Notas Curso SQL

April 16, 2022

1 Normalização de dados

Chamamos de normalização de dados o processo formal e passo a passo que examina o documento descritivo gerado pelos analistas de sistemas durante a análise de requisitos em busca de definir as entidades, atributos, relacionamentos, chaves primárias e chaves estrangeiras do banco de dados a ser modelado.

Este processo é realizado utilizando regras bem estabelecidas conhecidas como Formas Normais, definidas por Edgar Frank Codd em seu artigo.

Um dos objetivos principais da normalização é evitar ou pelo menos amenizar anomalias e inconsistências que podem ocorrer durante a inclusão, exclusão, alteração e consulta de registros em um banco de dados.

Um banco de dados normalizado dentro dos padrões reduz o trabalho de manutenção e ajuda a evitar o desperdício de espaço de armazenamento, dentre outros benefícios.

1.1 Primeira forma normal (1FN)

O processo de normalização aplica uma série de regras sobre as tabelas de um banco de dados para verificar se estas estão corretamente projetadas.

Embora existam cinco formas normais (ou regras de normalização), na prática usamos um conjunto de três formas normais, ou seja, um banco de dados é considerado normalizado se nele foram aplicadas as regras destas três formas normais.

Uma entidade, ou tabela, estará na primeira forma normal (1FN) se todos os campos forem atômicos (simples) e não multivaloreados (com múltiplos valores).

Podemos observar que os campos telefone e endereço são multivalores, enquanto que os demais são definidos como atômicos. Segundo a 1FN, todos campos devem ser atômicos e não multivalorados. Mas como resolver isto? Devemos criar novos campos que irão agregar estas informações que estão sobrando. Com isso, a nossa tabela ficará como

Ponto importante: devemos evitar ao máximo criar colunas que levarão à campos vazios. Seguindo essa lógica, como vamos tratar a coluna dos telefones? Pois podemos observa que alguns elementos possuem dois telefones, enquanto que outros possuem apenas um. Se formos pela para a criação de dois colunas telefone1 e telefone2, algumas colunas ficarão vazias, já que alguns clientes possuem apenas 1 telefone e uma tabela que aceita valores vazios é uma tabela problemática. Todos os campos devem ser importantes para estar na tabela e campos importantes não podem ficar vazios. Da mesma forma, imagina que o cliente possui 3 telefones? Ou até mesmo nenhum? Como seriam armazenados?

Neste caso, a solução é criar uma nova tabela!

Agora, temos que a nossa tabela está normalizada com a 1FN, agora seguiremos com a segunda forma normal 2FN. Vale lembrar que para aplicar a 2FN, devemos obrigatoriamente ter passado pela 1FN primeiro.

Figure 1: Podemos observar que o campo telefone e endereço são multivalorados.

Figure 2: Nova tabela, agora, com os campos atômicos.

Figure 3: Tabela normalizada de acordo com a 1FN

Quais os problemas de uma tabela não normalizada com a 1FN?

São vários. A primeira forma normal tenta resolver um dos maiores problemas de bancos de dados que é a repetição (redundância de dados) e a desorganização deles.

Imagine um campo de telefone que permita a entrada de mais de um valor (dois números de telefone) por exemplo. Como faríamos uma busca em um dos valores apenas?

Mesma coisa em um campo endereço onde as partes não estivessem desmembradas, da seguinte forma:

Rua das Oliveiras, 256, Parque Novo Mundo, São Paulo, SP.

Como seria possível fazer uma busca de todos os clientes que morassem no Parque Novo Mundo? Ou na cidade de São Paulo? Ou no estado de São Paulo?

Outra pergunta é se toda a tabela precisa obrigatoriamente ser normalizada com a 1FN?

Não, a normalização é um processo corretivo que deve ser aplicado em casos específicos onde o problema for identificado. Tudo irá depender de como a análise dos dados foi feita. De início você terá muita dificuldade em aplicar as regras de normalização e somente o tempo e o acúmulo de experiência farão esse processo ser natural para você. Um analista experiente aplica a normalização de dados por padrão, pois ele olha para uma tabela e já sabe que tem algo errado ali e aplica a correção para tal.

1.2 Segunda forma normal (2FN)

Uma entidade estará na 2FN se ela já se encontrar na 1FN e todos os atributos não chave foram totalmente dependentes da chave primária.

Qual é o problema aqui? Podemos analisar cada atributo: o primeiro atributo que é a chave primária da tabela é o nº do pedido, o segundo é o código do produto, o terceiro é o produto, o quarto é a quantidade de produto, o quinto o valor unitário do produto e, por último, o sexto é o subtotal (quantidade*valor_unitário). Podemos observar o seguinte problema, o nome do produto não parece estar totalmente associado com o n° do pedido, mas sim com o código do produto, o que fere o princípio para a entidade estar na 2FN. Então, para resolver este problema, basta que criemos uma outra tabela separada, que se relaciona com a primeira, como mostrado abaixo

Pronto! Nossa tabela está dentro da 2FN e pronta para ser validade com a 3FN.

Veja que a partir deste momento, o código do produto da tabela de pedidos faz referência à chave primária da tabela de produtos, portanto há um relacionamento entre as tabelas.

Outro ponto é que o valor unitário deveria/poderia estar na tabela produtos e não na tabela pedidos, mas resolveremos isso posteriormente estudando a 3FN.

1.3 Terceira forma normal (3FN)

Cada uma das formas normais tende a ir refinando a modelagem e deixando a estrutura de dados mais íntegra e exclusiva, evitando repetições desnecessárias e possíveis sobrecarga no gerenciador de banco de dados.

Uma tabela estará na 3FN se ela estive na 2FN e se nenhuma coluna não-chave depender de outra coluna não-chave. Ou seja, na 3FN temos simplesmente que eliminar os campos que podem ser obtidos pela equação de outros campos na mesma tabela. Podemos visualizar novamente na tabela pedidos,

Figure 4: Exemplo: Tabela pedidos

Figure 5: Criada a tabela de produtos a partir da tabela pedidos, onde ambas obecem a 2FN.

Figure 6: Exemplo: Tabela pedidos normalizada de acordo com a 3FN.

o atributo Subtotal é completamente desnecessário de ser mantido, pois ele é simplesmente o produto entre o valor unitário e a quantidade do produto solicitada, portanto, devendo ser eliminado.

O valor unitário também é um atributo associado à chave código_produto, portanto, ele deve estar associado a tabela produtos e não a tabela pedidos. Resolvendo isso, teremos

1.4 Outras formas normais

1.4.1 Quarta forma normal (4FN)

Uma entidade estará na 4FN se estiver na 3FN e na tabela não conter múltiplas entradas multivaloradas (valores repetidos em diferentes colunas), ou seja, a presença de uma ou mais linhas em uma tabela implica na presença de uma ou mais outras linhas na mesma tabela.

Exemplo 1: Considere uma tabela que armazene informações sobre planos de saúde e exames de um paciente: Paciente; Plano; Exame;

Separando as tabelas, de modo a evitar a repetição, temos

logo, a tabela paciente se encontrará com a quatro formas normais.

1.4.2 Quinta forma normal (5FN):

Uma entidade estará na 5FN se estiver na 4FN e quando um atributo está em outra tabela sem a necessidade de estar na tabella pesquisada e pode ser removido sem a perda de nenhuma informação.

2 Linguagem SQL

Já sabemos que SQL significa Structured Query Language ou Linguagem de Consulta Estruturada. Mais importante que isso, é que a SQL é a linguagem padrão dos Bancos de Dados Relacionais. Não por acaso, grande parte dos bancos de dados utilizam a sigla da linguagem no seu próprio nome:

1. MySQL
2. PostgreSQL
3. Microsoft SQL Server;
4. SQLite;
5. Dentro outros..

A linguagem SQL foi inspirada na álgebra relacional e ela se subdivide em 5 subgrupos:

DQL, DML, DDL, DCL, DTL. Todos são subgrupos da linguagem SQL. Mas mais à frente teremos um pouco mais de contato com cada um deles. Mas aqui, teremos uma pequena introdução sobre:

DQL: Data Query Language: ou Linguagem de Consulta de Dados, ou os comandos utilizados para fazer pesquisa em SQL.

DML: Data Manipulation Language, ou Linguagem de Manipulação de Dados, ou os comandos para alterar os dados.

Figure 7: Tabela pedidos e tabela produtos normarlizadas de acordo com as 3 formas normais.

Figure 8: Tabela Paciente.

Figure 9: Tabela Paciente de acordo com as 4FN.

DDL: Data Definition Language, ou Linguagem de Definição de Dados, ou os comandos para definir os tipos de dados que trabalharemos.

DCL: Data Control Language, ou Linguagem de Controle de Dados, ou os comandos utilizados para controlar o acesso aos dados.

DTL: Data Transaction Language, ou Linguagem de Transação de Dados, os comandos que compreende tudo em relação à transação dos dados no banco de dados.

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 pode ser de sucesso ou de erro.

OBS: Apesar de não ser obtrigatório, costumamos a escrever os comandos SQL todos em letra maiúscula, o que ajuda a entender melhor o nosso código já que o que for referente aos comandos SQL estarão destacados em maiúsculos e o que for referente aos dados/tabelas e etc estarão em minúsculo.

2.1 Data Query Language

No subgrupo DQL nós temos apenas 1 comando SQL: Select.

Este comando é utilizado para realizar consultas no banco de dados.

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. Este comando é composto de várias cláusulas e opções, possibilitando elaborar consultas das mais simples às mais complexas.

A exemplo de como fazer estas pesquisas, podemos considerar a tabela acima.

- SELECT * FROM tipos_produto;

Neste caso, estamos selecionando todos os dados da tabela "tipos_produto". O asterisco indica que queremos os dados de todos os campos da tabela.

- SELECT codigo, descricao FROM tipos_produto;

Neste caso, funcionará da mesma forma que utilizando o asterisco.

- SELECT codigo, descricao, codigo_tipo FROM produtos;

No exemplo acima, estamos selecionando todos os dados de apenas 3 colunas da tabela 'produtos'. Vej aque não especificamos a coluna 'preco'. Desta forma, não teremos estes dados.

- SELECt cod, desc, pre, ctp FROM produtos;

Neste caso, estamos selecionando dados de colunas inexistentes na tabela 'produtos', isso além de não trazer, como resultado, nenhum dado, ainda nos apresentará erro pois as colunas solicitadas não existem na tabela.

Podemos colocar um alias (apelido) em nome de tabela e campos.

Figure 10: Tabela produto e tipo de produto utilizado no exemplo.

- SELECT p.codigo AS cod, p.descricao AS desc, p.preco AS pre, p.codigo_produto AS ctp FROM produtos AS p;

Neste caso, estamos realizando uma busca especificando os campos que queremos da tabela 'produtos' e adicionando um alias tanto para o nome da tabela quanto para o nome dos campos.

2.2 Data Manipulation Language

Os comando presentes neste subgrupo é definido como: 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.

Entendendo o funcionamento do comando Insert.

Considerando o caso da tabela produtos e tipos_produtos anteriormente, consideramos

- INSERT INTO tipos_produto (descricao) VALUES ('Notebook');

Com isso, um novo registro será inserido na tabela e uma mensagem de sucesso será retornada.

- INSERT INTO produtos (descricao, preco, codigo_tipo_produto) VALUES ('Notebook', 1200, 1);

No exemplo acima, estamos inserindo os valores 'Notebook, 1200 e 1' na tabela 'produtos'. O campo 'codigo' do produto é chave primária e auto incremento, então será inserido automaticamente.

Entendendo o funcionamento do comando Update.

- UPDATE tipos_produto set descricao='Nobreak' WHERE codigo=3;

Com isso, o registro com código=3 será atualizado com o novo valor para o campo 'descricao'.

- UPDATE produtos set descricao='Notebook', preco=2800 WHERE codigo 20;

No exemplo acima, estamos atualizando um registro na tabela 'produtos' e note que estamos utilizando o filtro com a cláusula WHERE especificando qual registro queremos atualilzar. Além disso, estamos atualizando dois campos da tabela. Desta forma, o registro será atualilzado e o novo valor será mantido na tabela.

Entendendo o funcionamento do comando Delete

DELETE FROM tipos_produto WHERE codigo=3;
O registro será removido da tabela, sem chances de recuperação.
ATENÇÃO: É importante lembrar da importância da utilização do WHERE na utilização dos comando DELETE e UPDATE, tendo em vista que caso esses sejam realizados sem a especificação, eles tendem a aplicar o comando para toda a base de dados. Logo, se você fizer uma substituição pelo comando UPDATE, sem especificar com o WHERE, ele substituirá todos os dados na base de dados, assim como o DELETE irá deleter todos os dados na base de dados.