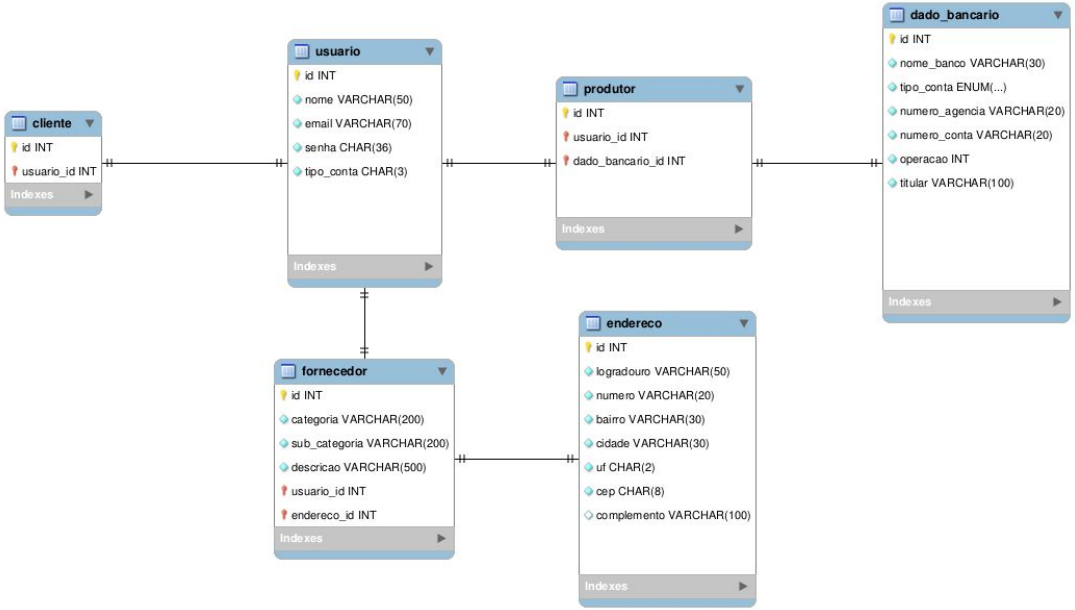
SQL

Modelo E/R (entidade/relacionamento)

O **M**odelo **E**ntidade **R**elacionamento é uma forma lógica de pensarmos na estrutura de um banco de dados e assim definir suas **Entidades**(tabelas), **Atributos**(colunas) e **Relacionamentos**(chaves).



Modelo E/R - Entidades

No modelo E/R, as entidades são armazenadas em um modelo tabular. Onde uma tabela representa uma entidade, suas colunas representam seus atributos e suas linhas representam as suas instâncias.

Exemplo: vamos supor que tenhamos uma entidade Carro persistida no banco de dados. Teríamos então a seguinte tabela:

Carro

id	***Nome***	***Marca***	***Modelo***	***Ano***
1	Uno	Fiat	Attractive 1.0 Flex	2015
2	Gol	Volkswagen	Comfortline	2017
3	Onix	Chevrolet	LT Turbo	2022

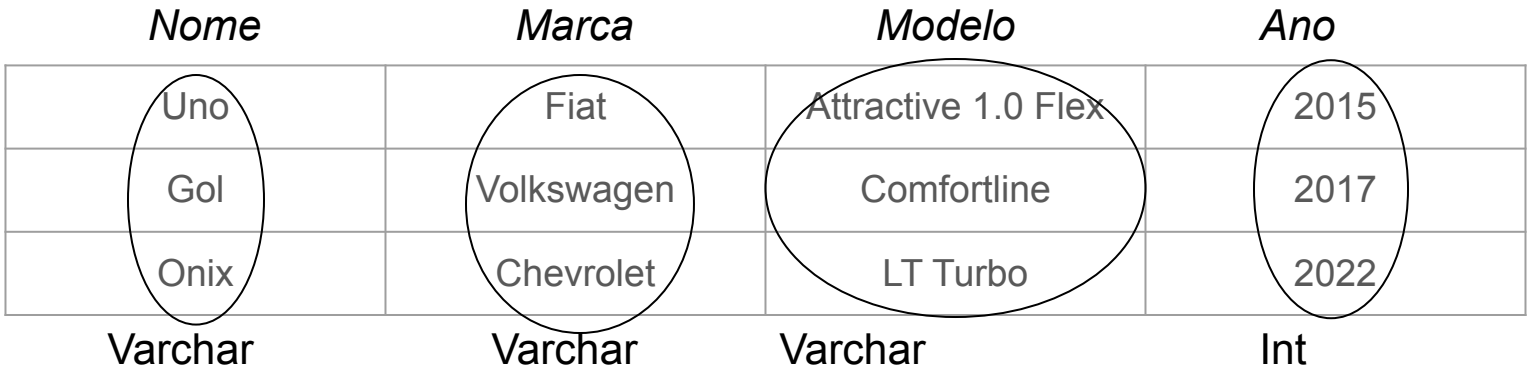
Modelo E/R - Atributos

Os atributos são as características e informações que cada entidade possui. Eles podem ser de alguns tipos diferente:

- **Atributos Nominativos** – são aqueles que identificam um objeto da classe, como por exemplo o CPF de um cliente, o nome do cliente, ou até mesmo o código do cliente dentro do sistema;
- **Atributos Descritivos** – são aqueles que descrevem alguma informação do objeto, como por exemplo, data de nascimento do cliente, o nome do cliente também pode ser um atributo descritivo, endereço, etc;
- **Atributos Referenciais** – são atributos que realizam a ligação entre entidades, como por exemplo o código do cliente dentro do pedido de venda, o código do produto dentro do pedido de venda, o código da categoria dentro do produto, etc

Modelo E/R - Atributos

Carro



Onde cada coluna possui um tipo de dado. Podendo ser Strings, inteiros, decimais e etc.

Atributos - Tipo de dados

Anteriormente abordamos que cada linha de uma tabela SQL possui um tipo de dado, os principais dados aceitos pela linguagem SQL são:

- **Numéricos exatos**
- **Numéricos aproximados**
- **Data e hora**
- **Cadeias de caracteres**
- **Cadeias de caracteres Unicode**

Numéricos Exatos

Tipo inteiro

Tipo	*Intervalo*	*Armazenamento*	*Declaração*
bigint	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	8 bytes	bigint
int	-2.147.483.648 a 2.147.483.647	4 bytes	int
smallint	-32.768 a 32.767	2 bytes	smallint
tinyint	0 a 255	1 byte	tinyint

Numéricos Exatos

Tipo decimal

Tipo	*Precisão*	*Armazenamento*	*Declaração*
decimal	1 - 9	5 bytes	**decimal (M, N)** **M - nº dígitos no total** **N - nº dígitos de precisão**
	10 - 19	9 bytes	
	20 - 28	13 bytes	
	29 - 38	17 bytes	

Numéricos Aproximados

Tipo float

Tipo	*Valor de n*	*Precisão*	*Armazenamento*	*Declaração*
float	1 - 24	7 dígitos	4 bytes	**float** **float (n)**
	25 - 53	15 dígitos	8 bytes	

Data e hora

Tipo	***Intervalo***	***Declaração***
date	0001-01-01 a 9999-12-31	date
datetime	Janeiro 1, 1753, a dezembro 31, 9999; 00:00:00 a 23:59:59.997	datetime
datetimeoffset	0001-01-01 a 9999-12-31; 00:00:00 a 23:59:59.9999999; -14:00 a +14:00	datetimeoffset [utc]
time	00:00:00.0000000 a 23:59:59.9999999	time

Cadeia de caracteres

Tipo	***Intervalo***	***Declaração***
char	Máximo 8000 caracteres	char(n)
varchar	Máximo 1.073.741.824 caracteres	varchar(max)

CHAR e VARCHAR são tipos de dados caractere, a diferença é que **CHAR** é um tipo de dado de comprimento fixo e **VARCHAR** é de comprimento variável.

Bibliografia

https://learn.microsoft.com/pt-br/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver16

Modelo E/R - Relacionamentos

Agora que definimos entidades e atributos, vamos definir os possíveis relacionamentos entre as entidades, para isso vamos definir três tipos de chaves e três formas de relacionamentos.

- Chave primária (PRIMARY KEY - PK)
- Chave única (UNIQUE KEY - UK)
- Chave estrangeira (FOREIGN KEY - FK)
- One-To-One
- One-To-Many
- Many-To-Many

Chave primária (primary key - pk)

A chave primária (PRIMARY KEY) identifica exclusivamente cada registro (instância) em uma tabela. As chaves primárias devem conter valores ÚNICOS e não podem conter valores NULL. Uma tabela pode ter apenas UMA chave primária que não deve ser alterada. Uma chave primária pode ser alterada, mas para alterá-la, é necessário ter a certeza que essa alteração não causará a quebra do banco de dados, é um processo trabalhoso e por isso não deve ser alterado.

Carro (id = pk)

id	***Nome***	***Marca***	***Modelo***	***Ano***
1	Uno	Fiat	Attractive 1.0 Flex	2015
2	Gol	Volkswagen	Comfortline	2017

Tabela carro

```
create table carro(
	id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
	nome VARCHAR(100),
	modelo VARCHAR(100),
	ano int
);
```

Chave única (unique key - uk)

Uma chave única é o conjunto de campos ou colunas de uma tabela que nos ajuda a identificar registros de forma única. A chave exclusiva garante a exclusividade das colunas no banco de dados. É semelhante à chave primária, mas pode aceitar um valor nulo, ao contrário dela.

Pessoa (id = pk, email telefone = uk)

id	***Nome***	***Email***	***Telefone***	***Nascimento***
1	Rafael	rafael.10@teste.com	21 0000-0000	2000/12/10
2	Carla	carla.10@teste.com	21 1111-1111	1999/01/05

Tabela pessoa

```
create table pessoa(
	id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
	nome VARCHAR(100),
	email VARCHAR(100),
	telefone VARCHAR(12),
	nascimento DATE,
	CONSTRAINT pessoa UNIQUE KEY (email), UNIQUE KEY (telefone)
);
```

Chave estrangeira (foreign key - fk)

Para criarmos um relacionamento entre duas tabelas, precisamos de duas chaves, uma é a chave primária e a outra é a chave estrangeira.

A chave primária é uma coluna que identifica exclusivamente uma determinada linha em uma tabela. A chave estrangeira é uma coluna em uma tabela cujos valores são referenciados a partir de uma chave primária em outra tabela. Uma chave primária em SQL identifica exclusivamente registros em uma tabela. No entanto, você deve ter uma chave primária para usar uma chave estrangeira no SQL.

Tabela Customers

```
create table customers(
        customer_id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        first_name VARCHAR(100),
        last_name VARCHAR(100),
        age int,
        country VARCHAR(5)
);
```

Tabela Orders

```
create table orders(
    order_id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    customer_id BIGINT NOT NULL,
    product VARCHAR(100),
    total DECIMAL(10,2),
    CONSTRAINT fk_order_customers FOREIGN KEY(customer_id) REFERENCES customers(customer_id)
);
```

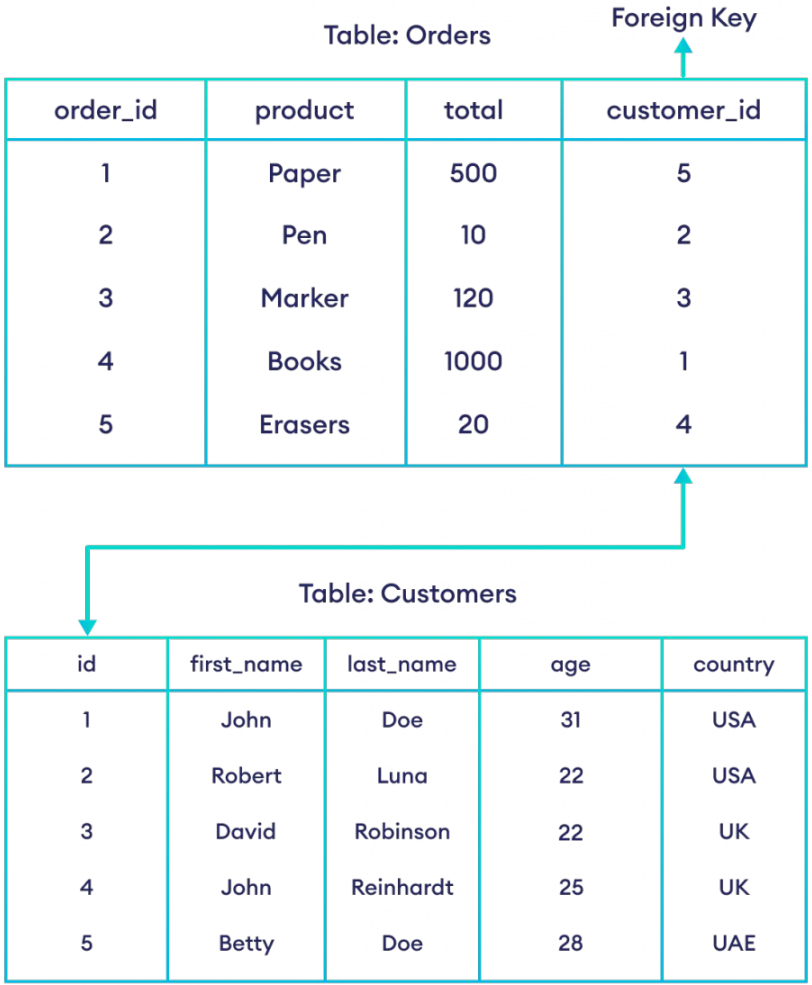


Table: Orders
Foreign Key
order_id
product
total
customer_id
1
Paper
500
5
2
Pen
10
2
3
Marker
120
3
4
Books
1000
1
5
Erasers
20
4
Table: Customers
id
first_name
last_name
age
country
1
John
Doe
31
USA
2
Robert
Luna
22
USA
3
David
Robinson
22
UK
4
John
Reinhardt
25
UK
5
Betty
Doe
28
UAE

Observações

O nome das chaves estrangeiras devem ser únicas dentro de um banco de dados, deve-se tomar cuidado ao declarar essas chaves.

As **constraint** presentes no script SQL são usadas para especificar regras para os dados em uma tabela. As **constraint** são usadas para limitar o tipo de dados que podem entrar em uma tabela. Isso garante a precisão e a confiabilidade dos dados na tabela. Se houver alguma violação entre a restrição e a ação de dados, a ação será abortada.

Por convenção, as palavras reservadas do SQL são escritas em UpperCase para facilitar a leitura do código.

One - To - One

No relacionamento One-to-One, um registro da primeira tabela será vinculado a zero **ou** um registro de outra tabela. Por exemplo, vamos voltar à tabela anterior onde cada customer na tabela **Customers** terá uma linha correspondente na tabela **Orders** que armazena os detalhes da order atual deste customer em particular. Assim, cada customer terá zero **ou** um registro na tabela **Orders**. Isso é chamado de relacionamento um para um (one-to-one).

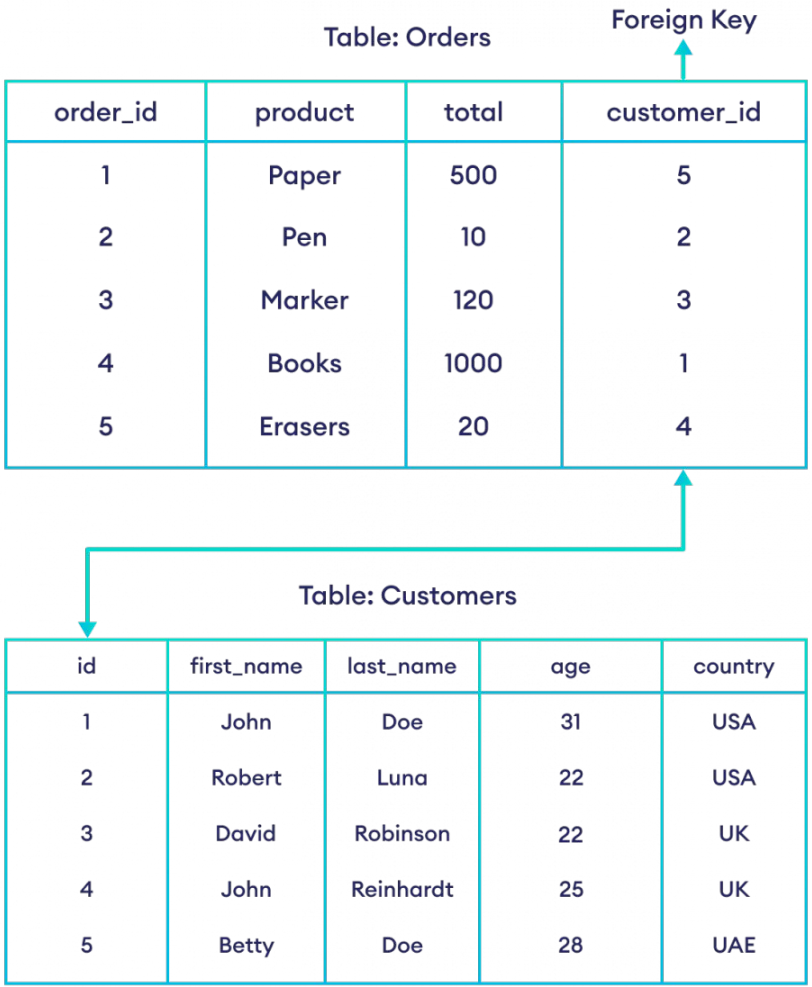
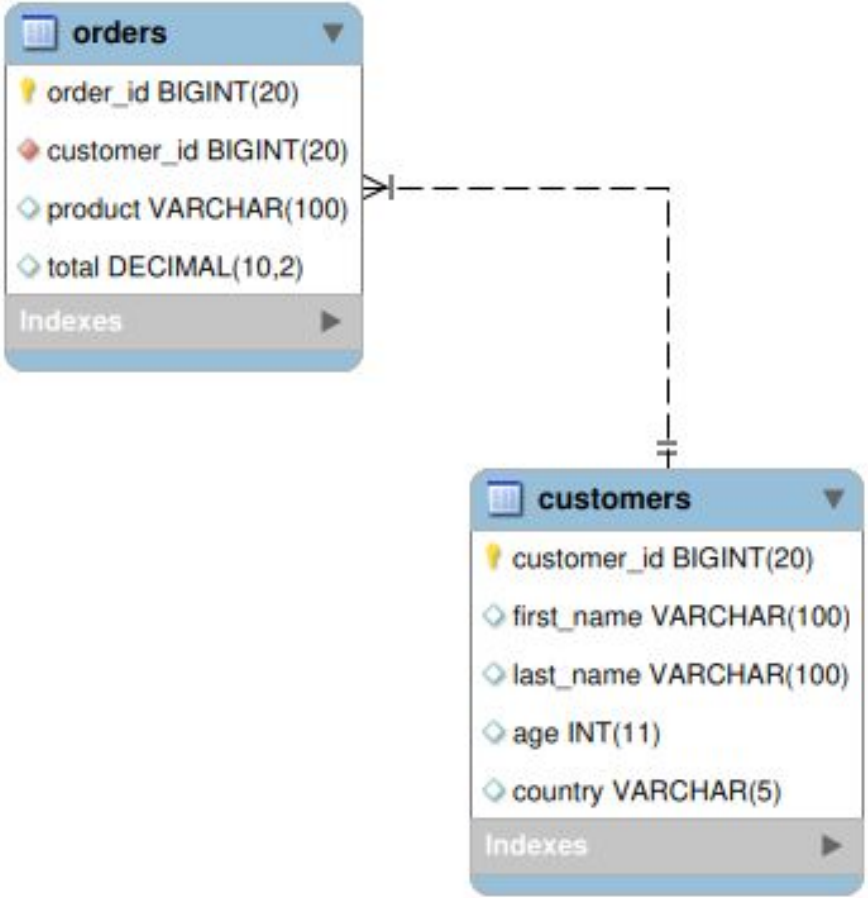


Table: Orders
Foreign Key
order_id
product
total
customer_id
1
Paper
500
5
2
Pen
10
2
3
Marker
120
3
4
Books
1000
1
5
Erasers
20
4
Table: Customers
id
first_name
last_name
age
country
1
John
Doe
31
USA
2
Robert
Luna
22
USA
3
David
Robinson
22
UK
4
John
Reinhardt
25
UK
5
Betty
Doe
28
UAE

One - To - Many

Um-para-muitos é a relação mais comum entre tabelas. Um único registro de uma tabela pode ser vinculado a um ou mais linhas em outra tabela.

Vamos dar um exemplo da tabela **Customers** e **Orders**. A tabela **Customers** armazena registros de clientes onde **customer_id** é a chave primária. A tabela **Orders** contém pedidos dos clientes onde **orders_id** é uma chave primária e **customer_id** é uma chave estrangeira. Cada cliente terá um ou mais registro na tabela **Orders**. Cada cliente pode ter vários pedidos. As tabelas **Customers** e **Orders** são veiculadas pela coluna-chave **customer_id**. É uma chave estrangeira na tabela **Orders** vinculada à chave primária **customer_id** na tabela **Customers**. Assim, um registro da tabela **Customers** pode apontar para vários registros na tabela **Orders**. Este é um relacionamento um-para-muitos.


orders
order_id BIGINT(20)
customer_id BIGINT(20)
product VARCHAR(100)
total DECIMAL(10,2)
Indexes
customers
customer_id BIGINT(20)
first_name VARCHAR(100)
last_name VARCHAR(100)
age INT(11)
country VARCHAR(5)
Indexes

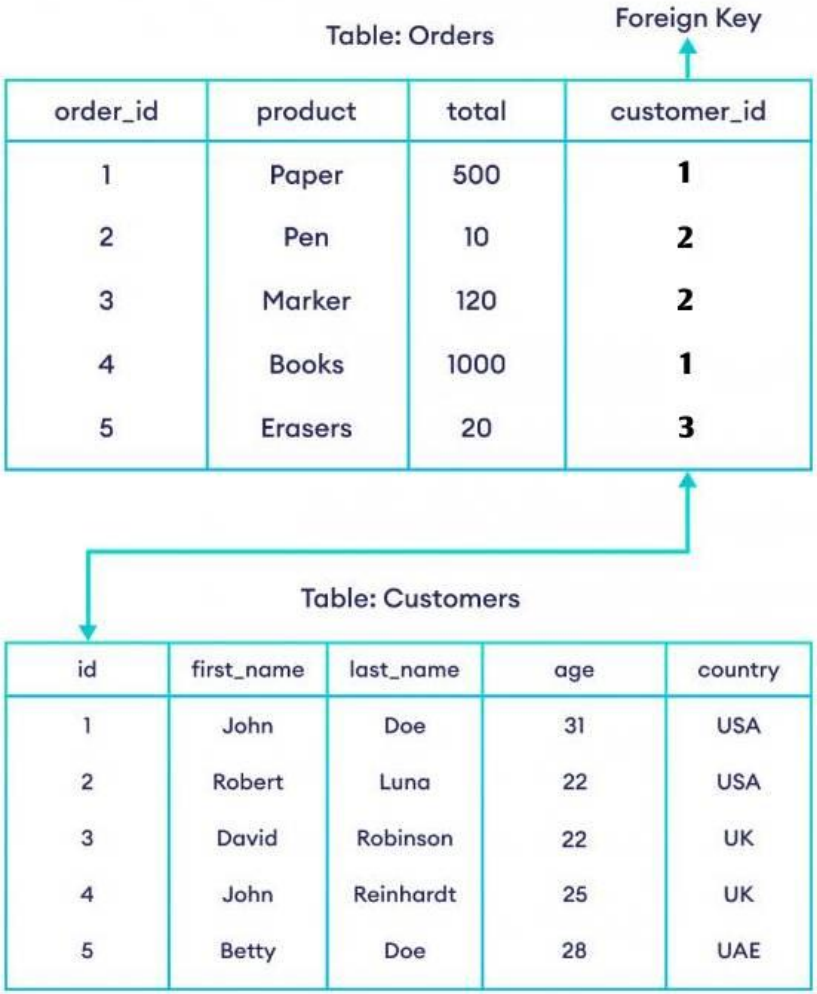


Table: Orders
Foreign Key
order_id
product
total
customer_id
1
Paper
500
1
2
Pen
10
2
3
Marker
120
2
4
Books
1000
1
5
Erasers
20
3
Table: Customers
id
first_name
last_name
age
country
1
John
Doe
31
USA
2
Robert
Luna
22
USA
3
David
Robinson
22
UK
4
John
Reinhardt
25
UK
5
Betty
Doe
28
UAE

Many - To - Many

O relacionamento Muitos-para-Muitos permite relacionar cada linha em uma tabela a muitas linhas em outra tabela e vice-versa. Por exemplo, um aluno da tabela **Student** pode ter vários cursos da tabela **Course** e também um curso pode estar associado a um ou mais alunos.

A figura a seguir demonstra a relação muitos-para-muitos entre a tabela **Student** e **Course** usando a tabela de junção **Student_Course**. Onde “quebramos” a relação muitos-para-muitos em duas relações um-para-muitos.

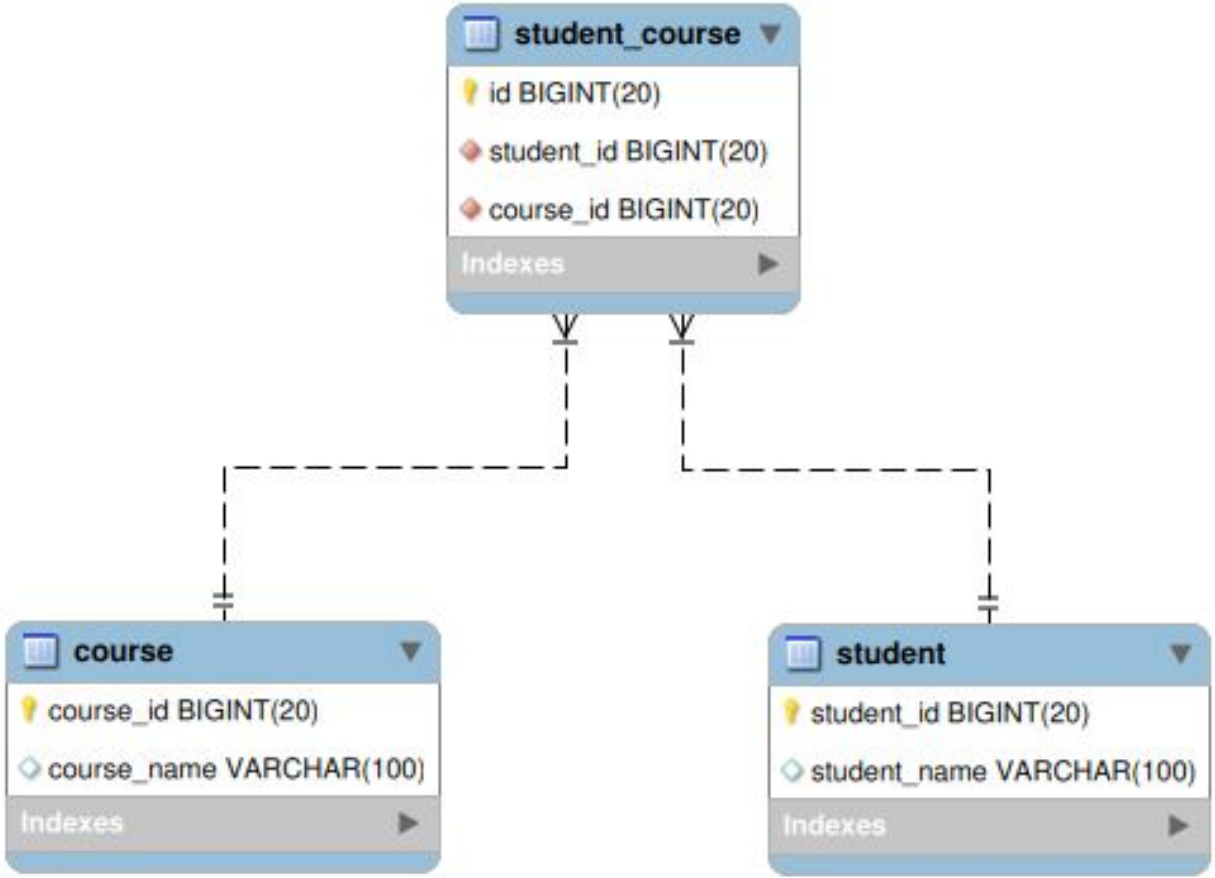

student_course
id BIGINT(20)
student_id BIGINT(20)
course_id BIGINT(20)
Indexes
course
course_id BIGINT(20)
course_name VARCHAR(100)
Indexes
student
student_id BIGINT(20)
student_name VARCHAR(100)
Indexes

Tabela Student

```
CREATE TABLE student(
    student_id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    student_name VARCHAR(100)
);
```

Tabela Course

```
CREATE TABLE course(
    course_id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    course_name VARCHAR(100)
);
```

Tabela de associação Student_Course

```
CREATE TABLE student_course(
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    student_id BIGINT NOT NULL,
    course_id BIGINT NOT NULL,
    CONSTRAINT fk_student_course FOREIGN KEY(student_id) REFERENCES student(student_id),
   CONSTRAINT fk_course_student FOREIGN KEY(course_id) REFERENCES course(course_id)
);
```

Student_Course

id	student_id	course_id
1	1	1
2	1	2
3	2	1

id	**nome**
1	Fabio
2	Carla

Student

id	**nome**
1	Ciência da Computação
2	Sistemas de Informação

Course

Os 5 subgrupos do SQL

- DQL (Data Query Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language)
- DDL (Data Definition Language)
- DTL (Data Transaction Language)

Cada subgrupo SQL possui comandos próprios de execução e ao executar estes comandos sempre temos como resultado duas coisas:

1. O resultado da execução do comando;
2. Uma mensagem de execução que pode ser de sucesso ou de erro.

DQL - Data Query Language

No subgrupo DQL temos apenas 1 comando SQL, o SELECT. Embora tenha apenas um comando, a DQL é a parte da SQL mais utilizada. O comando SELECT permite ao usuário especificar uma consulta (query) como uma descrição do resultado esperado.

Exemplo:

```
SELECT * FROM orders;
```

DML - Data Manipulation Language

No subgrupo DML temos 3 comandos SQL, INSERT, UPDATE e DELETE:

- INSERT: Usado para inserir um registro a uma tabela existente;
- UPDATE: Usado para alterar valores de dados em um ou mais registros de uma tabela;
- DELETE: Usado para remover registros de uma tabela

Insert

O comando INSERT nos permite inserir instâncias em uma tabela de determinado banco de dados.

```
INSERT INTO customers(customer_id,first_name,last_name,age,country)
VALUES (1,'Fabio', 'Correa', 10, 'br');
```

Neste exemplo, estamos inserindo os valores 1, ‘Fabio’, 'Correa', 10, ‘br’ na tabela 'customers'.

Update

UPDATE customers set first_name = 'Rafael' WHERE customer_id = 1;

Neste exemplo, estamos atualizando o registro com customer_id = 1 será atualizado com o novo valor para o campo 'Rafael'

Jamais deve-se utilizar o comando UPDATE sem o comando WHERE, pois estaremos atualizando todos os registros presentes no banco de dados. Essa atualização ocasionaria a perda permanente de dados, e caso o banco não tenha backup, os dados serão perdidos de forma definitiva.

DELETE

O comando DELETE nos permite deletar estruturas e instâncias presentes em um determinado banco de dados.

```
DELETE FROM customers WHERE customer_id = 1;
```

Neste exemplo, estamos excluindo o registro com customer_id = 1.

Jamais deve-se utilizar o comando DELETE FROM sem o comando WHERE, pois estaremos deletando todos os registros presentes no banco de dados. Essa exclusão ocasionaria a perda permanente de dados, e caso o banco não tenha backup, os dados serão perdidos de forma definitiva.

DDL - Data Definition Language

No subgrupo DDL temos 3 comandos SQL, CREATE, ALTER e DROP:

- CREATE: Usado para criar um banco de dados, tabela e outros objetos em um banco de dados;
- ALTER: Usado para alterar a estrutura de tabelas ou outro objeto em um banco de dados;
- DROP: Usado para apagar bancos de dados, tabela e outros objetos;

Create

O comando CREATE podemos criar databases, tabelas e etc.

CREATE DATABASE teste;

Neste exemplo, estamos criando um banco de dados chamado ‘teste’;

CREATE TABLE produto (....);

Neste exemplo, criamos a tabela ‘produto’ no banco de dados ‘teste’ criado anteriormente.

ALTER

O comando ALTER permite que façamos alterações nas estruturas das tabelas existentes em um determinado banco de dados;

ALTER TABLE produto ADD peso DECIMAL(8,2);

Neste exemplo, alteramos a estrutura da tabela 'produto' e adicionamos um novo campo chamado 'peso' do tipo decimal com até 8 dígitos e com 2 dígitos de precisão.

ALTER TABLE produto DROP peso;

Neste exemplo, alteramos a estrutura da tabela 'produto' e excluindo o campo chamado 'peso' do tipo decimal com até 8 dígitos e com 2 dígitos de precisão.

Drop

O comando DROP permite que façamos alterações de exclusão nas estruturas das tabelas existentes em um determinado banco de dados;

DROP TABLE produto;

Neste exemplo, estamos apagando a tabela 'produto'. Este comando apaga toda a estrutura e os dados, desde que esta tabela não tenha relacionamentos.

DROP DATABASE teste;

Neste exemplo, estamos apagando o banco de dados 'financeiro' e qualquer tabela ou dado dentro dele.

DCL (Data Control Language)

No subgrupo DCL temos 2 comandos SQL, o GRANT e o REVOKE.

- GRANT: Usado para autorizar um usuário a executar ou setar operações no banco de dados;
- REVOKE: Usado para remover ou restringir a capacidade de um usuário de executar operações;

GRANT

GRANT SELECT ON produto TO user_bd;

Neste caso, estamos dando permissão de consulta na tabela produto para o usuário ‘user_bd’.

REVOKE

```
REVOKE CREATE TABLE FROM user_bd;
```

Neste caso, estamos retirando a permissão de criação de tabelas no banco de dados para o usuário ‘user_bd’.

DTL - Data Transaction Language

No subgrupo DTL temos 4 comandos SQL, BEGIN, COMMIT e ROLLBACK.

- BEGIN (ou START TRANSACTION): Usado para marcar o começo de uma transação que pode ser completada ou não;
- COMMIT: Finaliza uma transação;
- ROLLBACK: Faz com que as mudanças nos dados existentes desde o último COMMIT sejam descartadas.

BEGIN

```
BEGIN TRANSACTION; --começamos a transação
    INSERT INTO produto VALUES ('Notebook');
    INSERT INTO produto VALUES ('Nobreak');
COMMIT; --termina a transação e grava os dados
```

Neste exemplo, estamos iniciando a transação, inserindo os dados e salvando no banco.

COMMIT

O comando COMMIT termina a transação e grava os dados.

ROLLBACK

```
BEGIN TRANSACTION; --começamos a transação
    INSERT INTO produto VALUES ('Notebook');
    INSERT INTO produto VALUES ('Nobreak');
ROLLBACK; --as inserções das linhas acima serão desfeitas
```

Neste exemplo, estamos iniciando a transação, inserindo os dados, porém, note que o comando COMMIT não está presente e não foi executado, em seu lugar executamos o comando ROLLBACK. Desta forma, as inserções foram desfeitas e o banco de dados voltará ao estado anterior ao início da transação.