

1º Trabalho Prático

## Sistemas de informação II

Licenciatura Engenharia Informática e de Computadores

**Docente:** Afonso Remédios

**Grupo 7:**

Pedro Miranda - 43584

Ricardo Rodrigues – 43594

Índice

[Introdução 3](#_Toc57997756)

[Problema Proposto 4](#_Toc57997757)

[Modelo Entidade-Associação 5](#_Toc57997758)

[Modelo Relacional 6](#_Toc57997759)

[Avaliação Experimental 8](#_Toc57997760)

[Exercício A 8](#_Toc57997761)

[Exercício B 8](#_Toc57997762)

[Exercício C 8](#_Toc57997763)

[Exercício L 8](#_Toc57997764)

[Conclusão 10](#_Toc57997765)

[WebGrafia 10](#_Toc57997766)

# Introdução

Neste trabalho tratamos de resolver um problema através da utilização de base dados, para tal, será necessário usar o mecanismo transacional das bases de dados. Este trabalho será resolvido na linguagem de programação de bases de dados *SQL*.

O mecanismo transacional tem um controlo que permite executar várias interações com a base de dados ao mesmo tempo, utilizando diferentes transações. Uma transação é um conjunto de instruções que irão ser usadas numa base de dados. Para que estas (leitura e escrita) sejam compatíveis podendo se executar simultaneamente, terão de ser principalmente atómicas, entre outras coisas. Para não corromperem a base de dados, também tem de ser consistentes, isto é, isoladas uma das outras de forma a garantir essa mesma consistência, têm de ter resultados perseverantes, tanto no caso de um *commit* como de um *rollback*.

As transações têm como objetivo preservar a integridade e a consistência de uma base de dados, isto é, mecanismos de recuperação em caso de falhas do sistema.

Os escalonamentos são o que nos permite utilizar várias transações e executá-las na mesma ordem. Um conflito num escalonamento acontece quando as operações pertencem a transações diferentes. Estes conflitos geram anomalias, que são variadas como *Dirty Read* ocorrendo quando uma transação consegue ler dados resultantes de escritas que ainda não deram *committed* por outra transação. A leitura não repetitiva (*Non Repeatable Read)*, isto é, quando uma leitura demonstra resultados diferentes, após ser executada novamente, mais tarde na mesma transação. Para finalizar temos a leitura fantasma (Phantom Read), que ocorre quando em duas leituras diferentes de uma transação, há uma alteração nos registos, e essas alterações estão numa leitura, mas na seguinte já não, na mesma transação.

De modo a evitar estas anomalias, são usados níveis de isolamento. Começando pelo nível de isolamento mais “baixo”, isto é, o *Read Uncommitted* que permite que uma transação manipule valores que não são sujeitos a *commit* por outras transações. O nível a seguir é o *Read Committed*, que deixa que uma transação leia e manipule dados que já sofreram *commit* por outras transações. De seguida temos o *Repeatable Read*, que garante que a mesma leitura de um dado se repita, originando resultados iguais para diferentes execuções na mesma transação. Para finalizar, o nível mais alto e seguro é o *Serializable*, que apesar de ser parecida à anterior, esta também garante que as linhas selecionadas por uma transação não podem ser alteradas/lidar por outra transação, até que a primeira transação seja concluída.

# Problema Proposto

“A empresa FSolv pretende desenvolver um sistema de informação para a gestão de faturas não simplicadas. O sistema tem de ser aditável. Uma fatura é identificada por um código que segue o formato: **FTyyyy-xxxxx**, onde yyyy representa o ano e xxxxx representa o número da fatura

emitida num ano. Mesmo que uma fatura seja anulada, o valor de xxxxx é monótono crescente.

Cada fatura é caracterizada por uma data de criação, uma data de emissão (eventualmente nula), um estado, um valor total (sem IVA) e o valor de IVA a pagar. Ambas as datas têm, pelo menos, resolução ao segundo. O valor total e o valor de iva são resultados dos valores associados dos itens existentes na fatura. Cada item é identificado por um número dentro da fatura e tem uma descrição. É possível definir para cada item da fatura um desconto e o número de unidades presentes na fatura. Considere que uma fatura pode ter os seguintes estados:

• Emitida, impossibilitando posteriores alterações à fatura;

• Em atualização, quando ainda não está finalizada (e.g. falta adicionar itens);

• Proforma, impossibilitando qualquer alteração à fatura, exceto a passagem para os estados Emitida ou Anulada;

• Anulada, impossibilitando posteriores alterações à fatura;

Quando é necessário devolver items presentes numa fatura, deve ser criada uma nota de crédito. As notas de crédito são semelhantes a faturas, exceto nos itens que a constituem, que podem ser um subconjunto não vazio dos itens da fatura anulada (e referenciada na nota de crédito).

Além disso, a identificação da uma nota de crédito \_e dado por **NCyyyy-xxxxx**, seguindo a mesma lógica da identificação das faturas. Uma nota de crédito só pode ter os estados “Em atualização e Emitida”. Neste último caso são impossíveis alterações \_a nota de crédito.

Os itens de uma fatura estão associados a produtos. Cada produto \_e identificado por um SKU, tem uma descrição, uma percentagem de iva (eventualmente nula), e o preço de venda unitário (sem iva). Deve ser possível registar o contribuinte associado a uma fatura. Um contribuinte tem um número de identificação fiscal, um nome (eventualmente nulo) e uma morada (eventualmente nula). Sempre que é feita uma alteração a uma fatura, por exemplo, uma mudança de estado, deve ficar registado no sistema a data em que a alteração foi feita, qual o estado antes da alteração (eventualmente o mesmo) “

# Modelo Entidade-Associação

Figura - Modelo Entidade-Associação

# Modelo Relacional

**Pk:** Primary Key (Chave Primária)

**FK:** Foreingn Key (Chave estrangeira)

**RI:** Restrições de Integridade

**Estado\_FAT** (estado\_ft)

PK: estado\_ft

**Estado\_NC** (estado\_nc)

PK: estado\_nc

**Codigo\_NotaCred** (ano, nr\_nc)

PK: (ano, nr\_nc)

**Codigo\_Fatura**(ano, nr\_nc)

PK: (ano, nr\_nc)

**Contribuinte** (nif, nome, morada)

PK: nif

**Fatura** (codigo\_fat, ano, nr\_fat, dt\_emissao, dt\_criacao, val\_total, val\_iva, estado, nif)

PK: codigo\_fat

FK: (estado) referencia Estado\_FAT (estado\_ft)

(ano, nr\_fat) referencia Codigo\_Fatura (ano,nr\_fat)

(nif) referencia Contribuinte (nif)

RI: dt\_emissao tem de ser maior que a dt\_criacao.

**Nota\_Cred** (codigo\_nc, ano, nr\_nc, dt\_emissao, dt\_criacao, val\_nc, estado, codigo\_fat)

PK: codigo\_nc

FK: (estado) referencia Estado\_NC (estado\_nc)

(ano, nr\_nc) referencia Codigo\_NoTaCred (ano, nr\_nc)

(codigo\_fat) referencia Fatura (codigo\_fat)

RI: dt\_emissao tem de ser maior que a dt\_criacao.

**Produto** (sku ,desc\_prod, prec\_iva, prec\_unit)

PK: sku

**Item** (num\_item, codigo\_fat, sku, desc\_item, desconto, num\_uni)

PK: (num\_item, codigo\_fat)

FK: (codigo\_fat) referencia Fatura (codigo\_fat)

(sku) referencia Produto (sku)

**Item\_NC** (quantidade, codigo\_nc, num\_item, codigo\_fat)

PK: quantidade

FK: (codigo\_nc) referencia Nota\_Cred (estado\_nc)

(num\_item, codigo\_fat) referencia Item (num\_item, codigo\_fat)

**Fatura\_Hist** (dt\_atualizacao, ultimo\_estado, dt\_emissao, dt\_criacao, val\_total, val\_iva, codigo\_fat, nif)

PK: dt\_atualizacao

FK: (codigo\_fat) codigo\_fat referencia Fatura (codigo\_fat)

nif referencia Contribuinte (nif)

RI: dt\_emissao tem de ser maior que a dt\_criacao.

**Item\_Hist** (num\_tem, codigo\_fat,sku,desc\_item,desconto,num\_uni)

FK: (codigo\_fat) referencia Fatura (codigo\_fat)

RI: O atributo quantidade só pode tomar valores entre 0 e 1, inclusive.

# Avaliação Experimental

A normalização foi feita até à 3ª forma normal, de forma a evitar problemas

na gestão da base de dados.

De forma a testar as funcionalidades de alguns exercícios, corremos o *script* de testes (Exercício M) por inteiro, que chama a funcionalidade em si e depois faz *rollback* para manter a BD original, isto é, não haver alterações no *populate* das tabelas de novo.

## Exercício A

Para criar o modelo físico apenas criámos a base de dados e as várias tabelas, tendo por base o modelo EA e o relacional.

## Exercício B

Para apagar o modelo físico, apenas fazemos *drop* das várias tabelas.

## Exercício C

Neste exercício criámos um *populate tables* com alguns dados inseridos na base de dados para depois servirem como alvo de teste de alguns dos exercícios.

## Exercício I

## Exercício K

Neste exercício foi-nos pedido para atualizar o estado de uma fatura, para tal tivemos de nos certificar que só poderia ser feita alguma alteração a uma fatura caso o estado atual da mesma estivesse “Em atualização” ou “Proforma”. Caso o estado atual seja “Proforma” garantimos que só poderá fazer uma atualização para os estados “Anulada” ou “Emitida”. Após passar nestas verificações, fazemos os *updates* à tabela Fatura e para garantir que estas mudanças da fatura são guardadas como histórico, fazemos *inserts* na tabela Fatura\_Hist e temos em atenção que estas atualizações feitas no momento necessitam de ficar com a data de atualização e emissão, chamando assim a função *GETDATE()* e guardando nesses atributos. Utilizámos o nível de isolamento *Read Committed* para evitar a leitura errada do número da fatura que será modificada.

## Exercício L

Neste exercício tivemos de fazer uma view, e colocar a informação da tabela fatura e da tabela contribuinte.

De seguida criámos um *Trigger* na *View* de forma a que só fossem permitidos realizar *Updates* e no atributo “estado” como era pedido no enunciado. Utilizámos um *Trigger* para que caso haja uma alternativa de fazer *update* noutro atributo, ou fazer um *insert* ou *delete*, este é acionado, garantido assim a sua consistência.

# Conclusão

Dado o problema exposto, foi-nos dado o conhecimento de como usar a

gestão transacional e da sua utilização. Recorremos a escalonamentos

teóricos de forma a conseguirmos prever que anomalias teríamos de evitar

em cada caso. Para evitar a existência de anomalias, foi aprendida a utilidade de recorrer a níveis de isolamento, e os seus impactos positivos e negativos a nível computacional, na gestão de uma base de dados.

Concluímos assim que o trabalho foi realizado com sucesso, podendo ainda serem feitas algumas melhorias para as quais poderão ser feitas para a parte 2 do mesmo.

# WebGrafia

* Relatório do trabalho
* Slides da disciplina do moodle
* [Transact-SQL Reference (Database Engine) - SQL Server | Microsoft Docs](https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver15)
* [Microsoft PowerPoint - bd2\_processamento-de-transacoes.ppt [Modo de Compatibilidade] (ufop.br)](http://www.decom.ufop.br/guilherme/BCC441/geral/bd2_processamento-de-transacoes.pdf)
* [bd\_transactions (up.pt)](https://www.dcc.fc.up.pt/~edrdo/aulas/bd/teoricas/bd_transactions.pdf)