna **cat_produto** da tabela **Produto** deve receber um valor que esteja presente na coluna **cat_codigo** da tabela **Categoria**.

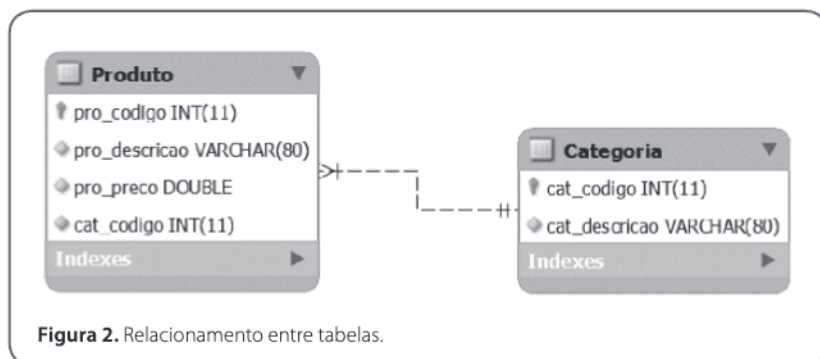


Figura 2. Relacionamento entre tabelas.

Em um relacionamento, o SGBD pode controlar as alterações realizadas na tabela que possui a chave primária por meio de uma restrição conhecida como **integridade referencial**. Isso garante que uma ação seja executada quando alguma linha for excluída ou alterada na tabela da chave primária. As ações que podem ser realizadas pela integridade referencial em cascata são listadas a seguir.

- **NO ACTION:** produz um erro e desfaz as alterações realizadas na tabela que possui a chave primária quando já existem linhas associadas na tabela da chave estrangeira.
- **CASCADE:** as linhas correspondentes da tabela que possui a chave estrangeira são excluídas ou atualizadas automaticamente quando uma linha da tabela da chave primária é excluída ou modificada.
- **SET NULL:** valores NULL são definidos automaticamente nas linhas da tabela que possui a chave estrangeira quando as linhas correspondentes são excluídas na tabela da chave primária.
- **SET DEFAULT:** os valores definidos como padrão são atualizados nas linhas da tabela da chave estrangeira quando as linhas correspondentes da tabela que possui a chave primária são excluídas.

Índices para o refinamento do banco de dados

Para compreender a aplicação de índices, considere o banco de dados MySQL e a ferramenta Computer-Aided Software Engineering (CASE) MySQL Workbench, que pertence ao conjunto de *softwares* disponibilizados pelo MySQL. O MySQL Workbench é uma ferramenta que possui uma interface gráfica e intuitiva e que facilita as atividades de definição de banco de dados, como a criação de tabelas, índices, relacionamentos, etc.

Um recurso importante do MySQL Workbench é realizar operações de atualização do banco de dados a partir do modelo de Entidade Relacionamento (ER) e vice-versa. Então, por meio de engenharia reversa, ele pode criar modelos ER a partir de bancos de dados MySQL (2019d).



Link

Acesse o link a seguir para fazer o download do MySQL Workbench.

<https://qrgo.page.link/RyWs>

Criação de índices no MySQL

No MySQL, existem quatro tipos de índices: *primary*, *index*, *unique* e *fulltext*. Cada um desses tipos possui suas próprias características e vantagens. A aplicação de cada um depende exclusivamente do contexto, como o tamanho da tabela, o número de inserções/atualizações na tabela, entre outros fatores (MYSQL, 2019a).

Aqui, a demonstração da aplicação dos índices será realizada por meio da tabela **Empregado**, que você pode ver na Figura 3. Essa tabela apresenta cinco colunas: **matricula**, **CPF**, **nome**, **data_admissao** e **observacao**, que possuem características adequadas para a explicação dos tipos de índices do MySQL (2019e).

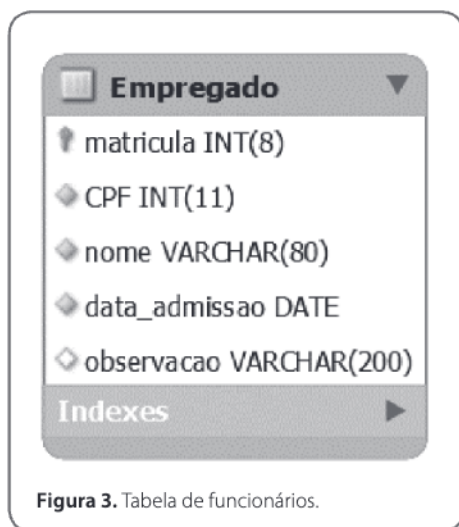


Figura 3. Tabela de funcionários.

Primary

O índice *primary* é criado automaticamente pela chave primária da tabela e mantém as linhas classificadas pela coluna-chave (índice interno). Na tabela **Empregado**, o índice da chave primária está definido para a coluna **matricula**, que é identificada com o ícone de uma chave do seu lado esquerdo na Figura 3.

A seguir, veja a criação de índice *primary* na definição da tabela. A indicação do índice está na linha 7, por meio da instrução PRIMARY KEY.

```
1 CREATE TABLE `mydb`.`Empregado` (  
2   `matricula` INT(8) NOT NULL,  
3   `CPF` INT(11) NOT NULL,  
4   `nome` VARCHAR(80) NOT NULL,  
5   `data_admissao` DATE NOT NULL,  
6   `observacao` TEXT NULL,  
7   PRIMARY KEY (`matricula`))  
8 ENGINE = InnoDB;
```

De outro modo, o índice pode ser criado por meio da instrução ALTER TABLE em uma tabela existente:

```
1 ALTER TABLE `mydb`.`Empregado`  
2 ADD PRIMARY KEY (`matricula`);
```

Index

Index é um tipo de índice que permite a entrada de valores duplicados e não deve fazer parte do conjunto de colunas que pertencem à chave primária. Na tabela **Empregado**, ele pode ser inserido na coluna **nome** para facilitar a busca nos sistemas que realizam pesquisa por esse campo.

Considerando que a tabela **Empregado** já existe, para criar um índice *index* chamado **in_nome** para a coluna **nome**, entre com a seguinte instrução SQL:

```
1 ALTER TABLE `mydb`.`Empregado`  
2 ADD INDEX `in_nome` (`nome` ASC);
```

Ao criar um índice do tipo *index*, o projetista de *software* levará em consideração se o campo em questão recebe filtros de consulta por meio de cláusulas WHERE de SQL. Ele também vai considerar o custo-benefício entre a quantidade de pesquisas e o número de inserções/atualizações no sistema. Afinal, colunas indexadas tendem a baixar o desempenho das atualizações porque o SGBD também precisa atualizar as estruturas indexadas.

Unique

Unique é um tipo de índice que não permite que a coluna contenha valores repetidos. Porém, ele não restringe valores nulos. Ele deve ser utilizado em campos que não pertencem à chave primária, mas que exigem valores únicos, como um número de CPF ou um endereço de *e-mail*.

Em uma tabela existente, a instrução a seguir realiza a criação de um índice do tipo *unique* chamado **uq_cpf**. Veja:

```
1 | ALTER TABLE `mydb`.`Empregado`  
2 | ADD UNIQUE INDEX `uq_cpf` (`CPF` ASC);
```

Fulltext

Fulltext é um índice específico para campos de texto que armazenam muitas palavras. A criação de um índice desse tipo é parecida com a criação dos demais tipos, embora o seu comportamento seja diferente, pois o *fulltext* pesquisa por palavras em qualquer lugar do texto da coluna. Observe:

```
1 | ALTER TABLE `mydb`.`Empregado`  
2 | ADD FULLTEXT INDEX `ft_observacao` (`observacao` ASC);
```

Consultas em colunas indexadas no MySQL

Uma vez que os índices são criados, as consultas executadas por meio de instruções SQL fazem as buscas nas colunas indexadas de maneira otimizada. A instrução apresentada na Figura 4 realiza a busca de um empregado por meio de sua matrícula. O resultado indica o uso do índice *primary* (*tabular explain* — coluna **possible_keys**) na ferramenta MySQL Workbench (MYSQL, 2019d).



The screenshot shows a database query tool interface. At the top, there's a tab labeled 'empregado - Table'. Below it, a toolbar contains various icons for file operations, execution, and search. A 'Limit to 1000 rows' dropdown is visible. The query editor shows the following SQL query:

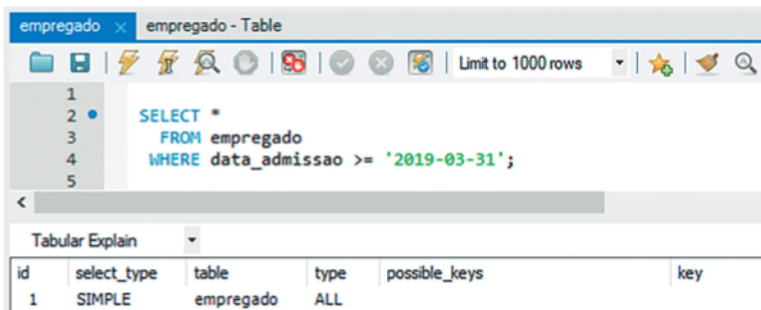
```
1  
2 SELECT *  
3 FROM Empregado  
4 WHERE matricula = '20120446';  
5
```

Below the query editor, a 'Tabular Explain' dropdown is set to 'Tabular Explain'. The resulting explain plan is shown in a table:

id	select_type	table	type	possible_keys	key
1	SIMPLE	Empregado	const	PRIMARY	PRIMARY

Figura 4. Consulta por chave primária.

Por outro lado, a consulta pela data de admissão não utiliza índices, uma vez que esse campo não é indexado na tabela. Dessa forma, todos os registros da tabela são percorridos, tornando a consulta mais lenta em relação à consulta por matrícula. Como você pode ver na Figura 5, nenhum índice foi utilizado, já que falta a indicação na coluna **possible_keys** (*tabular explain*).



The screenshot shows the same database query tool interface. The query editor shows the following SQL query:

```
1  
2 SELECT *  
3 FROM empregado  
4 WHERE data_admissao >= '2019-03-31';  
5
```

The 'Tabular Explain' dropdown is also set to 'Tabular Explain'. The resulting explain plan is shown in a table:

id	select_type	table	type	possible_keys	key
1	SIMPLE	empregado	ALL		

Figura 5. Consulta por campo não indexado.



Referências

MANNINO, M. V. *Projeto, desenvolvimento de aplicações e administração de banco de dados*. 3. ed. Porto Alegre: McGraw-Hill, 2008.

MYSQL. *How MySQL uses indexes*. 2019a. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html>. Acesso em: 15 abr. 2019. 15 abr. 2019.

MYSQL. *MySQL CHECK Constraint Emulation*. 2019b. Disponível em: <http://www.mysql-tutorial.org/mysql-check-constraint/>. Acesso em: 15 abr. 2019.

MYSQL. *Data definition statements*. 2019c. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-data-definition.html>. Acesso em: 15 abr. 2019.

MYSQL. *MySQL Workbench*. 2019d. Disponível em: <https://www.mysql.com/products/workbench/>. Acesso em: 15 abr. 2019.

MYSQL. *CHECK Constraints*. 2019e. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/create-table-check-constraints.html>. Acesso em: 15 abr. 2019.

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de gerenciamento de banco de dados*. 3. ed. Porto Alegre: Penso, 2013.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

