



BANCO DE DADOS

AULA 3

Prof. Lucas Rafael Filipak

CONVERSA INICIAL

Segue a apresentação da aula com a estrutura de conteúdos trabalhados em temas:

1. Linguagem *Structured Query Language* (SQL):

- 1.1. Características e instruções;

- 1.2. Importância do padrão.

2. Operações DDL:

- 2.1 Criando bancos de dados.

3. Criando tabelas;

4. Modificando tabelas:

- 4.1 Apagando tabelas.

Os objetivos desta aula são: contextualizar a linguagem *Structured Query Language* (SQL) compreendendo sua evolução e seus padrões; entender as três categorias de comandos SQL e de que forma elas são aplicadas; estudar os comandos da categoria DDL, que são os comandos utilizados na estrutura do banco de dados; criar e modificar tabelas.

TEMA 1 – LINGUAGEM *STRUCTURED QUERY LANGUAGE* (SQL)

Com o surgimento dos bancos de dados relacionais nos anos 1970, Edgar Frank Codd desenvolveu um modelo de dado relacional chamado *Sequel* ou *linguagem de consulta em inglês estruturado* (*strutucred english query language*). Posteriormente, passou a ser chamada de SQL ou *linguagem de consulta estruturada* (*structured query language*), apresentada oficialmente pela IBM em novembro de 1976, em seu *IBM Journal of R&B*.

A partir de 1986, muitas linguagens de empresas diferentes (concorrentes) permitiram que os programadores acessassem e manipulassem dados relacionais. O padrão SQL foi considerado fácil de aprender e foi muito bem aceito universalmente. Importante deixar claro que o SQL não é uma linguagem de programação, mas sim uma linguagem de consulta estruturada. Nesse mesmo ano, o instituto nacional de padrões (*American National Standard Institute* – ANSI) em conjunto com a organização internacional de padrões (*International Standards Organization* – ISO) definiram um padrão para o SQL, que ficou conhecida como SQL-86 ou SQL-1. Posteriormente, várias revisões

foram feitas no SQL e, a cada revisão, o ano desta era colocado ao lado do nome da linguagem ou a versão (sequencial).

A grande vantagem do SQL foi que os programadores aprendiam uma única linguagem, que com pequenas modificações poderia ser aplicada em uma vasta variedade de sistemas gerenciais de banco de dados (SGBDs). A competitividade entre as empresas que desenvolvem sistemas de bancos de dados é muito grande e, para garantir uma maior satisfação de seus usuários, algumas delas, como a Oracle e a Microsoft, resolveram aderir a um padrão próprio para utilização da linguagem SQL em seus SGBDs.

Os padrões criados pelas empresas têm suas particularidades, mas seguem a base do SQL. Como exemplo foi utilizada a Tabela 1, na qual se pode notar que em determinado SGBD, os tipos de dados recebem nomes diferentes, mas sua utilização é a mesma.

Tabela 1 – Comparação de dados em diferentes SGBD

Descrição Geral	SQL Server	MySQL	Oracle	Exemplo
Bit	bit	bit	(não)	1
Inteiro	int	int	number	25
decimal	decimal	decimal	number	58.63
real	float	float	number	80.62345

Fonte: Kaminski, 2011.

No início, o custo dos sistemas que utilizavam o SQL era muito alto. Com o passar dos anos, os valores foram caindo, e hoje existem alguns sistemas que são *open source* e/ou gratuitos. De acordo com Erick Scudero (2016) o MySQL, PostgreSQL e o MongoDB são exemplos que se enquadram nessa categoria.

Para Damas (2005, p. 1), a linguagem SQL é considerada um padrão dos sistemas de gerenciamento de banco de dados relacional (SGBDR), por isso todos os fabricantes a integram em seus produtos.

1.1 Características e instruções

A linguagem SQL implementa os conceitos definidos no modelo relacional. Pode-se ver em Damas (2005, p. 3) que, com a linguagem SQL é possível:

- Criar, alterar e remover todos os elementos de um banco de dados, como tabelas, *views*, índices etc.;
- Inserir, alterar e apagar dados;
- Consultar o banco de dados;
- Controlar o acesso dos usuários ao banco de dados e as operações a que cada um deles pode ter acesso;
- Obter a garantia da consistência e integridade dos dados.

Em 1982 foi lançada a primeira versão padronizada da linguagem SQL, que veio ganhando melhorias de acordo com sua evolução. A padronização do SQL apresenta características que, conforme Puga (2013, p. 170), merecem destaque:

- As instruções padronizadas seguem a mesma nomenclatura e formato para diferentes tipos de SGDB, respeitando as particularidades de cada um;
- A migração de um SGDB para outro não requer grandes mudanças;
- Quando ocorre a migração de um SGDB, a adaptação dos profissionais é facilitada, com redução de tempo e de custos para treinamento, pois as instruções possuem nomes e funcionalidades iguais;
- Portabilidade entre as plataformas.

Para Puga (2013, p. 170), o SQL apresenta três categorias de instruções:

- DDL → *Data Definition Language*: utilizada para definição e descrição dos dados.

Quadro 1 – Categorias de instrução DDL

Finalidade	Definição e manutenção das estruturas do banco de dados, tais como a criação do próprio banco de dados e das tabelas que o compõem, além das relações entre as tabelas e os objetos do banco de dados.
Instruções	
Create	Criação de estruturas de objetos do banco de dados.
Alter	Alteração da estrutura de objetos do banco de dados.
Drop	Eliminação das estruturas de objetos do banco de dados.
Truncate	Exclusão física de linhas de tabelas.
Rename	Renomeação de objetos do banco de dados.
Comment	Inclusão de comentários aos objetos do banco de dados.

Fonte: Puga, 2013, p. 170.

- DML → *Data Manipulation Language*: utilizada para a manipulação dos dados. Na DML, existe uma subcategoria de instruções, chamada DCL (*Data Control Language*), que é utilizada para controlar o acesso aos dados (Puga, 2013, p. 170).

Quadro 2 – Categorias de instrução DML

Finalidade	Consultas, inserções, exclusões e alterações em um ou mais registros, de uma ou mais tabelas, de maneira simultânea.
Instruções	
Insert	Inserção de dados.
Update	Alteração de dados.
Delete	Exclusão de dados.
Select	Consulta de dados.
Merge	Combinação das instruções insert, update e delete.

Fonte: Puga, 2013, p. 170.

- DCL → *Data Control Language*: utilizada para controle de segurança do banco de dados, atribuindo e retirando permissões (Puga, 2013, p. 170).

Quadro 3 – Subcategoria de instrução DCL

Finalidade	Controle dos privilégios de usuários, de forma que o administrador do banco de dados possa determinar o nível de acesso de um usuário aos objetos do banco de dados, concedendo privilégios ou retirando esse acesso e revogando os privilégios.
Instruções	
Grant	Atribuição de privilégios aos usuários do banco de dados.
Revoke	Revogação de privilégios dos usuários do banco de dados.

Fonte: Puga, 2013, p. 171.

- TCL → *Transact Control Language*: utilizada para controle de transações (Puga, 2013, p. 170).

Quadro 4 – Categorias de instrução TCL

Finalidade	Controle de transações, consideradas o conjunto de uma ou mais operações DML realizadas no banco de dados.
Instruções	
Commit	Confirmação das manipulações.
Rollback	Desistência das manipulações.
Savepoint	Criação de pontos para o controle das transações.

Fonte: Puga, 2013, p. 171.

A criação dos grupos de comandos levou em consideração a finalidade de cada comando. Os comandos que mexem na estrutura da tabela ficaram na categoria DDL; os que operam com os dados ficaram na categoria DML, os comandos que atribuem e retiram permissões dos usuários (nos dados) ficaram na categoria DCL, e os comandos que fazem transações ficaram na categoria TCL.

1.2 Importância do padrão

A linguagem SQL começou a se tornar popular e se tornou um grande sucesso, obrigando empresas como ANSI e ISO a padronizar a linguagem SQL. Em 1986 foi criado um padrão ANSI e, em 1987, foi criado um padrão ISO. Desde então, várias versões do padrão SQL foram criadas, acrescentando novos comandos e funcionalidades em cada nova versão. Na Tabela 2 há alguns detalhes sobre as versões do padrão ANSI.

Tabela 2 – Versões do padrão ANSI

SQL:86	Primeira versão da linguagem, lançada em 1986, consiste basicamente na versão inicial da linguagem criada pela IBM.
SQL:92	Lançada em 1992, inclui novos recursos tais como tabelas temporárias, novas funções, expressões nomeadas, valores únicos, instrução <i>case</i> etc.
SQL:1999	Lançada em 1999, foi a versão que teve mais recursos novos significativos, entre eles: a implementação de expressões regulares, recursos de orientação a objetos, <i>queries</i> recursivas, <i>triggers</i> , novos tipos de dados (boolean, LOB, <i>array</i> e outros), novos predicados etc.
SQL:2003	Lançada em 2003, inclui suporte básico ao padrão XML, sequências padronizadas, instrução <i>MERGE</i> , colunas com valores autoincrementais etc.
SQL:2006	Lançada em 2006, não inclui mudanças significativas para as funções e comandos SQL. Contempla basicamente a interação entre SQL e XML

Fonte: Prado, 2015.

Os padrões são “dialetos” que os principais fabricantes de SGBDs implementam em seus bancos de dados. Atualmente, além dos seus dialetos, os bancos também implementam o padrão ANSI. A empresa Oracle tem o seu padrão, comumente chamado de *padrão Oracle*, que nada mais é que o dialeto SQL da Oracle. Os dialetos geralmente surgem quando o fabricante precisa implementar algum recurso no SGBD, mas esse recurso ainda não foi validado/criado no padrão ANSI.

A Figura 1 e a Figura 2 representam o mesmo comando sendo executado no padrão Oracle (Figura 1) e no padrão ANSI (Figura 2). Não se atente a

entender os comandos, mas sim a comparar e reparar que existem diferenças de sintaxes entre eles.

Figura 1 – Instrução SQL com ligação representando LEFT JOIN utilizando o dialeto Oracle

```
SELECT    e.first_name || e.last_name NAME,  
          D.DEPARTMENT_NAME  
FROM      HR.EMPLOYEES E,  
          HR.DEPARTMENTS D  
WHERE     E.DEPARTMENT_ID = D.DEPARTMENT_ID (+);
```

Fonte: Prado, 2015.

Figura 2 – Instrução SQL com ligação representando LEFT JOIN utilizando o padrão ANSI

```
SELECT    E.FIRST_NAME || E.LAST_NAME NAME,  
          D.DEPARTMENT_NAME  
FROM      HR.EMPLOYEES E  
LEFT JOIN HR.DEPARTMENTS D  
ON        E.DEPARTMENT_ID = D.DEPARTMENT_ID;
```

Fonte: Prado, 2015.

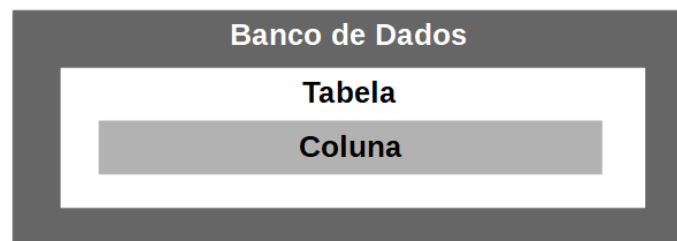
Para finalizar a explicação sobre padrões, Prado (2015) afirma que, de um modo geral, não há diferença de desempenho entre os dois padrões.

TEMA 2 – OPERAÇÕES DDL

A *Data Definition Language* (DDL) é a primeira parte da criação de um banco de dados, pois é nela que estão os comandos para criação do banco de dados, das tabelas e a definição das relações.

Para visualizar melhor, a Figura 3 traz os componentes e a sua hierarquia dentro do banco de dados. Lembre-se também de que um sistema de gerenciamento de banco de dados pode possuir mais de um banco de dados. Os bancos de dados podem ter uma ou mais tabelas, e as tabelas podem possuir uma ou mais colunas.

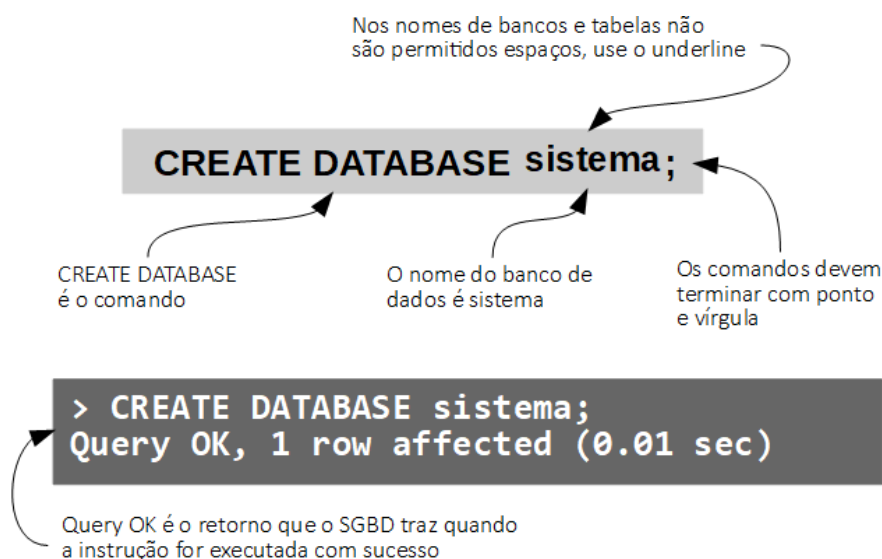
Figura 3 – Hierarquia dos componentes do banco de dados



2.1 Criando o banco de dados

O primeiro passo consiste na criação do banco de dados. A sintaxe para a criação é simples e não é necessário que seja digitada em letras maiúsculas. A sintaxe é *CREATE DATABASE nome_banco*. Portanto, “CREATE DATABASE” é o comando utilizado para criar o banco de dados. A Figura 4 traz um exemplo de execução desse comando no MYSQL. Outros SGBDs podem trazer outras mensagens, pois cada um possui diferentes mensagens de execução.

Figura 4 – Exemplo do uso do comando CREATE DATABASE



Deve-se prestar atenção ao nome do banco, pois este não pode ter espaços ou caracteres especiais, e deve ser o mais descritivo possível. Outra situação importante é que dentro de um SGBD pode existir mais de um banco de dados, então o nome deste deve ser único. Para visualizar os bancos de dados que já foram criados, utiliza-se o comando *SHOW DATABASES*.

Figura 5 – Exemplo do uso do comando SHOW DATABASES

```
> SHOW DATABASES;

+-----+
| Databases |
+-----+
| base_alunos |
| base_prefeitura |
| conjunto |
| sistema |
| sistema_bkp |
+-----+

5 rows in set (0.00 sec)
```

A Figura 5 apresenta um exemplo da execução do comando, mostrando o nome de cinco bancos que já foram criados. Note que o banco sistema, que criamos com a execução do comando da Figura 4, está na lista. O próximo passo depois da criação do banco de dados é informar para o SGBD o banco de dados que será utilizado. A partir do comando `USE [nome_banco]`, Figura 6, todos os comandos SQLs serão aplicados no banco de dados selecionado.

Figura 6 – Exemplo do comando USE

USE sistema;

A partir da instrução executada, todos os comandos serão executados na base de dados selecionada (sistema).

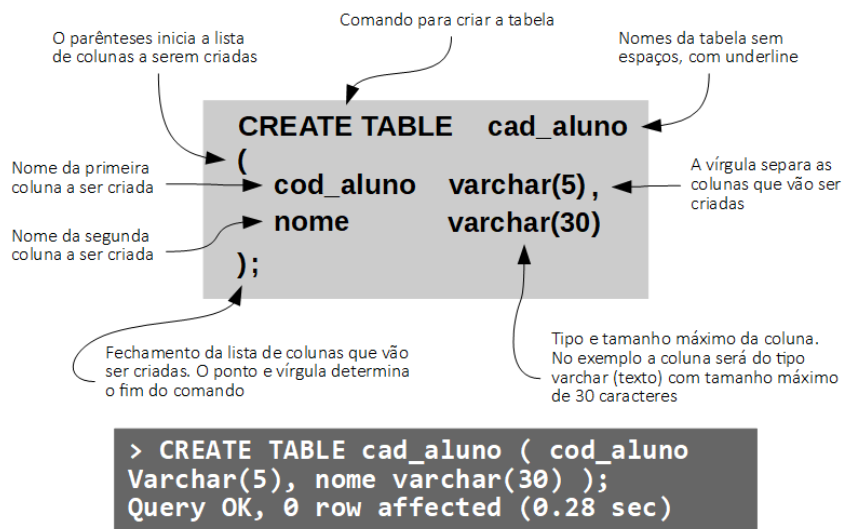
```
> USE sistema;
Database changed
```

Database changed é a confirmação que a base foi alterada

TEMA 3 – CRIANDO AS TABELAS

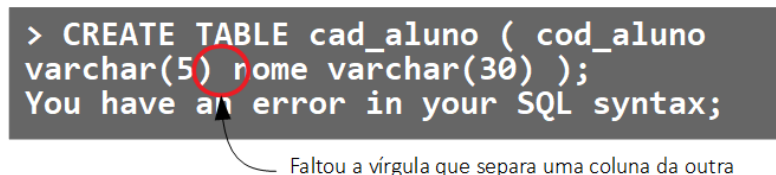
Um banco de dados pode ser formado por uma ou mais tabelas. Cada tabela também é nomeada com uma identificação única e deve conter colunas agrupadas pelo seu tipo. Para exemplificar, o banco de dados possui uma tabela chamada *alunos* (que vai conter os dados dos alunos), outra tabela chamada *professores* (com os dados dos professores), e assim por diante. O comando para a criação de uma nova tabela é o *CREATE TABLE*.

Figura 7 – Exemplo do comando CREATE TABLE



Se a sintaxe estiver correta – como a exibida na Figura 7 – aparecerá uma mensagem positiva de execução. No MySQL, a mensagem é **Query OK**. Mas se algo na sintaxe estiver errado, a mensagem de execução é negativa. No MySQL a mensagem de erro é “*You have an error in your SQL syntax*”.

Figura 8 – Exemplo de erro na sintaxe da query



Na Figura 8, o comando para a construção da tabela não foi executado, pois faltou uma vírgula entre a criação da coluna `cod_aluno` e a coluna `nome`. Observe que o programa retorna uma mensagem de erro.

Depois de criadas as tabelas (pelo menos uma), é possível visualizar as tabelas existentes no banco de dados. O comando utilizado é o *SHOW TABLES*, como mostra a Figura 9.

Figura 9 – Exemplo do comando SHOW TABLES

```
> SHOW TABLES;

+-----+
| Tables_in_sistema |
+-----+
| cad_aluno          |
+-----+

1 row in set (0.00 sec)
```

Note que nos comandos de *CREATE TABLE* e *SHOW TABLES* não é preciso informar o nome do banco de dados, pois, como o comando *USE sistema* foi executado anteriormente, o gerenciador do banco sabe que todas as consultas serão realizadas no banco chamado *sistema*. Entretanto, caso esse sistema não tenha sido executado, é possível informar o nome do banco precedendo os objetos, ou então executar o comando *USE [nome_banco]* a qualquer momento.

Para visualizar a estrutura de uma tabela, utiliza-se o comando *DESCRIBE nome_tabela*. O comando vai mostrar a estrutura da tabela, com o nome das colunas (*Field*), os tipos de dados e tamanhos (*Types*). As outras colunas serão explicadas posteriormente.

Figura 10 – Exemplo do comando DESCRIBE

```
> DESCRIBE cad_aluno;
```

Field	Types	Null	Key	Default	Extra
cod_aluno	varchar(5)	YES		NULL	
nome	varchar(30)	YES		NULL	

2 rows in set (0.01 sec)

O comando da Figura 10 retornou duas linhas (2 rows), pois essa tabela é composta de duas colunas, *cod_aluno* e *nome*, conforme comandos que executamos na Figura 3.

A restrição NULL, que também é chamada de *constraint*, representa que a coluna pode ser vazia. Essa definição ocorre automaticamente na criação da tabela. Caso seja obrigatório o preenchimento de dados nessa coluna, é preciso modificar a restrição da coluna para NOT NULL. Esse procedimento pode ser realizado no momento em que a tabela está sendo criada ou depois, por meio de uma alteração da tabela – como veremos no tema 4 deste material.

Figura 11 – Exemplo do uso do comando NOT NULL

O comando NOT NULL informa que o preenchimento da coluna é obrigatório

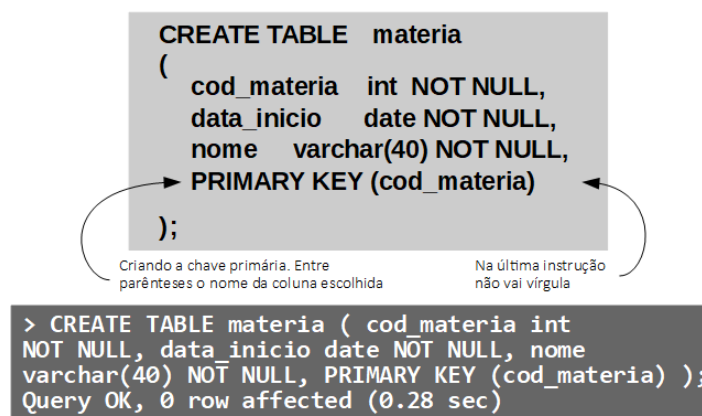
```
CREATE TABLE cad_aluno
(
  cod_aluno varchar(5) NOT NULL,
  nome      varchar(30) NOT NULL
);
```

A tabela criada na Figura 11 tem as suas duas colunas definidas como NOT NULL; em outras palavras, foi gerada uma restrição ou *constraint* nessas colunas obrigando o preenchimento de valores. Mas o que é estes “estarem preenchidos”? Até agora foi vista apenas a estrutura da tabela, sem inserir nenhum dado (conteúdo). Em breve, serão estudados comandos que realizam a inserção dos dados em uma tabela, e como essa restrição influenciará nesse procedimento.

A definição de uma chave primária é uma condição imposta em uma ou mais colunas de uma tabela, de modo que essa coluna não tenha valores nulos – ou seja, sempre terá um valor único, portanto este não se repete em uma mesma tabela, e é utilizado como referência para gerar relacionamentos com outras tabelas do banco de dados. Como exemplo, pode-se utilizar a tabela *cad_pessoas* e como chave primária a coluna *CPF*, pois o CPF é um dado único, o que torna o registro igualmente único. Algumas curiosidades sobre chave primária:

- A coluna que recebe a chave primária não pode ser NULL;
- Cada tabela tem apenas uma chave primária;
- Tipos de dados mais utilizados para as chaves são INT e VARCHAR;
- O valor da chave primária é atribuído no momento que o registro é criado.

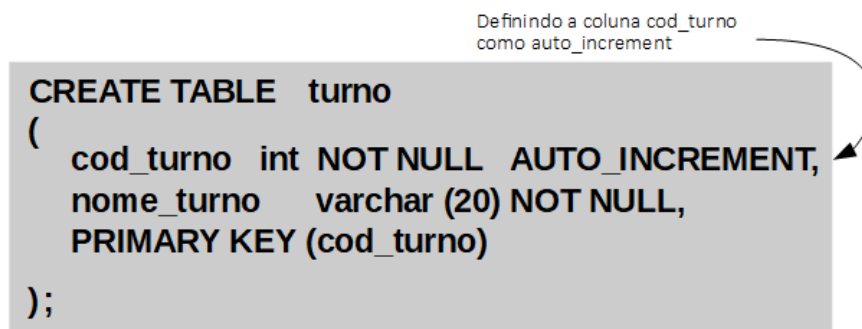
Figura 12 – Exemplo do uso do comando PRIMARY KEY



Na Figura 12, a coluna *cod_materia* foi escolhida para ser a chave primária da tabela. Com essa restrição, quando inseridos os dados na tabela não será possível ter duas matérias com o mesmo código, e toda matéria deve ter um código pelo fato da coluna chave de uma tabela ter a condição NOT NULL.

Algumas tabelas podem não possuir uma coluna que possa ser utilizada como chave primária (única). Nesse caso, é necessário criar na tabela uma coluna do tipo inteiro e atribuir a ela a condição de `auto_increment`.

Figura 13 – Exemplo do uso do comando `AUTO_INCREMENT`



Definindo a coluna `cod_turno` como `auto_increment`

```
CREATE TABLE turno
(
  cod_turno int NOT NULL AUTO_INCREMENT,
  nome_turno varchar (20) NOT NULL,
  PRIMARY KEY (cod_turno)
);
```

O `AUTO_INCREMENT`, conforme exibido na Figura 13, informa o SGDB que ele deve, a cada novo registro inserido na tabela, incrementar automaticamente e de forma sequencial a coluna `cod_turno`. No primeiro registro inserido nessa tabela, a coluna `cod_turno` recebe o valor 1, no segundo, ela recebe o valor 2, e assim por diante, garantindo assim que não haverá valores duplicados. O valor inicial do `AUTO_INCREMENT` pode ser configurado (ver Figura 18).

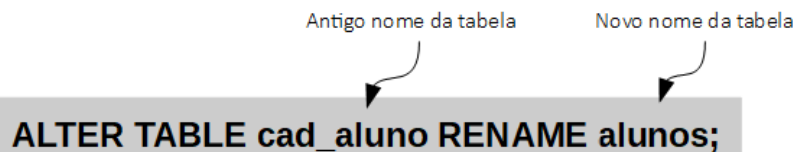
TEMA 4 – MODIFICANDO TABELAS

O planejamento de um banco de dados nem sempre é uma tarefa tão fácil, e modificações na estrutura da tabela são comuns. Quando necessário realizar alterações nas definições da estrutura do banco de dados, podemos utilizar o comando `ALTER TABLE`, o qual permite:

- Renomear uma tabela;
- Alterar tipo e tamanho de uma coluna;
- Adicionar ou remover colunas na tabela.

No que se refere a renomear uma tabela, a Figura 14 representa a alteração no nome da tabela, passando de `cad_aluno` para `alunos`.

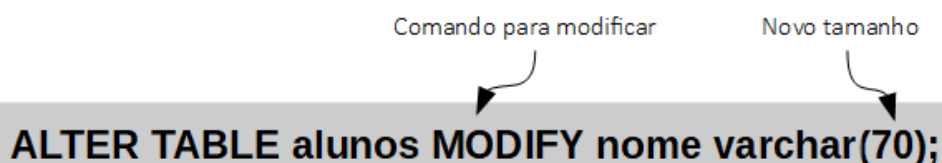
Figura 14 – Exemplo do uso do comando RENAME



```
ALTER TABLE cad_aluno RENAME alunos;
```

Para alterar o tamanho de uma coluna, utiliza-se o parâmetro *MODIFY*. Na Figura 15, a coluna *nome* passa do tamanho 30 para 70. Não é preciso informar o tamanho antigo, somente o novo tamanho.

Figura 15 – Exemplo do uso do comando MODIFY

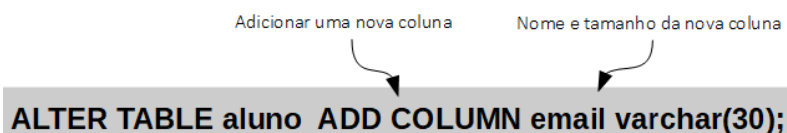


```
ALTER TABLE alunos MODIFY nome varchar(70);
```

Observe que, ainda que a alteração seja em uma coluna, usamos o comando **ALTER TABLE** e, por isso, precisamos informar o nome da tabela em que se encontra a coluna *nome*. Por esse motivo, todas as modificações realizadas em uma coluna serão realizadas por meio do procedimento de modificação da tabela, pois, ao alterar uma coluna, consequentemente se modifica uma tabela. Assim, o comando deve seguir o seguinte formato: **ALTER TABLE [nome_tabela] MODIFY [nome da coluna] [tipo]**.

Para adicionar novas colunas em uma tabela já existente, utiliza-se o parâmetro *ADD COLUMN*, informando o nome da nova coluna junto com o tipo e tamanho. Na Figura 16, foi adicionada a coluna *email* na tabela *aluno*. Pode-se utilizar de maneira simplificada: **ALTER TABLE aluno ADD email varchar(30)**.

Figura 16 – Exemplo do uso do comando ADD COLUMN



```
ALTER TABLE aluno ADD COLUMN email varchar(30);
```

Como pode ser visto na Figura 17, o parâmetro *DROP COLUMN* é utilizado para apagar uma coluna de uma tabela. Não é preciso informar o tipo e o tamanho da coluna. Pode-se utilizar de maneira simplificada o comando: **ALTER TABLE [nome_tabela] DROP [nome_coluna]**.

Figura 17 – Exemplo do uso do comando DROP COLUMN

Apagar uma coluna Nome da coluna a ser excluída

```
ALTER TABLE aluno DROP COLUMN email;
```

Como visto anteriormente, o parâmetro `auto_increment` permite que um número único seja gerado quando um novo registro é inserido em uma tabela. Ao executar o comando da Figura 18, os registros iniciarão os incrementos a partir do número 15, portanto, o primeiro registro salvo na tabela receberá o valor 15.

Figura 18 – Exemplo do uso do comando AUTO_INCREMENT

```
ALTER TABLE alunos AUTO_INCREMENT = 15 ;
```

4.1 Apagando tabelas

Conforme o comando da Figura 19, também é possível apagar as tabelas do banco de dados que não serão utilizadas. O comando `DROP TABLE [nome_tabela]` apaga a tabela e também todos os objetos relacionados a essa tabela, incluindo dados, índices, restrições de tabela, *triggers* (gatilhos) e permissões (alguns objetos citados serão estudados posteriormente).

Figura 19 – Exemplo do uso do comando DROP TABLE

DROP TABLE é o comando O nome da tabela que vai ser apagada

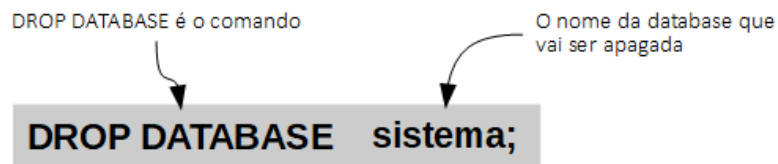
```
DROP TABLE cad_alunos;
```

```
> DROP TABLE cad_alunos;  
Query OK, 0 rows affected (0.22 sec)
```

Após a utilização desse comando, a tabela não poderá ser mais acessada ou recuperada. Caso você queira apenas apagar os dados armazenados na tabela e manter a estrutura da tabela no banco de dados, pode ser utilizado o comando `DELETE` (que será visto em breve).

Assim como as tabelas, podemos observar na Figura 20 que o banco de dados também pode ser apagado. Para isso, deve ser utilizado o comando *DROP DATABASE [nome_banco]*.

Figura 20 – Exemplo do uso do comando DROP DATABASE



O diagrama mostra o comando SQL `DROP DATABASE sistema;` em um fundo cinza. Duas setas explicativas apontam para partes do comando: uma seta curva aponta de "DROP DATABASE é o comando" para a palavra "DROP DATABASE", e outra seta curva aponta de "O nome da database que vai ser apagada" para a palavra "sistema;".

```
DROP DATABASE sistema;
```

FINALIZANDO

Nesta aula apresentamos, sobre a linguagem SQL: suas origens e principais modificações históricas. Abordamos sua padronização e sua importância, e que os comandos SQLs são classificados em três categorias que variam de acordo com a função dos comandos. Estudamos sobre os comandos da categoria DDL, que são responsáveis por fazer a estrutura do banco de dados: criação e exclusão do banco, criação, modificação e exclusão das tabelas etc.

REFERÊNCIAS

DAMAS, L. **SQL – Structured Query Language**. 6. ed. Rio de Janeiro: FCA, 2005.

KAMINSKI, M. Instruções SQL no SQL Server, Oracle e MySQL. **Revista SQL Magazine**, edição 90, 2011. Disponível em: <<https://www.devmedia.com.br/instrucoes-sql-no-sql-server-oracle-e-mysql/22005>>. Acesso em: 6 set. 2019.

LIMA, A. G. de A. **Padrão SQL e sua evolução**. Disponível em: <<http://www.ic.unicamp.br/~geovane/mo410-091/Ch05-PadiaoSQL-art.pdf>> Acesso em: 6 set. 2019.

PRADO, F. **SQL padrão ANSI X padrão Oracle**: qual é mais rápido? Blog do DBA/Instrutor Fábio Prado, 12 mar. 2015. Disponível em: <<http://www.fabioprado.net/2012/05/sql-padrao-ansi-x-padrao-oracle.html>>. Acesso em: 6 set. 2019.

PUGA, S. **Banco de dados**: implementação em SQL, PL/SQL, Oracle 11g. São Paulo: Pearson Education do Brasil, 2013.

SCUDERO, E. TOP 10 principais SGBDs do mercado global! **Becode**, 2016. Disponível em: <<https://becode.com.br/principais-sgbds/>>. Acesso em: 6 set. 2019.