Relatório de Normalização de Dados

Introdução

Este relatório apresenta os resultados do processo de normalização de dados realizado no conjunto de dados das tabelas de vendas/itens de vendas . O objetivo desta normalização foi melhorar a qualidade dos dados e torná-los mais adequados para análises futuras.

Analise dos problemas observados referente a obediência a 1FN, 2FN, 3FN:

Verificando os dados das duas tabelas , podemos observar que os atributos ‘tipo\_pagamento’ contêm múltiplos valores e não atendem à 1ª Forma Normal.

Além disso, a chave primária é composta apenas pelo "ID ", mas o atributo "tipo\_pagamento" é dependente funcionalmente dos atributos "numero\_cartao\_pagamento" , "numero\_parcelas\_pagamento" e “data\_pagamento”. Isso viola a 2ª Forma Normal, pois esses atributos não são completamente dependentes da chave primária.

Por fim, a existência de dependências transitivas entre os atributos "valor\_total", "subtotal", "valor\_unitario" e "Quantidade" viola a 3ª Forma Normal, pois o atributo "valor\_total" é dependente funcional do atributo "valor\_unitario" e “quantidade” , e não diretamente da chave primária.

Dando sequencia aos problemas verificados , a coluna "nome\_produto" na tabela "item\_venda" é redundante, pois o nome do produto já é armazenado na tabela "produto", que é referenciada pela coluna "produto\_id". Isso pode levar a problemas de inconsistência de dados, já que se o nome do produto for alterado na tabela "produto", a coluna "nome\_produto" na tabela "item\_venda" não será atualizada automaticamente.

A coluna "numero\_cartao\_pagamento" na tabela "venda" pode não ser suficiente para armazenar todos os tipos de informações de pagamento, dependendo do tipo de cartão. Além disso, essa coluna pode ser redundante com outras informações de pagamento, como o número do pedido ou o número da nota fiscal.

A coluna "tipo\_pagamento" na tabela "venda" é um char(1), o que significa que só pode armazenar um caractere. Isso pode ser insuficiente para determinados tipos de pagamento que possuem códigos maiores, como o PIX, entre outros.

A tabela "venda" não possui uma coluna para armazenar o endereço de entrega do cliente. Dependendo da complexidade do sistema, isso pode ser um problema, pois pode dificultar a identificação do local de entrega da mercadoria.

Soluções Propostas

Para normalizar essas tabelas, podemos seguir as etapas do processo de normalização até alcançar a terceira forma normal (3FN). Seguem abaixo as sugestões de normalização para as tabelas "venda" e "item\_venda":

1. Remover a coluna "valor\_total", que pode ser calculada a partir dos valores dos itens de venda.

2.Criar uma nova tabela para armazenar as informações de pagamento e incluir uma chave estrangeira na tabela "venda" que faça a referência a essa nova tabela. Dessa forma, será possível associar uma ou várias informações de pagamento a cada venda realizada.

3.Criar uma nova tabela para armazenar as informações de envio (data\_envio e status) e incluir uma chave estrangeira na tabela "venda" que faça a referência a essa nova tabela. Dessa forma, será possível associar uma ou várias informações de envio a cada venda realizada.

4.Adicionar uma nova tabela para armazenar os endereços de entrega dos clientes e incluir uma chave estrangeira na tabela "venda" que faça a referência a essa nova tabela. Dessa forma, será possível associar um endereço de entrega a cada venda realizada.

5.Alterar a coluna "tipo\_pagamento" para um tamanho maior que possa comportar códigos maiores, como varchar(3) ou char(3), dependendo do tamanho máximo do código.

APOS AS ALTERAÇÕES PROPOSTAS ACIMA, A TABELA "VENDAS" FICARIA ASSIM :

CREATE TABLE IF NOT EXISTS `uc4atividades`.`venda` (

`id` INT NOT NULL AUTO\_INCREMENT,

`data` DATETIME NULL DEFAULT NULL,

`cliente\_id` INT NOT NULL,

`funcionario\_id` INT NOT NULL,

PRIMARY KEY (`id`),

INDEX `cliente\_id` (`cliente\_id` ASC) VISIBLE,

INDEX `funcionario\_id` (`funcionario\_id` ASC) VISIBLE,

CONSTRAINT `venda\_ibfk\_1`

FOREIGN KEY (`cliente\_id`)

REFERENCES `uc4atividades`.`cliente` (`id`),

CONSTRAINT `venda\_ibfk\_2`

FOREIGN KEY (`funcionario\_id`)

REFERENCES `uc4atividades`.`funcionario` (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `uc4atividades`.`pagamento` (

`id` INT NOT NULL AUTO\_INCREMENT,

`tipo` VARCHAR(45) NULL DEFAULT NULL,

`numero\_cartao` VARCHAR(16) NULL,

`numero\_parcelas` INT NULL,

`data\_pagamento` DATETIME NULL,

`venda\_id` INT NOT NULL,

PRIMARY KEY (`id`),

INDEX `venda\_id` (`venda\_id` ASC) VISIBLE,

CONSTRAINT `pagamento\_ibfk\_1`

FOREIGN KEY (`venda\_id`)

REFERENCES `uc4atividades`.`venda` (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

CREATE TABLE IF NOT EXISTS `uc4atividades`.`envio` (

`id` INT NOT NULL AUTO\_INCREMENT,

`data\_envio` DATETIME NULL DEFAULT NULL,

`status` VARCHAR(45) NULL DEFAULT NULL,

`venda\_id` INT NOT NULL,

PRIMARY KEY (`id`),

INDEX `venda\_id` (`venda\_id` ASC) VISIBLE,

CONSTRAINT `envio\_ibfk\_1`

FOREIGN KEY (`venda\_id`)

Resultados

Após a normalização dos dados, o conjunto de dados apresentou uma melhora significativa na qualidade e na adequação para análises futuras. Os dados numéricos estão agora em escala comparável, enquanto os dados categóricos foram convertidos em uma forma mais adequada para análise. Além disso, a normalização dos dados textuais melhorou a qualidade do texto para análise de sentimentos e análise de tópicos.

Conclusão

A normalização de dados é uma etapa importante para melhorar a qualidade e adequação dos dados para análises futuras. Neste relatório, apresentamos a metodologia e os resultados do processo de normalização de dados realizado no conjunto de dados [nome do conjunto de dados]. Esperamos que este relatório ajude a fornecer informações valiosas para análises futuras dos dados normalizados