



AULA 2

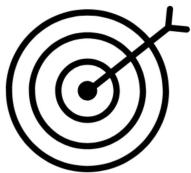
Criação de banco de dados relacionais: do modelo conceitual ao físico

Que história é essa?

Esta disciplina tem por objetivo ensinar os conceitos básicos de bancos de dados, com ênfase na teoria e prática. Aqui, serão vistos conceitos introdutórios para o desenvolvimento de sistemas de bancos de dados, incluindo os modelos de dados e a linguagem SQL, por meio do projeto de novos bancos de dados que respondem aos requisitos impostos pelas aplicações.

Serão abordados os seguintes assuntos: transformação do modelo conceitual em modelo lógico; transformação do modelo lógico em modelo físico; introdução à linguagem SQL; criação e manutenção de tabelas; e restrições de integridades no banco de dados.

Anteriormente, foi abordada a identificação de entidades, atributos e relacionamentos de uma aplicação que gerencia a inscrição de alunos em uma plataforma de cursos autoinstrucionais. Com essas informações, você pode criar o dicionário de dados e o DER.



Objetivos de aprendizag em da aula

Ao final desta aula, você irá:

- Diferenciar modelo conceitual de modelo lógico e físico.
- Construir comandos básicos de DDL (CREATE, ALTER e DROP) em linguagem SQL.
- Criar um banco de dados relacional.
- Consultar as tabelas do dicionário de dados utilizando a linguagem SQL (SELECT).

Transformação do modelo conceitual em modelo lógico

Durante o projeto de banco de dados, a primeira transformação que o modelo conceitual sofre diz respeito à conversão de seus elementos em elementos de um modelo/esquema lógico que depende do modelo adotado pelo SGBD escolhido. Nesta disciplina, será adotado o modelo relacional referente aos **SGBDs relacionais**, com **os dados sendo armazenados em tabelas**, também conhecidas como **relações**. Os detalhes quanto ao armazenamento interno dos dados não fazem parte do modelo lógico.

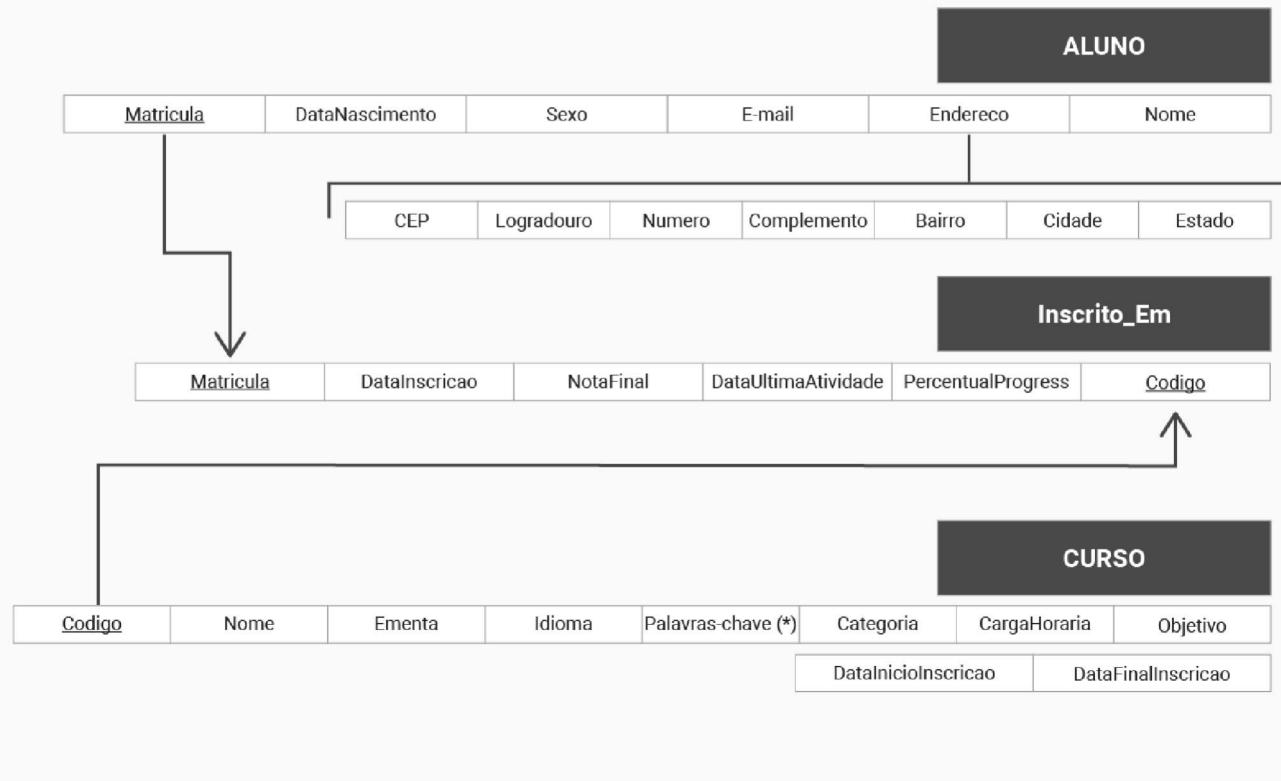
O mapeamento de um esquema conceitual, definido pelo MER, em um esquema lógico do modelo relacional envolve transformar cada elemento de um esquema de entidades e relacionamentos (entidades, relacionamentos e atributos) em elementos equivalentes da modelagem relacional (tabelas, atributos e restrições de integridade). A tabela a seguir apresenta as principais transformações desta etapa:

Modelo de entidades e relacionamentos (MER)	Modelo relacional
Entidade	Tabela.
Atributo de entidade	Atributo (ou coluna) de tabela.
Atributo identificador	Chave primária (PK).
Relacionamento NxN	Tabela com os atributos identificadores das entidades envolvidas e os atributos do relacionamento, se houver. Os atributos identificadores irão compor a PK e serão também chaves estrangeiras para as tabelas participantes.
Relacionamento 1xN	O atributo identificador da entidade com menor cardinalidade máxima ("1") e os atributos do relacionamento, se houver, devem ser adicionados como atributos na tabela referente à entidade com maior cardinalidade no relacionamento ("N"). O atributo identificador será uma chave estrangeira.
Relacionamento 1x1	Duas alternativas principais: (1) fusão: tabela que inclui todos os atributos de todas as entidades, assim como os atributos do relacionamento, se houver; ou (2) tratar como relacionamentos 1xN.

Com base nas tabelas do dicionário de dados construídas anteriormente, pode-se iniciar essa etapa de transformação.

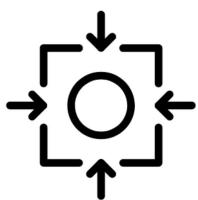
Inicialmente, cada entidade foi transformada em uma tabela com os respectivos atributos. Assim, foram criadas as tabelas ALUNO e CURSO. Depois que todas as entidades foram devidamente mapeadas, serão analisados os relacionamentos presentes. Como nesta aplicação tem-se somente um relacionamento **NxN**, uma nova tabela será criada: **Inscrito_Em**. Nesse caso, **a chave primária da tabela que representa o relacionamento é definida pela combinação dos identificadores das duas entidades (ALUNO e CURSO) que participam desse relacionamento**. No modelo relacional, cada parte dessa chave primária será uma chave estrangeira. Essa alternativa é típica de relacionamentos NxN. Assim, tem-se o seguinte esquema relacional resultante da transformação, em que os atributos identificadores estão sublinhados:

Esquema relacional EAD



Noções básicas de normalização

Além das transformações, a normalização das tabelas pode ser aplicada para garantir que o modelo não apresente redundâncias e potenciais problemas de manutenção dos dados.



Mantendo o foco

Redundância de dados: mesma informação armazenada em mais de uma tabela ou atributo. A presença de redundância de dados não controlada pode levar a vários problemas, incluindo:

- **inconsistência de dados:** se os dados redundantes não forem atualizados corretamente, podem surgir inconsistências, em que uma cópia dos dados é alterada, mas a outra não;
- **desperdício de espaço de armazenamento:** armazenar a mesma informação em vários lugares consome mais espaço do que o necessário;
- **dificuldades na manutenção:** modificar informações redundantes em vários locais pode levar a erros e dificultar a manutenção do banco de dados.

No entanto, em alguns casos, uma certa quantidade de redundância controlada por replicação pode ser aceitável para otimizar o desempenho das consultas. O equilíbrio entre normalização e desempenho depende das necessidades específicas do sistema e dos requisitos de consulta.

Na literatura, as formas normais mais utilizadas são: primeira, segunda e terceira. No entanto, para entender as formas normais, primeiro é preciso explicar o que são as **dependências funcionais**.

A dependência funcional ocorre quando o valor de uma coluna A depende do valor de uma coluna B da mesma tabela. As dependências funcionais podem ser de três tipos: parciais, totais e transitivas. A dependência funcional total ocorre quando a tabela tem duas colunas como identificadores e as demais colunas dependem da combinação de ambas. Já na dependência parcial, existem colunas que dependem somente de um dos identificadores. A dependência transitiva ocorre quando uma coluna depende de outra que não é identificadora.

A primeira forma normal (**1FN**) indica que uma tabela só deve ter colunas com valores atômicos e, pelo menos, um identificador único. Observe que as tabelas da imagem anterior têm os seguintes identificadores:

Tabela	Identificador	Tipo de identificador
ALUNO	Matricula	Simples
CURSO	Codigo	Simples
Inscrito_Em	Matricula, Codigo	Composto

Quanto a valores atômicos, existem dois casos a serem tratados: o atributo composto **Endereco** da tabela ALUNO e o atributo multivalorado **Palavras-chave** da tabela CURSO.

No caso do atributo composto **Endereco**, este deverá ser substituído pelos atributos que o compõem: **CEP**, **Logradouro**, **Bairro**, **Cidade**, **Estado**, **Numero** e **Complemento**.

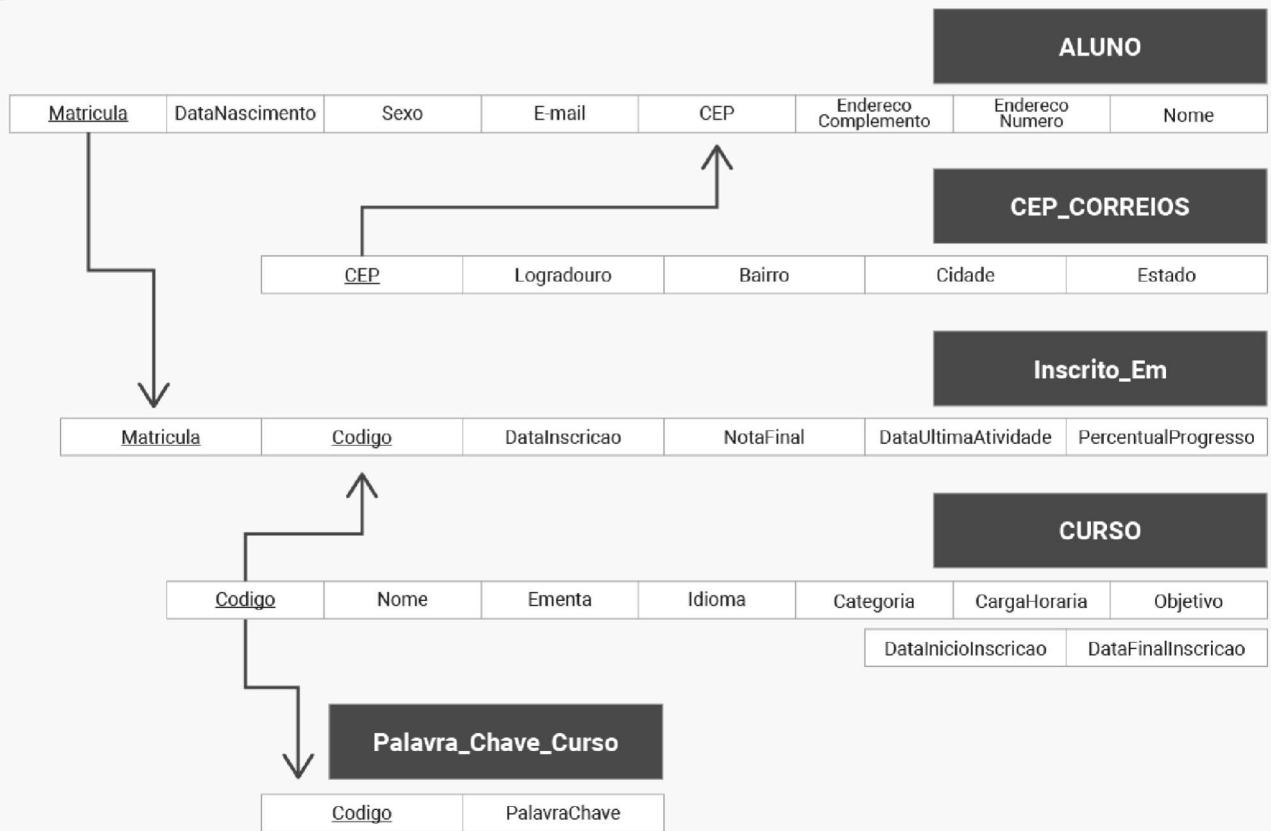
No caso do atributo multivalorado **Palavras-chave**, duas alternativas podem ser consideradas: (1) quando o número máximo de ocorrências **N** estiver definido, podem ser criados **N** atributos, como **Palavra_chave_1** até **Palavra_chave_N**; ou (2) criar uma tabela com dois atributos: **Codigo** do CURSO e **Palavra_Chave**. Neste exemplo, será adotada a segunda opção, criando a tabela Palavra_Chave_Curso.

A segunda forma normal (**2FN**) indica que a tabela deve estar na 1FN e também não pode ter dependência funcional parcial. Considerando o “Esquema relacional EAD”, a única tabela que tem identificador composto é **Inscrito_Em** e todos os atributos desta dependem da combinação das colunas que compõem o identificador. No entanto, suponha um outro exemplo, em que existiria uma tabela Item_Venda com o identificador composto por **CodigoProduto** e **NumeroNotaFiscal** e os atributos **Quantidade**, **ValorTotal** e **ValorProduto**. Essa tabela não estaria na **2FN** se o atributo **ValorProduto** depende somente de **CodigoProduto**.

Na terceira forma normal (**3FN**), a tabela deve estar na 2FN e não ter dependências funcionais transitivas. Analisando cada atributo das tabelas ALUNO, CURSO, Inscrito_Em e Palavra_Chave_Curso, e também recorrendo às informações que constam no dicionário de dados, é possível identificar a dependência funcional transitiva dos atributos **Logradouro**, **Bairro**, **Municipio** e **Estado** em relação ao atributo **CEP** na tabela ALUNO. Nesses casos, é recomendada a criação de uma nova tabela e a transferência desses atributos para ela. Assim, uma quinta tabela passa a fazer parte desse esquema lógico do banco de dados da aplicação

de EAD depois das transformações necessárias para a normalização de dados, como apresentado a seguir:

Esquema relacional EAD NORMALIZADO



Observe que a tabela CEP_CORREIOS ainda poderia ser submetida a uma nova transformação, uma vez que existe uma dependência funcional transitiva entre **Município** e **Estado**.

Ao final do projeto lógico, há um esquema lógico associado a algum modelo lógico conhecido, que tenha a especificação completa exigida por um modelo de dados, a saber, estrutura de organização e linguagens de manuseio dos dados. Nesse ponto, o desenvolvedor, em conjunto com o DBA, já pode escolher um **SGBD específico para implementar o projeto físico correspondente ao esquema lógico definido**. No caso de SGBDs relacionais, será utilizada a linguagem **SQL** com as sintaxes específicas de criação de tabelas e restrições de integridade.

A escolha do SGBD relacional que será utilizado é crucial para atender aos requisitos funcionais e não funcionais da aplicação. Os mecanismos de banco de dados que operam na arquitetura cliente/servidor focam a implementação de um repositório compartilhado de dados corporativos com recursos que garantam escalabilidade, concorrência de acesso, centralização de dados e controle de acesso. Servidores de banco de dados requerem esforço de administração e máquinas dedicadas com suporte humano especializado. Porém, para funcionalidades específicas, SGBDs como Oracle, SQL Server, PostgreSQL e MySQL podem apresentar diferenças na implementação.

Depois da criação dos objetos de acordo com o esquema lógico modelado (como o que será feito nesta aula), o DBA ainda precisará se dedicar a outras atividades que são essenciais para garantir um banco de dados com bom desempenho:

- avaliar a criação de índices para otimização de consulta;
- avaliar se o volume e o padrão de acesso aos dados poderiam ser beneficiados com estratégias de particionamento de dados;
- ajustar a configuração de parâmetros de memória e disco do SBGD;
- aplicar critérios de privilégios mínimos na concessão de acessos aos dados, preferencialmente pela criação de perfis de acesso;
- configurar a política de *backup* do banco de dados e testar a recuperação dos dados para garantir a integridade em caso de falhas e desastres;
- avaliar a escalabilidade do servidor por meio de testes de carga e implementar, caso seja requisito da aplicação, estratégias de balanceamento de carga e replicação.

A implantação da aplicação em produção irá requerer que o DBA implemente ferramentas de monitoramento para acompanhar o desempenho e realizar otimizações com base nos padrões de uso e requisitos do sistema.

A linguagem SQL tem um conjunto de operações que podem ser agrupadas em cinco subconjuntos: DML, DDL, DQL, DCL e TCL.

Nesta disciplina, serão abordados os primeiros três subconjuntos. Clique nos itens para saber mais:

Linguagem de manipulação de dados, ou DML (*data manipulation language*) ↓

Operações para inserir (*INSERT*), atualizar (*UPDATE*) e apagar (*DELETE*) dados em tabelas.

Linguagem de definição de dados, ou DDL (*data definition language*) ↓

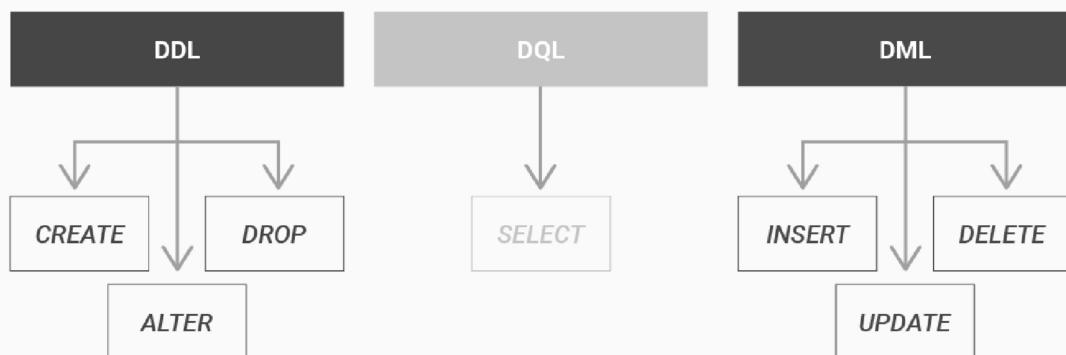
Operações para criar (*CREATE*), apagar (*DROP*) e alterar (*ALTER*) objetos do banco de dados, como tabelas, restrições de integridade, índices, visões, etc. Essas operações atualizam o dicionário de dados do SGBD.

Linguagem de consultas de dados, ou DQL (*data query language*) ↓

Operação de consulta (*SELECT*) de dados em tabelas e visões do banco de dados e também do dicionário de dados.

Para visualizar esses três subgrupos, acompanhe:

SQL



Não serão abordados DCL e TCL. A linguagem de controle de dados, ou DCL (*data control language*), contempla **operações de autorização de usuários** para manipulação de dados e objetos dentro do SBGD. Alguns comandos são *GRANT* (concessão de privilégios) e *REVOKE*.

(revogação de privilégios). Por último, a linguagem de controle de transações, ou TCL (*transaction control language*), agrupa as operações para executar transações usadas para garantir a consistência do banco de dados.

Considerando a DDL, pode-se apresentar alguns exemplos da sintaxe genérica:

DDL

Exemplos da sintaxe

(1) CREATE TABLE nome_da_tabela (coluna1 tipo_de_dado1 [restricoes1], coluna2 tipo_de_dado2 [restricoes2]);	(2) CREATE TABLE nome_da_tabela (coluna1 tipo_de_dado1 [restricoes1], coluna2 tipo_de_dado2 [restricoes2], ..., CONSTRAINT nome_da_restricao PRIMARY KEY (coluna_primaria), CONSTRAINT nome_da_outros_restricao FOREIGN KEY (coluna_fk) REFERENCES outra_tabela(coluna_referenciada));
(3) ALTER TABLE nome_da_tabela ADD nome_da_nova_coluna tipo_de_dado [restricoes];	(6) ALTER TABLE nome_da_tabela RENAME TO novo_nome_da_tabela
(4) ALTER TABLE nome_da_tabela DROP COLUMN nome_da_coluna;	(7) ALTER TABLE nome_da_tabela ADD CONSTRAINT nome_da_restricao PRIMARY KEY (coluna);
(5) ALTER TABLE nome_da_tabela MODIFY COLUMN nome_da_coluna novo_tipo_de_dado;	(8) ALTER TABLE nome_da_tabela DROP CONSTRAINT nome_da_restricao;
(9) ALTER TABLE nome_da_tabela RENAME COLUMN nome_atual_da_coluna TO novo_nome_da_coluna;	(10) DROP TABLE nome_da_tabela

- (1) Criação de tabela com atributos, tipo de dado e restrições do atributo.
- (2) Criação de tabela com atributos, tipo de dado, restrições do atributo e restrições da tabela, como chave primária e chave estrangeira.
- (3) Alteração de tabela incluindo atributo, tipo de dado e restrição do atributo.
- (4) Alteração de tabela removendo atributo.
- (5) Alteração de tabela modificando o tipo de dado do atributo.
- (6) Renomear tabela.
- (7) Alteração de tabela incluindo restrição.
- (8) Alteração de tabela removendo restrição.
- (9) Alteração de tabela renomeando atributo.
- (10) Apagar tabela.

Quanto aos tipos de restrições que podem ser criadas, têm-se:

Tipos de restrições

Tipo	Explicação	Exemplo
NOT NULL	Impede que uma coluna aceite valores nulos, garantindo que cada registro tenha um valor para essa coluna.	<code>CREATE TABLE exemplo (id INT NOT NULL, nome TEXT(50) NOT NULL);</code>
UNIQUE	Garante que todos os valores em uma coluna (ou em um conjunto de colunas) sejam únicos em toda a tabela.	<code>CREATE TABLE exemplo (id INT UNIQUE, email TEXT(100) UNIQUE);</code>
PRIMARY KEY	Combina as restrições NOT NULL e UNIQUE. A coluna (ou conjunto de colunas) é uma chave primária da tabela quando ela é o identificador único para a entidade ou relacionamento.	<code>CREATE TABLE exemplo (id INT PRIMARY KEY, nome TEXT(50) NOT NULL);</code>
FOREIGN KEY	Define uma chave estrangeira, criando uma ligação entre duas tabelas. A coluna referenciada deve ser uma chave primária na tabela referenciada.	<code>CREATE TABLE ordens (ordem_id INT PRIMARY KEY, produto_id INT, FOREIGN KEY (produto_id) REFERENCES produtos(id));</code>
CHECK	Define uma condição que os valores em uma coluna devem atender.	<code>CREATE TABLE exemplo (idade INT CHECK (idade >= 18), salario DECIMAL(10, 2) CHECK (salario > 0));</code>
DEFAULT	Especifica um valor-padrão para uma coluna, que será usado quando nenhum valor é fornecido durante a inserção de dados.	<code>CREATE TABLE exemplo (status TEXT(10) DEFAULT 'ativo');</code>

SQLite

Qualquer SGBD relacional poderia ser utilizado para as atividades práticas, mas neste curso iremos utilizar o SQLite por sua facilidade de uso, além de ser um **banco pequeno, rápido e portável**. Porém, é importante ressaltar algumas de suas particularidades:

- Foca o **armazenamento local** de dados para aplicativos e dispositivos individuais.
- **Suporta um número ilimitado de leitores simultâneos**, mas só permite um atualizador por vez.

- O banco de dados pode chegar a **281 terabytes**, mas é armazenado em um único arquivo de disco, enquanto a maioria dos sistemas de arquivos limita o tamanho máximo dos arquivos a algo menor do que isso.
- **Não tem uma classe de armazenamento ou tipo de dados especializado para armazenar datas e/ou horas.** Em vez disso, as funções de data e hora nativas do SQLite são capazes de manipular datas e horas como valores TEXT, REAL ou INTEGER.
- Algumas características da linguagem SQL **não são suportadas**, como: *ALTER TABLE*: Somente as variantes *RENAME TABLE*, *ADD COLUMN*, *RENAME COLUMN* e *DROP COLUMN* do comando *ALTER TABLE* são suportadas. Outros tipos de operações *ALTER TABLE*, como *ALTER COLUMN*, *ADD CONSTRAINT* e assim por diante, são omitidos.

Como criar um banco de dados com o SQLite?

Veja a seguir:

Interativo

Descrição do interativo

DDL: Create

Quando o mapeamento do esquema conceitual para o esquema lógico estiver completo, é necessário **implementar as tabelas em um SGBD utilizando a fração DDL da linguagem SQL** que permite criar os objetos do banco de dados relacional.

Cabe, nesse momento, relembrar a importância das regras ou restrições de integridade no modelo de dados relacional. Para bem definir um esquema relacional particular, e garantir no banco de dados apenas as informações úteis e desejadas, é necessário especificar restrições que controlem a integridade dos dados que se desejam representar.

Voltando ao comando **CREATE TABLE**, tem-se que a linguagem oferece alguns tipos de dados que permitem controlar domínios de valores para cada coluna da tabela. Por exemplo, para atributos numéricos, existem tipos como **INTEGER** e **NUMERIC**, que garantem não apenas que dígitos sejam utilizados, mas que expressões aritméticas também possam ser usadas, além de controlarem o total de algarismos, a parte inteira e decimal de um número. Se for preciso armazenar uma cadeia de caracteres (**strings**) qualquer, envolvendo letras e números, pode-se utilizar o tipo **TEXT**, especificando o tamanho máximo.

A seguir, tem-se a sequência de comandos DDL para criar o modelo físico correspondente à plataforma de cursos (EAD):

Interativo

Descrição do interativo

Analisando alguns trechos específicos, destacam-se:

Na tabela CEP_CORREIOS, temos a chave primária na coluna CEP do tipo inteiro e obrigatória:

CEP **INTEGER** **CONSTRAINT** CEP_CHAVE_PRIMARIA **PRIMARY KEY**

CONSTRAINT CEP_OBRIGATORIO **NOT NULL**

Na mesma tabela, tem-se outra coluna obrigatória do tipo texto e tamanho máximo de 100 caracteres:

LOGRADOURO TEXT (100) CONSTRAINT LOGRADOURO_OBRIGATORIO NOT NULL

A tabela ALUNO tem uma chave primária inteira e obrigatória, cujo valor é gerado automaticamente usando a cláusula AUTOINCREMENT.

MATRICULA INTEGER CONSTRAINT ALUNO_CHAVE_PRIMARIA PRIMARY KEY AUTOINCREMENT CONSTRAINT MATRICULA_OBRIGATORIA NOT NULL

E uma chave estrangeira para a tabela CEP_CORREIOS:

ENDERECO_CEP INTEGER CONSTRAINT CEP_ENDERECO_OBRIGATORIO NOT NULL CONSTRAINT CEP_CHAVE_ESTRANGEIRA REFERENCES CEP_CORREIOS (CEP)

Outro tipo de restrição encontrada na tabela ALUNO é a de domínio para o atributo SEXO, que é obrigatório:

SEXO TEXT (25) CONSTRAINT SEXO_OBRIGATORIO NOT NULL CONSTRAINT SEXO_INVALIDO CHECK (SEXO IN ('Feminino', 'Masculino', 'Prefiro não informar')),

Um caso de chave primária composta por dois atributos está na tabela Inscrito_Em:

CONSTRAINT INSCRITO_EM_CHAVE_PRIMARIA PRIMARY KEY (MATRICULA, CODIGO)

Todas as datas foram criadas usando o tipo de dados inteiro, como no caso do seguinte atributo da mesma tabela:

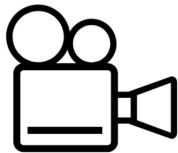
DATA_ULTIMA_ATIVIDADE INTEGER

Usando esses comandos DDL, pode-se criar o banco de dados da plataforma de cursos a distância (EAD) usando o *prompt* de comando do sqlite:

```
C:\sqlite>sqlite3 eadv1.db
SQLite version 3.45.1 2024-01-30 16:01:20 (UTF-16 console I/O)
Enter ".help" for usage hints.
sqlite> CREATE TABLE CEP_CORREIOS (
(x1...>     CEP INTEGER CONSTRAINT CEP_CHAVE_PRIMARIA PRIMARY KEY
(x1...>     CONSTRAINT CEP_OBRIGATORIO NOT NULL,
(x1...>     LOGRADOURO TEXT (100) CONSTRAINT LOGRADOURO_OBRIGATORIO NOT NULL,
(x1...>     BAIRRO TEXT (100) CONSTRAINT BAIRRO_OBRIGATORIO NOT NULL,
(x1...>     CIDADE TEXT (100) CONSTRAINT CIDADE_OBRIGATORIA NOT NULL,
(x1...>     ESTADO TEXT (100) CONSTRAINT ESTADO_OBRIGATORIO NOT NULL
(x1...> );
sqlite>
... Outros comandos ...
sqlite> CREATE TABLE Inscrito_Em (
(x1...>     MATRICULA INTEGER
(x1...>     CONSTRAINT ALUNO_INSCRITO_EM_CHAVE_ESTRANGEIRA REFERENCES ALUNO (MATRICULA)
(x1...>     CONSTRAINT MATRICULA_INSCRITA_EM_OBRIGATORIA NOT NULL,
(x1...>     CODIGO INTEGER
(x1...>     CONSTRAINT CODIGO_INSCRITO_EM_OBRIGATORIO NOT NULL
(x1...>     CONSTRAINT CURSO_INSCRITO_EM_CHAVE_ESTRANGEIRA REFERENCES CURSO (CODIGO),
(x1...>     DATA_INSCRIÇÃO INTEGER CONSTRAINT DATA_INSCRIÇÃO_OBRIGATORIA NOT NULL,
(x1...>     PERCENTUAL_PROGRESSO NUMERIC,
(x1...>     NOTA_FINAL INTEGER,
(x1...>     DATA_ULTIMA_ATIVIDADE INTEGER,
(x1...>     CONSTRAINT INSCRITO_EM_CHAVE_PRIMARIA PRIMARY KEY (MATRICULA, CODIGO)
(x1...> );
```

Comando do sqlite

Agora, você poderá acompanhar o passo a passo na prática.



Modelagem lógica e física utilizando o brModelo

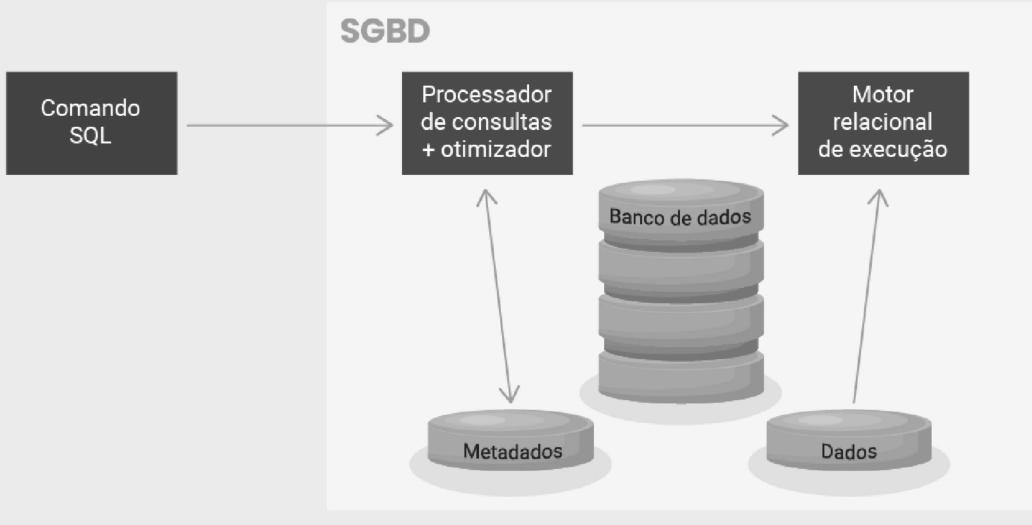
Neste vídeo, serão repetidas essas etapas de modelagem lógica e física com o apoio de uma ferramenta.



Catálogo de metadados

Os dados sobre as tabelas, restrições e outros objetos criados em um banco de dados também são armazenados em tabelas do **catálogo de metadados do SGBD**. Esse catálogo é consultado tanto por comandos de SQL elaborados por usuários quanto pela própria máquina de consultas do SGBD. Uma das etapas que a máquina de consultas realiza é o **parser do comando SQL recebido**. A verificação da sintaxe do comando SQL requer checar no catálogo de metadados se os objetos referenciados existem e se o usuário tem o respectivo acesso.

Processamento de consulta SQL pelo SGBD



O padrão ANSI SQL não define qual é o modelo de dados do dicionário de dados do SGBD. No entanto, define algumas visões (*views*) específicas do catálogo de metadados que permitem que os usuários consultem informações sobre objetos do banco de dados, como tabelas, índices, colunas, restrições, etc.

Algumas visões importantes para verificar a criação do modelo físico. Clique nos itens para saber mais:

INFORMATION_SCHEMA.TABLES



Contém informações sobre tabelas no banco de dados, como nome da tabela, tipo de tabela (tabela base, visão, etc.), tipo de armazenamento, etc.

INFORMATION_SCHEMA.COLUMNS



Fornece informações sobre as colunas das tabelas, incluindo o nome da coluna, tipo de dado, se a coluna permite nulos, etc.

INFORMATION_SCHEMA.KEY_COLUMN_USAGE

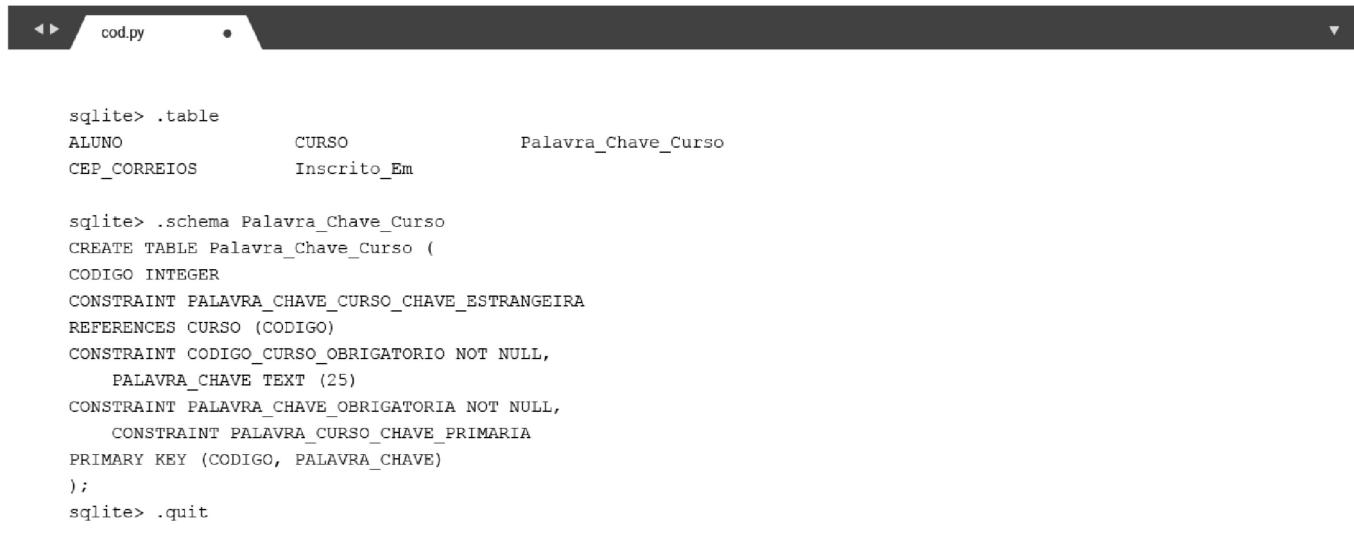


Fornece informações sobre as colunas que participam de chaves primárias e estrangeiras.

Essas visões do catálogo de metadados oferecem aos usuários uma maneira padronizada de consultar informações sobre a estrutura do banco de dados.

Verificação dos objetos criados

O SQLite não tem o esquema INFORMATION_SCHEMA como outros SGBDs relacionais e, portanto, podem-se usar os comandos de linha para verificar os objetos criados. Depois de executar todos os comandos DDL, é possível usar o comando **.table** para listar todas as tabelas criadas e o comando **.schema Nome_Tabela** para verificar a DDL de cada objeto:



```
cod.py
```

```
sqlite> .table
ALUNO          CURSO          Palavra_Chave_Curso
CEP_CORREIOS   Inscrito_Em

sqlite> .schema Palavra_Chave_Curso
CREATE TABLE Palavra_Chave_Curso (
    CODIGO INTEGER
        CONSTRAINT PALAVRA_CHAVE_CURSO_CHAVE_ESTRANGEIRA
            REFERENCES CURSO (CODIGO)
        CONSTRAINT CODIGO_CURSO_OBRIGATORIO NOT NULL,
        PALAVRA_CHAVE TEXT (25)
    CONSTRAINT PALAVRA_CHAVE_OBRIGATORIA NOT NULL,
        CONSTRAINT PALAVRA_CURSO_CHAVE_PRIMARIA
            PRIMARY KEY (CODIGO, PALAVRA_CHAVE)
);
sqlite> .quit
```

Comando .table e comando .schema Nome_Tabela

Outra alternativa é executar consultas SQL diretamente no catálogo de metadados. Antes, porém, será apresentada a **sintaxe básica** do SELECT:

DQL

SELECT básico	SELECT avançado
<pre>(2) SELECT column1, column2, column3 AS c3 ... FROM table1 WHERE condition;</pre>	<pre>(2) SELECT [DISTINCT ALL] column1, column2, ... FROM table1 [JOIN type JOIN table2 ON condition] WHERE condition GROUP BY column1, column2, ... HAVING condition ORDER BY column1 [ASC DESC], column2 [ASC DESC], ...;</pre>

Na cláusula **SELECT**, é possível especificar cada atributo de interesse separado por vírgulas para a consulta ou, então, pode-se substituir por * (asterisco), caso todos os atributos sejam relevantes para a consulta. Na cláusula **FROM**, é necessário especificar de qual tabela os dados devem ser lidos, sendo possível acessar mais de uma tabela por consulta. A cláusula **WHERE** é opcional, mas altamente recomendada para especificar o filtro das linhas dos dados a serem recuperados da tabela referenciada. Sem a cláusula WHERE, a consulta irá retornar todas as linhas da tabela.

Para consultar as tabelas de um banco sqlite, pode-se usar o comando SELECT:



```
SELECT name, sql  
FROM sqlite_schema  
WHERE type = 'table';
```

Comando SELECT

Aqui, **name** retornará o nome do objeto e a condição **type = 'table'** filtrará somente objetos que são tabelas. A coluna **sql** retorna o comando de DDL para a criação da respectiva tabela.

A sintaxe do SELECT permite que o nome das colunas receba apelidos (nomes alternativos) em tempo de consulta usando a palavra reservada AS. Então, o comando SELECT a seguir pode produzir o mesmo resultado do anterior, mas com novos nomes de colunas:

```
SELECT name AS nome_tabela, sql AS comando_DDL
FROM sqlite_schema
WHERE type = 'table';
```

Comando SELECT

DDL: *alter* e *drop*

Tão importante quanto criar é manter as estruturas do banco de dados aderentes aos requisitos de dados das aplicações. Suponha que, durante o desenvolvimento, a área usuário solicitou uma mudança na estrutura do endereço. Em vez do nome do Estado, a aplicação utilizará a sigla de Unidade da Federação, um campo texto de duas posições. Essa alteração requer que o DER, assim como o dicionário de dados, seja atualizado. Além disso, a tabela CEP_CORREIOS também precisa refletir essa mudança de requisito. Nesse caso, podem-se usar comandos DDL ALTER para remover o atributo Estado e incluir o atributo UF enquanto a tabela estiver vazia.

```
ALTER TABLE CEP_CORREIOS
ADD COLUMN UF TEXT(2) CONSTRAINT UF_OBRIGATORIO NOT NULL;

ALTER TABLE CEP_CORREIOS DROP COLUMN ESTADO;
```

SQL DDL: ALTERAÇÃO

Bons estudos e boa prática.



Refita

A construção de um esquema relacional completo, com tabelas, atributos e restrições de integridade, por meio dos comandos DDL CREATE e ALTER, gera metadados no catálogo do SGBD na fase de modelagem física.

No entanto, ao longo das etapas anteriores (conceitual e lógica), um dicionário de dados com nome, tipo, domínio, obrigatoriedade, descrições e regras de negócio para as entidades, atributos e relacionamento também é criado.

Refita sobre a importância de manter a documentação sobre a modelagem de dados, tanto o DER quanto o dicionário de dados, depois da criação do banco de dados. Pesquise sobre o papel do

administrador de dados e do administrador de banco de dados nas organizações.