Normalização de tabela, tipo de dados, modelo de bases de dados relacional e criação de database

Oscar J. O. Ayala

2023

1 Introdução

SQL é uma linguagem padrão para criar, armazenar, manipular e recuperar bancos de dados. Este documento cobre os tópicos básicos em SQL. Os códigos podem ser estendidos para SQL Server, MS Access, Oracle, Sybase, Informix, Postgres e outros sistemas de banco de dados.

1.0.1 O que é SQL?

O nome SQL significa Linguage de consulta de estrutura (ou em inglês, Structured Query Language), permite accesar e manipular base de dados, e tornou-se um padrão do American National Standards Institute (ANSI) em 1986, e da International Organization for Standardization (ISO) em 1987.

1.0.2 O que o SQL pode fazer?

- SQL pode executar consultas em um banco de dados
- SQL pode recuperar dados de um banco de dados
- SQL pode inserir registros em um banco de dados
- SQL pode atualizar registros em um banco de dados
- SQL pode excluir registros de um banco de dados
- SQL pode criar novos bancos de dados
- SQL pode criar novas tabelas em um banco de dados
- SQL pode criar procedimentos armazenados em um banco de dados
- SQL pode criar visualizações em um banco de dados
- SQL pode definir permissões em tabelas, procedimentos e visualizações

1.1 Tabelas

No SQL, uma tabela é dividida em entidades menores chamadas campos e registros. Um campo é uma coluna em uma tabela projetada que lista informações específicas para cada registro (por exemplo, ID do cliente, endereço, entre outros). Um registro é uma linha e se refere a entradas individuais na tabela projetada

1.1.1 Tabelas normalizadas e não normalizada

Tabelas normalizadas referem-se àquela relação de campos e registros, em que não há duplicações de linha e/ou coluna.No entanto, as tabelas não normalizados têm duplicações de recursos verticais e/ou horizontais e requerem mais espaço e uso de memória. Assim, é conveniente trabalhar com tabelas padronizadas.

1.1.2 Estruturas tabelas normalizadas

Muitos casos requerem tabelas com o mínimo de informação possível (eficiência). As tabelas normalizadas permitem estilizar, separar informações, em várias tabelas de uma tabela não normalizada. Por exemplo, suponha que você tenha uma tabela não normalizada para uma empresa como na Figura 1. Em os campos são: produtos, clientes, data de compra e valor do produto. Evidência-se que existem duplicações entre os registros com relação aos campos, o que dificulta uma boa análise e requer mais espaço.

Produto	Clientes	Data de compra	Valor do produt
A	Oscar Ayala	26/04/2022	150
B	Lucas Stefano	26/03/2022	150
A	Oscar Ayala	26/05/2022	140
C	Lucas Stefano	16/02/2022	130
A	Erik Cassiano	26/05/2022	120
B	Erik Cassiano	18/03/2022	150

Figura 1: Tabela não normalizada.

Gerar uma tabela normalizada requer a compreensão dos campos e registros. Na Figura 1, observe que o produto e o nome do cliente referem-se a compra do cliente, logo a compra está relacionada ao tipo de cliente, e este último às exceções de pagamento. A Figura 2 contém essas relações, que permitem a construção de diferentes tabelas normalizadas projetadas.

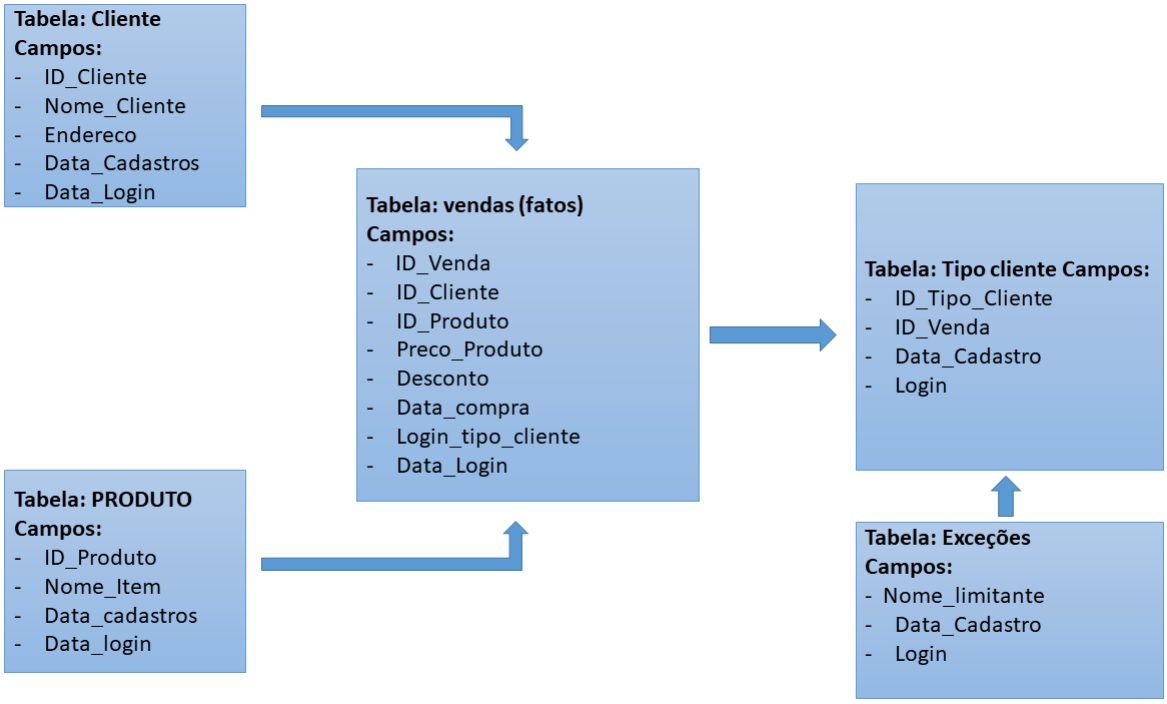

Figura 2: Tabelas normalizadas projetadas

Na Figura 2, note que varios campos são usados para projetar as tabelas normalizadas. As colunas **Cliente** e **Produto** permitem obter informçoes de interreses não repetidas sobre estes, o que fornecem duas tabelas com seus respetivos campos. Logo, estas tabelas fornecem informação a uma terceira tabela com o campo **vendas** e os campos respectos. Depois, se tem a tabela tipo de cliente com seus campos, que são influência com as exepções que pode modificar ou manter sua calificação. Note que os precios não foram colocados no cliente, porque não se relaciona diretamente com ele e se for colocado afetaria o registro anteiores de clientes se houver um desconto ou algo parecido.

Na Figura 2, observe que vários campos são usados para projetar as tabelas normalizadas. As colunas **Cliente** e **Produto** permitem obter informação de interesse não repetidas sobre os mesmos, que disponibiliza duas tabelas de dimensão. As tabelas então fornecem informações para uma terceira tabela de fato, com o nome **vendas** e seus respectivos campos. Adicionalmente, tem-se a tabela de tipos de clientes com seus campos, que é influenciada pela tabela de dimensão exceções que pode modificar ou manter sua qualificação. Observe que os preços não foram colocados em produtos, pois se fosse colocado afetaria o registro do produto se houver desconto ou algo parecido.

Observe que não há uma relação necessária entre o nome e número das tabelas normalizadas e os nomes e número dos campos da tabela não normalizada. Da mesma forma, é preciso usar a lógica e a criatividade.

1.2 Criação conceptual de bases de dados

Nesta seção, se cria o modelo de banco relacional a ser utilizado ao longo deste documento. O nome da database está relacionado com o projeto ou problema em estudo. A base de dados principal tem o nome de **SQL_BD_1**.

- **Chaves primárias (primary keys - PK)**: São aqueles que **não se repetem** e identificam de forma única um registro na mesma tabela, podendo ser usados como índice de referência para criar relacionamentos com as demais tabelas de um banco de dados. Por exemplo, o número de identificação do cidadão, a codificação da peça de automóvel, entre outros, todos são chaves primárias porque não se repetem e permitem relações com mais uma tabela numa base de dados.

  Na Figura 2, a tabela **cliente** tem como chave primária *ID_Cliente* e a tabela **Produto** tem a chave primária **ID_Produto**.

- **Chaves secundária ou estrangeira (Foreign Key - FK)**: Refere-se ao tipo de relacionamento entre diferentes tabelas de dados no banco de dados. Uma chave estrangeira é chamada quando existe um relacionamento entre *duas* tabelas por meio de uma chave primária. Sempre haverá relacionamentos entre tabelas em chaves secundárias, por exemplo, se uma tabela tiver uma chave primária em outra tabela.

Na Figura 2, a tablea **Vendas(fatos)** tem as chaves primárias *ID_Cliente* e **Produto** das tabelas **cliente** e **Produto**, assim essas resultam em chaves estrangerias para a tabela **Vendas (fatos)**. No entanto, na Tabela **Vendas (fatos)** o *ID_vendas* é uma chave primária.

1.3 tipo de dados

1.3.1 Tipo de dados de cadeia

1. **char(n)**: cadeia de caráteres de comprimento fixo, cumprimento máximo 8 caracteres, com armazenamento definido.
2. **varchar(n)**: cadeia de caracteres de comprimento variável, cumprimento máximo 8000 caracteres , com armazenamento de 2 bytes + número de caracteres. A variável n indica o número de caráteres desejado de 1 até 8000
3. **varchar(max)**: cadeia de caráteres de comprimento de variável, cumprimento máximo 1.073.741.824 caráteres, com armazenamento de 2 bytes + número de caracteres.
4. **text**: cadeia de caráteres de comprimento de variável, cumprimento máximo 2 GB de texto de dados, com armazenamento de 4 bytes + número de caracteres.
5. **nchar**: cadeia Unicode de comprimento fixo, cumprimento máximo 4.000 caráteres, com armazenamento de comprimento $\times$ 2.
6. **nvarchar**: cadeia Unicode de comprimento variável, cumprimento máximo 4.000 carateres.
7. **nvarchar(max)**: cadeia Unicode de comprimento variável, cumprimento máximo 536.870.912 carateres.
8. **ntext**: cadeia Unicode de comprimento variável, cumprimento máximo 2GB de texto de dados.
9. **binary(n)**: cadeia binária com comprimento fixo, comprimento máximo 8.000 bytes.
10. **varbinay**: cadeia binária com comprimento variável, comprimento máximo 8.000 bytes.
11. **varbinay(max)**: cadeia binária de comprimento variável, comprimento máximo 2GB.
12. **image**: cadeia binária de comprimento variável, comprimento máximo 2GB

1.3.2 Tipo de dados numéricos

1. **bit**: inteiro que pode ser 0, 1 ou NULL.
2. **tinyint**: permite números inteiros de 0 a 255, armazena 1 bytes.
3. **smallint**: permite números inteiros entre -32.768 e 32.768, armazena 2 bytes.
4. **int**: permite números inteiros entre $-2.147.483.648e2.147.483.648$, armazena 4 bytes.
5. **bigint**: permite números inteiros entre $-9.223.372.036.854.775.808$ e $9.223.372.036.854.775.808$, armazena 8 bytes.
6. **decimal(p, s)**: precisão fixa e números de escala. Permite números de $-10^{38}+1$ a $10^{38}-1$. O parâmetro p indica o número total máximo de dígitos que podem ser armazenados (à esquerda e à direita do ponto decimal). p deve ser um valor de 1 a 38, o padrão é 18. Enquanto, o parâmetro s indica o número máximo de dígitos armazenados à direita do ponto decimal, s deve ser um valor de 0 a p, o valor padrão é 0. Armazena de 5 a 17 bytes.
7. **numeric(p,s)**: Precisão fixa e números de escala. Permite números de $-10^{38}+1$ a $10^{38}-1$. O parâmetro p indica o número de casas décimais. p deve ser um valor de 1 a 38, o padrão é 18. Enquanto, o parâmetro s indica o número máximo de decimais, s deve ser um valor de 0 a p, o valor padrão é 0. Armazena de 5 a 17 bytes.
8. **smallmoney**: dado monetário desde $-214.748,3648$ a $214.748,3648$, armazena 4 bytes.
9. **money**: dado monetário desde $-922.337.203.685.477,5808$ e $922.337.203.685.477,5808$, armazena 8 bytes.
10. **float(n)**: dados de número de precisão flutuante de $-1,79E+308$a$1,79E+308$. O parâmetro n indica se o campo deve conter 4 ou 8 bytes. Logo, float(24) contém um campo de 4 bytes e float(53) contém um campo de 8 bytes. O valor padrão de n é 53. Armazena de 4 a 8 bytes.
11. **real**: Dados de número de precisão flutuante de $-3,40E+38$a$3,40E+38$. Armazena 4 bytes.

1.3.3 Tipo de dados de datas e horas

1. **datetime**: De 1º de janeiro de 1753 a 31 de dezembro de 9999 com precisão de 3,33 milissegundos, armazena 8 bytes.
2. **datetime2**: De 1º de janeiro de 0001 a 31 de dezembro de 9999 com precisão de 100 nanossegundos, armazena de 6-8 bytes.
3. **smalldatetime**: De 1 de janeiro de 1900 a 6 de junho de 2079 com precisão de 1 minuto, armazena 4 bytes.
4. **date**: *Armazenar apenas uma data.* De 1º de janeiro de 0001 a 31 de dezembro de 9999, armazena 3 bytes.
5. **time**: Armazene um tempo apenas com uma precisão de 100 nanossegundos, armazena de $3-5$ bytes

6. **datetimeoffset**: O mesmo que datetime2 com a adição de um deslocamento de fuso horário, armazena de $8 - 10$ bytes.

7. **timestamp**: Armazena um número exclusivo que é atualizado sempre que uma linha é criada ou modificada. O valor do timestamp é baseado em um relógio interno e não corresponde ao tempo real. Cada tabela pode ter apenas uma variável timestamp.

1.4 Sintaxe SQL

SQL não diferença maiúsculas de minúsculas, **SELECT** é o mesmo **select**. Neste documento se escrevem todas as palavras-chaves SQL em maiúsculas.

O ponto e vírgula é a maneira padrão de separar cada instrução SQL em sistemas de banco de dados que permitem que mais de uma instrução SQL seja executada na mesma chamada ao servidor. Aqui se usa, ponto e vírgula no final de cada instrução SQL.

Em SQL server, cada instrução é executada com **fn + F5** ou **F5** e para inserir comentários de escreve dois hifens seguidos, $--$.

Alguns dos comandos do SQL mais importantes são:

- SELECT, extrai dados de um banco de dados
- UPDATE, atualiza dados em um banco de dados
- DELETE, exclui dados de um banco de dados
- INSERT INTO, insere novos dados em um banco de dados
- CREATE DATABASE, cria um novo banco de dados
- ALTER DATABASE, modifica um banco de dados
- CREATE TABLE, cria uma nova tabela
- ALTER TABLE, modifica uma tabela
- DROP TABLE, exclui uma tabela
- CREATE INDEX, cria um índice (chave de pesquisa)
- DROP INDEX, exclui um índice

1.4.1 Criação de base de dados e tabelas em SQL server

Para criar uma base de dados se deve acessar e fazer conexão a **SQL server Management Edition**, executar Ctrl +
$textN$ onde se abre um script. Assim, se cria a base de dados com a instrução,

```
CREATE TABLE SQL_BD_01;
```

Posteriormente, se devem criar as tabelas iniciando com as de dimensões (aquelas que não tem FOREIGN KEY),

```
CREATE TABLE Aluno
(id_aluno       int PRIMARY KEY NOT NULL,
 nome_aluno     varchar(100)    NOT NULL,
 data_nacimento date            NOT NULL,
 sexo           varchar(1)      NOT NULL, -- M masculino e F femenino
 data_cadastro  datetime        NOT NULL,
 login_cadastro varchar(20)     NOT NULL
);
```

No comando acima, a tabela **Aluno** é criada com seis (6) campos. A coluna $student_id$ é uma chave primária inteira, enquanto $student_name$ é um vetor string com 100 caracteres possíveis, datas também são incluídas, no caso do campo textbf $date_birth$ é uma única data permitida e $date_cadastro$ mais de uma data para possíveis atualizações de data e hora. Fica estabelecido que todos os campos não são nulos (NOT NULL) como medida lógica para análise dos dados.

1.4.2 Constraint - Criação em SQL server

Esta se refere a um comando ou instrução para criação de uma *restrição*(constraint) na base de dados. O que implica, colocar limitações para obter regras que sejam respeitadas pelo relacionamento entre as tabelas de um banco de dados. Esta se usa para relacionar tabela através das chaves estrangeiras (**FK**), veja

```
CONSTRAINT fk_nome
FOREIGN KEY (campo_tb_atual)
REFERENCES tb_de_ligação (id_campo_tb_ligação)
```

Uma prática recomendada é primeiro criar todas as tabelas no banco de dados e, em seguida, inserir as chaves secundárias necessárias. Para relacionar tabelas pode-se utilizar o comando **ALTER TABLE** e a instrução **ADD** que permite adicionar a instrução que se deseja alterar, para evitar ter que deletar ou criar tabelas repetidas. Lembre-se que este processo deve ser feito em tabelas que por definição incluam FOREIGN KEY.

```
ALTER TABLE nome_tabela
 ADD CONSTRAINT fk_nome FOREIGN KEY (campo_tb_atual) REFERENCES tb_de_ligação (id_campo_tb_liga
```

Para deletar uma **CONSTRAINT** se pode usar a seguinte instrução,

```
ALTER TABLE nome_tabela
 DROP CONSTRINT fk_nome;
```

No entanto, como se faz na prática a específicação de FOREIGN KEY? Para cada tabela se pode dar a intrução de alterar a tabela **atual** (ALTER TABLE $nome_tabela_atual$) depois de escreve o comando de restrição (ADD). Segui a restrição requerida que vai relacionar a chave estrangeira da tabela atual com a chave primára em outra tabela. A instrução CONSTRAINT fk_nome, indica o nome da chave secundária, FOREINGN KEY ($campo_atual$) se refere ao nome do campo que é chave extrangeira, e REFERENCES $tb_ligacao$ ($id_campo_tb_ligacao$) diz o nome da tabela e o campo (chave primária) que vai se relacionar. Veja um exemplo, com a chave secundária id_Aluno na tabela Turmas,

Porém, como é feita a especificação de FOREIGN KEY na prática? Para a tabela atual que contém a chave estrangeira, se pode dar o comando para modificar a tabela **atual** (ALTER TABLE $current_table_name$) depois de escrever o comando de restrição (ADD). Seguidamente, a restrição necessária que relacionará a chave estrangeira da tabela atual com a chave primária em outra tabela. A instrução CONSTRAINT fk_nome indica o nome da chave secundária, FOREINGN KEY ($current_field$) refere-se ao nome do campo que é uma chave estrangeira e REFERENCES $tb_ligacao$ ($id_campo_tb_ligacao$) indica o nome da tabela e o campo (chave primária) a ser correspondido. Veja um exemplo, com a chave secundária id_Aluno na tabela Turmas,

```
ALTER TABLE turmas
 ADD CONSTRAINT fk_alunostur FOREIGN KEY (id_alunos) REFERENCES Alunos (id_alunos);
```

1.4.3 INSERT - Inserindo dados nas tabelas

A instrução para inserir dados nas tabelas é dada por,

```
INSERT INTO nome_tabela
(campos da tabela)
VALUES
(valores para insert)
```