EXEMPLO PRÁTICO DE NORMALIZAÇÃO 1FN, 2FN e 3FN

NORMA 1FN

1FN

Para estar de acordo com a norma 1fn é necessário que todos os dados sejam atômicos, ou seja, cada célula deve conter um único valor, e não deve haver repetições ou conjuntos de valores.

Na tabela abaixo temos um problema**, o Colaborador de eventos, pode ter mais de 1 telefone**, e isso gera um desacordo com a norma 1fn, que não permite conjuntos de valores.

ColaboradorID	Nome	Cargo	Email1	Email2
1	Jonilton	Fotógrafo	jonilton@gmail.com	
2	Wesley	Músico	wesley@gmail.com	wesley_2023@gmail.com
3	Carmelita	Decorador	carmelita@hotmail.com	carmelita_ok@hotmail.com

Código SQL da tabela:

```
CREATE TABLE Colaboradores (
    ColaboradorID INT PRIMARY KEY,
    Nome VARCHAR(100),
    Cargo VARCHAR(50),
    Email1 VARCHAR(100),
    Email2 VARCHAR(100)
);
```

Para resolver este problema, devemos dividir as tabelas criando uma tabela de contato dos profissionais para armazenar os telefones, o que permite que o Colaborador possa ter armazenados o número de telefones que forem necessários, esta solução é a mesma para campos tipo email, endereço, cursos, quando são necessários ser armazenados mais de 1 valor.

Tabela Colaboradores

ColaboradorID	Nome	Cargo
1	Jonilton	Fotógrafo
2	Wesley	Músico
3	Carmelita	Decorador

Tabela Contato

EmailID	Email	ColaboradorID
1	jonilton@gmail.com	1
2	wesley@gmail.com	2
3	wesley_2023@gmail.com	2
4	carmelita@hotmail.com	3
5	carmelita_ok@hotmail.com	3

Código SQL da tabela:

```
DROP TABLE IF EXISTS colaboradores;

CREATE TABLE Colaboradores (
    ColaboradorID INT PRIMARY KEY,
    Nome VARCHAR(100),
    Cargo VARCHAR(50)
);

CREATE TABLE Contatos (
    EmailID INT PRIMARY KEY,
    Email VARCHAR(100),
    ColaboradorID INT,
    FOREIGN KEY (ColaboradorID) REFERENCES Colaboradores(ColaboradorID)
);
```

NORMA 2FN

2fn – para estar de acordo com a norma 2fn a tabela precisa estar primeiro de acordo com a 1fn, **e todos os atributos não chave devem ser dependentes das chave primárias**, caso composta, ser dependente de todas as chaves.

ColaboradorID	Eventoid	NomeEvento	DuraçaoEvento	ValorHora	Total
1	1	Casamento	10	50,00	500,00
2	1	Casamento	10	75,00	750,00
3	2	Formatura	7	80,00	560,00

Na tabela acima temos um problema em que a chave primárias é composta das chaves ColaboradorID e EventoId e todos os campos deveriam ser dependentes destas duas chaves, e na verdade isso não está acontecendo, os campos NomeEvento e Duração evento, são dependentes da chave Eventoid e os campos ValorHora e Total fazem referência ao ColaboradorID, pois é o valor que o mesmo receberá ao final do evento.

Código SQL da tabela:

```
CREATE TABLE Pagamento (
    ColaboradorID INT,
    EventoID INT,
    NomeEvento VARCHAR(100),
    DuracaoEvento INT,
    ValorHora DECIMAL(10, 2),
    Total DECIMAL(10, 2),
    PRIMARY KEY (ColaboradorID, EventoID)
  );
```

Para resolver este problema, devemos dividir as tabelas criando uma tabela com os dados dependentes de ColaboradorID e outra dos dados dependentes de EventoId

Tabela Pagamento

ColaboradorID	ValorHora	Total
1	50,00	500,00
2	75,00	750,00
3	80,00	560,00

Tabela Evento

EventoID	NomeEvento	DuraçaoEvento
1	Casamento	10
1	Casamento	10
2	Formatura	7

Código SQL da tabela:

```
DROP TABLE IF EXISTS Pagamento;

CREATE TABLE Pagamento (
    ColaboradorID INT PRIMARY KEY,
    ValorHora DECIMAL(10, 2),
    Total DECIMAL(10, 2),
    FOREIGN KEY (ColaboradorID) REFERENCES Colaboradores(ColaboradorID)
);

CREATE TABLE DadosEvento (
    EventoID INT PRIMARY KEY,
    ColaboradorID INT,
    NomeEvento VARCHAR(100),
    DuracaoEvento INT,
    FOREIGN KEY (ColaboradorID) REFERENCES Pagamento(ColaboradorID)
);
```

NORMA 3FN

3fn

Para que uma tabela possa estar ajustada na terceira forma normal (3FN), além de ser obrigatório obedecer à forma normal anterior (2FN), **não pode existir nenhuma dependência transitiva, ou seja, nenhuma coluna não chave poderá depender de outra coluna que seja não chave.**

ColaboradorID	Eventoid	NomeEvento	DuraçaoEvento	ValorHora	Total
1	1	Casamento	10	50,00	500,00
2	1	Casamento	10	75,00	750,00
3	2	Formatura	7	80,00	560,00

Já vimos que para colocar esta tabela na 2fn devemos dividir a mesma em duas novas tabelas, o problema é que o campo Total, continua dependendo dos campos DuraçãoEvento e ValorHora, ou seja, é a multiplicação dos dois, gerando um valor final, o que cria uma dependência transitiva onde um campo não chave, depende de colunas que também são não chave, para resolver isso devemos criar uma tabela onde o id do colaborador precisa estar presente, outra forma, mais para visualização do resultados, seria uma View(tabela virtual), e com esta nova tabela solucionamos o problema da 3fn, e a parte de cálculo ficaria para o código, por exemplo, em Java.

Tabela

ColaboradorID	Eventoid	NomeEvento	DuraçaoEvento	ValorHora
1	1	Casamento	10	50,00
2	1	Casamento	10	75,00
3	2	Formatura	7	80,00

Tabela Solução: ajustando as dependências, agora o valor hora se tornou dependente apenas da chave primária e o evento id se torna uma chave estrangeira.

ColaboradorID	ValorHora	Eventoid
1	50,00	1
2	75,00	1
3	80,00	2

```
CREATE TABLE NovoPagamento (
    ColaboradorID INT PRIMARY KEY,
    ValorHora DECIMAL(10, 2),
    EventoID INT,
    FOREIGN KEY (ColaboradorID) REFERENCES Colaboradores(ColaboradorID),
    FOREIGN KEY (EventoID) REFERENCES DadosEvento (EventoID)
);
```

Obs: a solução também pode ser a View abaixo:

View TotalPago

ColaboradorID	ValorRecebido
1	500,00
2	750,00
3	560,00

Cógido SQL da View:

```
CREATE VIEW TotalPago AS
SELECT np.ColaboradorID, (np.ValorHora * de.DuracaoEvento) AS ValorRecebido
FROM NovoPagamento np
JOIN DadosEvento de ON np.ColaboradorID = de.ColaboradorID;
```