DataWarehouse

Filipe Fidalgo
ffidalgo@ipcb.pt

Revisões - Descrição do Modelo

Consideremos uma organização que guarda registo das receitas culinárias que produz.

É importante saber quais os ingredientes que estão na composição de cada receita, bem como a respetiva quantidade usada. Cada uma das receitas tem sempre um ingrediente principal que a carateriza (ex: Bacalhau à Brás – ingrediente principal Bacalhau).

Cada receita tem também uma ou várias categorias associadas, por exemplo, uma determinada receita pode para algumas pessoas ser uma entrada, mas para outras ser um prato principal.

As receitas são ainda associadas a uma determinada região do país. Sendo que cada região tem sempre e apenas uma receita que é o seu ex-libris.

Finalmente de algumas receitas é conhecido o chefe que a criou.

Revisões - Descrição do Modelo

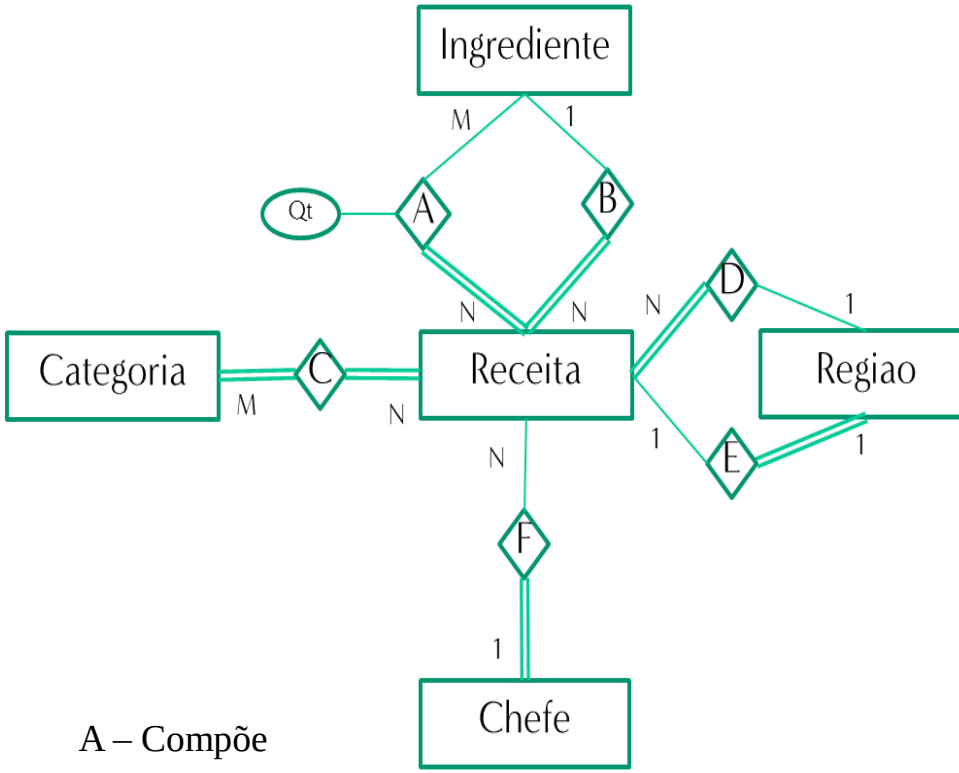
Consideremos uma organização que guarda registo das receitas culinárias que produz.

É importante saber quais os ingredientes que estão na composição de cada receita, bem como a respetiva quantidade usada. Cada uma das receitas tem sempre um ingrediente principal que a carateriza (ex: Bacalhau à Brás – ingrediente principal Bacalhau).

Cada receita tem também uma ou várias categorias associadas, por exemplo, uma determinada receita pode para algumas pessoas ser uma entrada, mas para outras ser um prato principal.

As receitas são ainda associadas a uma determinada região do país. Sendo que cada região tem sempre e apenas uma receita que é o seu ex-libris.

Finalmente de algumas receitas é conhecido o chefe que a criou.



A – Compõe

B – IngredientePrincipal

C – Corresponde

D – Associada

E - Representa

F – Criada

Descrição do Modelo

Quantos aos atributos:

De cada ingrediente para além de um identificador e respetivo nome, conhece-se ainda o respetivo valor calórico por unidade e o sotck.

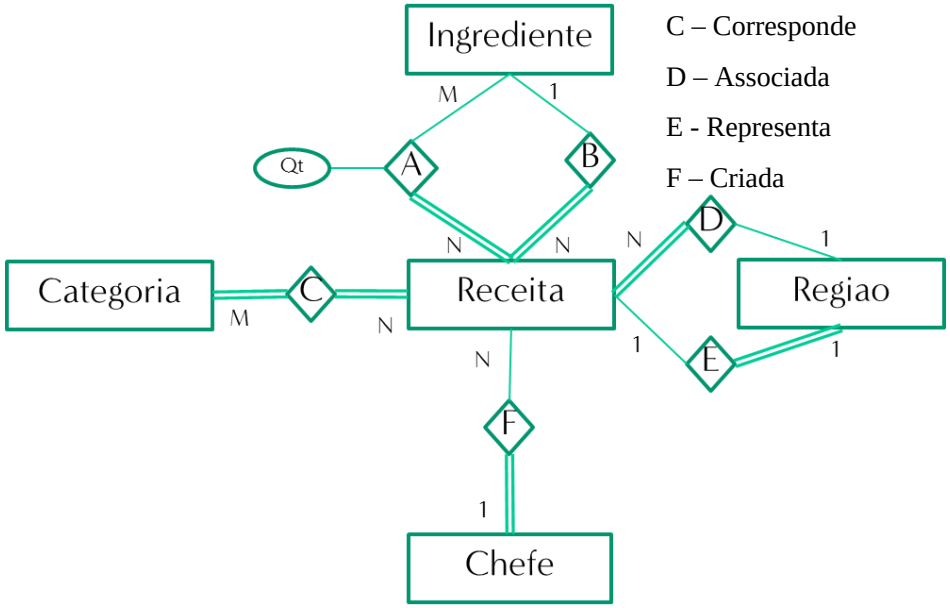
Sobre a categoria guardamos registo do identificador e do nome.

A receita tem também um identificador, um nome, uma descrição e a data em que foi criada. Dispõe ainda de um texto que descreve a respetiva preparação, e o total de calorias (obtido pelo multiplicação do valor calórico por unidade de cada ingrediente e a respetiva quantidade usada).

Quanto à região guardamos o identificador e o seu nome.

Do chefe regista-se o identificador, o nome, a morada e a sua data de nascimento.

A – Compõe
B – IngredientePrincipal
C – Corresponde
D – Associada
E - Representa
F – Criada



Ingrediente(<u>id_ingrediente</u>, nome_ingrediente, KalUnidade, stock)

Categoria (<u>id_categoria</u>, nome_categoria)

Receita (<u>id_receita</u>, nome_receita, descrição_receita, preparação, data_criacao, KalTotal)

Regiao (<u>id_regiao</u>, nome_regiao)

Chefe (<u>id_chefe</u>, nome_chefe, morada, data_nasc)

Descrição do Modelo

Quantos aos atributos:

De cada ingrediente para além de um identificador e respetivo nome, conhece-se ainda o respetivo valor calórico por unidade e o stock.

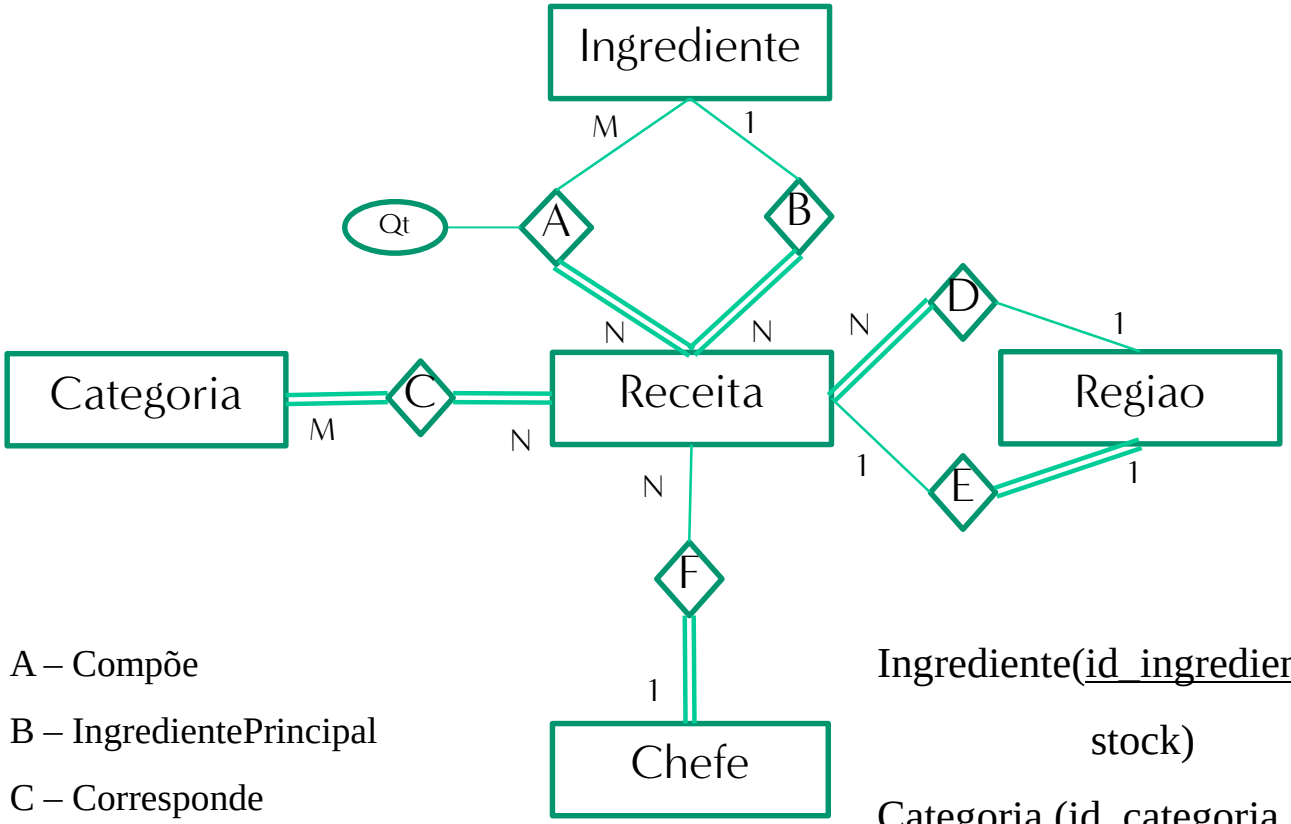
Sobre a categoria guardamos registo do identificador e do nome.

A receita tem também um identificador, um nome, uma descrição e a data em que foi criada. Dispõe ainda de um texto que descreve a respetiva preparação, e o total de calorias (obtido pelo multiplicação do valor calórico por unidade de cada ingrediente e a respetiva quantidade usada).

Quanto à região guardamos o identificador e o seu nome.

Do chefe regista-se o identificador, o nome, a morada e a sua data de nascimento.

Modelo ER



A – Compõe

B – IngredientePrincipal

C – Corresponde

D – Associada

E - Representa

F – Criada

Ingrediente(id_ingrediente, nome_ingrediente, KalUnidade, stock)

Categoria (id_categoria, nome_categoria)

Receita (id_receita, nome_receita, descrição_receita, preparação, data_criacao, KalTotal)

Regiao (id_regiao, nome_regiao)

Chefe (id_chefe, nome_chefe, morada, data_nasc)

Modelo Relacional

Ingrediente(id_ingrediente, nome_ingrediente, KalUnidade, stock)

Categoria (id_categoria, nome_categoria)

Regiao (id_regiao, designação, ReceitaID)

Chefe (id_chefe, nome_chefe, morada, data_nasc)

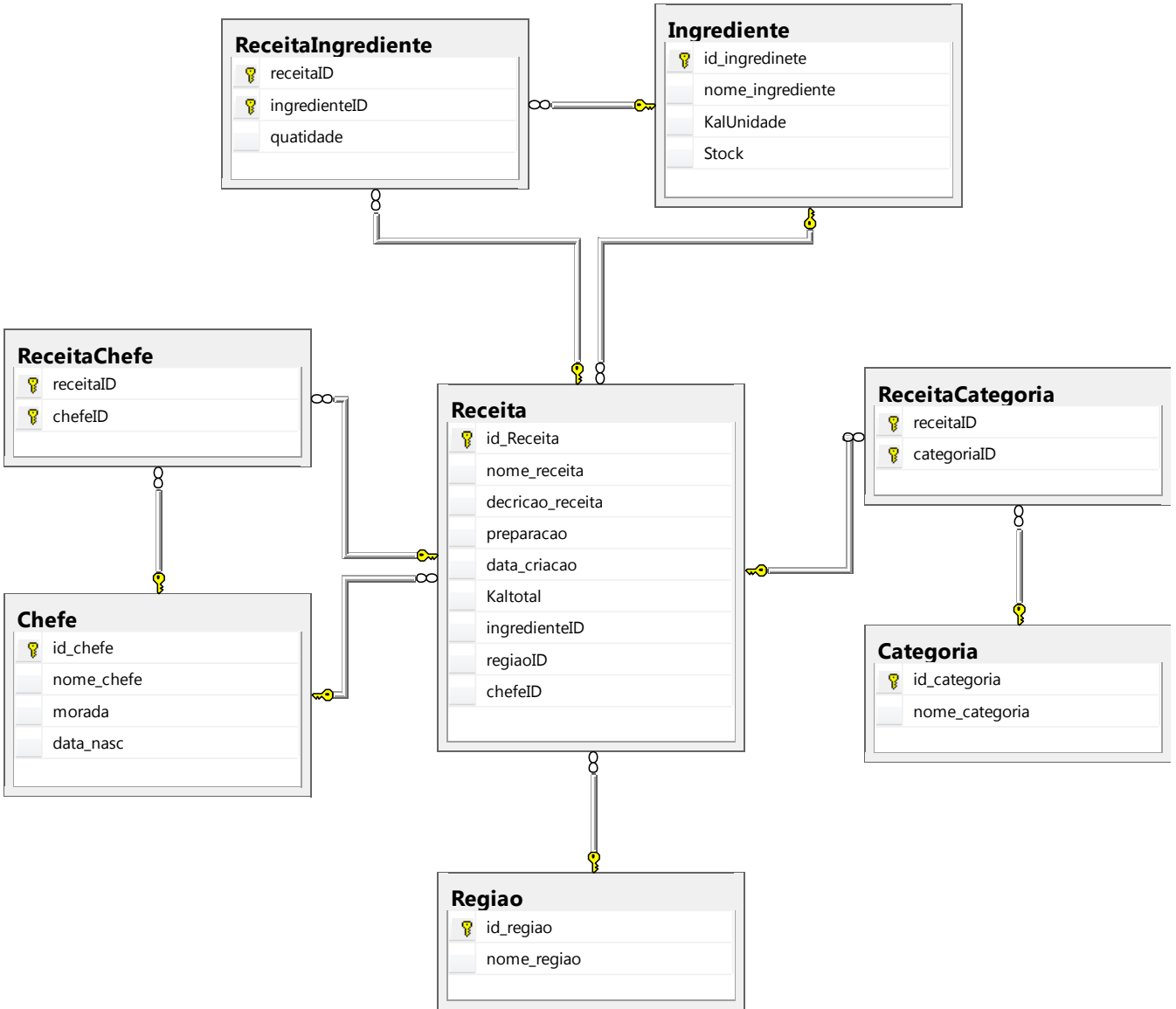
Receita (id_receita, nome_receita, descrição_receita,
preparação, data_criação, KalTotal, ingredienteID, regiaoID)

ReceitaIngrediente (receitaID, ingredienteID, quantidade)

ReceitaCategoria (receitaID, categoriaID)

ReceitaChefe (receitaID, chefeID)

Modelo Relacional



SQL

...

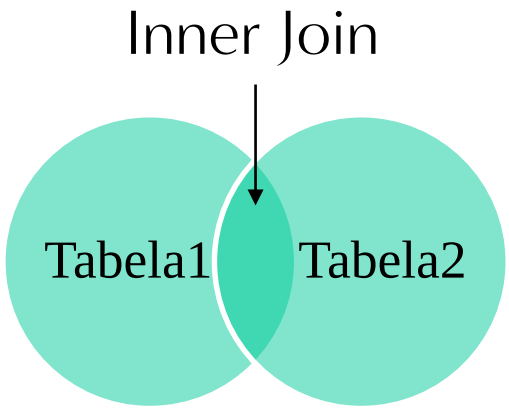
Sumário

- SQL – Junções
  - Inner Join;
  - Outer Join (Left, Rigth, Full);
  - Cross Join;
  - Self Join;
  - Union (All);
  - Intersect.

SQL Join

- SQL - Inner Join;

A operação de junção interna Inner Join, é realizada normalmente entre duas tabelas com o objetivo de obter os elementos que satisfazem a condição de junção, isto é, a condição de comparação entre as colunas de cada tabela.
Normalmente é usado o operador de igualdade, e o relacionamento faz-se entre a chave primaria de uma tabela e a chave forasteira da outra.




```
SELECT lista_colunas
FROM tabela1
    [INNER] JOIN tabela2
        ON condição_de_junção1
    [[INNER] JOIN tabela3
        ON condição_de_junção2]...
```

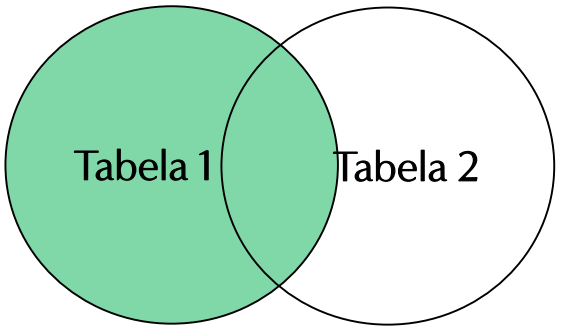
SQL Join

- SQL - Outer Join;

A operação de junção externa
Outer Join, é realizada entre duas tabelas com o objetivo de obter os elementos que satisfazem a condição de junção, mais os registos que não verificam as condições de junção de uma (LEFT OUTER JOIN, RIGHT OUTER JOIN) ou de ambas (FULL OUTER JOIN) as tabelas.

Quando se utiliza uma operação de junção externa à esquerda (LEFT OUTER JOIN), o resultado inclui todos os registos da primeira (esquerda) tabela.

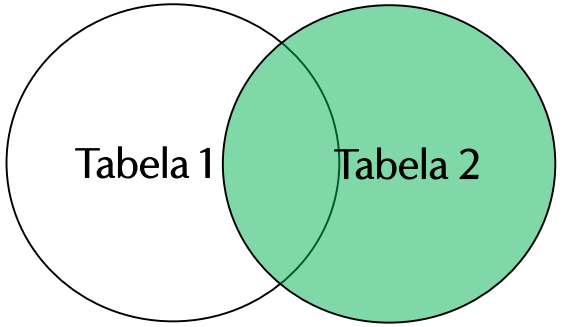
Left Outer Join



SQL Outer Join e Self Join

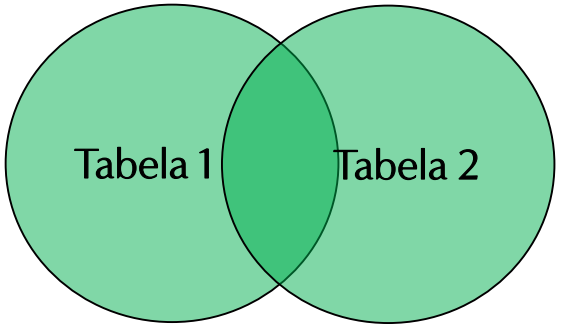
Da mesma forma, quando se usa uma operação de junção externa à direita (RIGHT OUTER JOIN), o resultado inclui todos os registos da segunda (direita) tabela.

Rigth Outer Join



No caso de se utilizar uma operação de junção externa completa (FULL OUTER JOIN), o resultado inclui todos os registos de ambas as tabelas.

Full Outer Join



É ainda possível realizar um operação de junção de uma tabela com ela mesma, designando-se nesta caso a junção como uma auto-junção (Self Join).

SQL Join

- SQL - Outer Join;

Quando não existe correspondência na condição de junção, as colunas poderão apresentar no seu resultado valores nulos!

Também aqui normalmente se usa como operador de junção a igualdade, relacionando a chave primária de uma tabela e uma chave forasteira da outra tabela.

```
SELECT lista_colunas
FROM tabela1
    {LEFT|RIGHT|FULL} [OUTER] JOIN tabela2
     ON condição_de_junção1
    [{LEFT|RIGHT|FULL }[OUTER] JOIN tabela3
     ON condição_de_junção2]...
[ORDER BY lista_de_ordenação]
```

SQL - Union

- SQL - Union;

Esta operação ao invés de combinar resultados com base nas colunas, combina resultados resultantes de dois (ou mais) instruções de SELECT.

Os resultados de cada instrução de SELECT devem ser *UNION COMPATIBLE*, isto é, mesmo número de campos, e tipos de dados compatíveis.

Os dados duplicados são eliminados pode definição, mas poderão ser incluídos, sendo nesse caso necessário usar a palavra "ALL" depois do operador UNION.

Instrução_**SELECT**_1

UNION [ALL]

Instrução_**SELECT**_2

[**UNION** [ALL]

Instrução_**SELECT**_3] ...

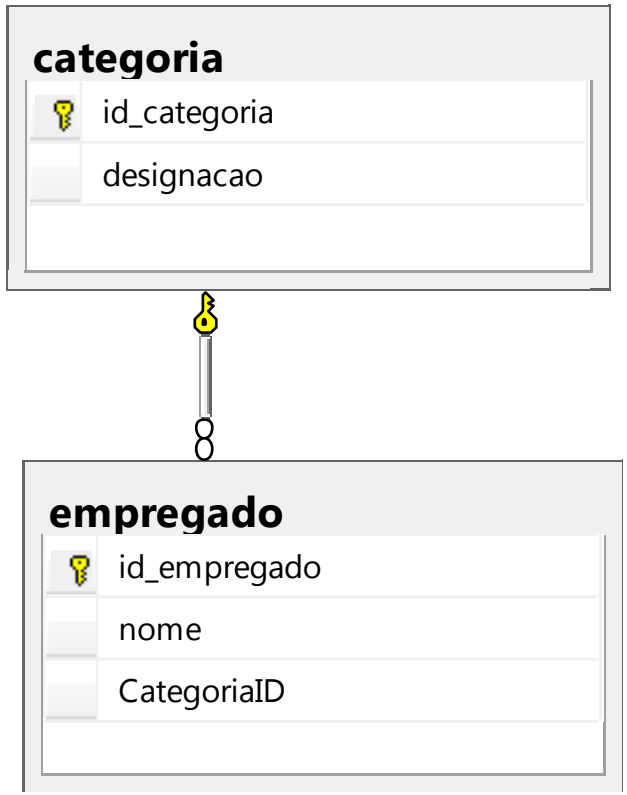
[**ORDER BY** lista_colunas_

Instrução_SELECT_1]

SQL - Exemplo

Consideremos o esquema seguinte, em que os empregados têm categorias.

Vamos assumir que neste caso (não dando relevância a questões de normalização...) que podemos ter empregados que não têm categoria, e categorias para as quais não existem também empregados.



Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Criar a estrutura

```
create table categoria(
id_categoria int,
designacao varchar(50),
constraint TB_Categoria_PK primary key(id_categoria)
);

create table empregado(
id_empregado int,
nome varchar(50),
CategoriaID int,
constraint TB_Empregado_PK primary key(id_empregado)
);

alter table empregado
add constraint TB_empregado_categoria_FK
foreign key (CategoriaID) references Categoria(id_categoria);


insert into categoria values (1, 'Programador');
insert into categoria values (2, 'Analista');
insert into categoria values (3, 'Gestor de Projeto');


insert into empregado values (1, 'Filipe', 1);
insert into empregado values (2, 'Fidalgo', 2);
insert into empregado values (3, 'Rui', 1);
insert into empregado values (4, 'Rita', NULL);
```

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

Observando as relações, podemos concluir que:

- Existe uma categoria (3 – Gestor de Projeto), que não tem qualquer empregado;
- Existe um empregado (4- Rita) , que não tem categoria atribuída.

- Vejamos estão algumas questões que podemos colocar ao modelo:

Listar para os empregados toda a sua informação, e a designação da sua categoria?

O resultado seria:

	id_empregado	nome	CategoriaID	designacao
1	1	Filipe	1	Programador
2	2	Fidalgo	2	Analista
3	3	Rui	1	Programador

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

Listar para os empregados toda a sua informação, e a designação da sua categoria?

O resultado seria:

	id_empregado	nome	CategoriaID	designacao
1	1	Filipe	1	Programador
2	2	Fidalgo	2	Analista
3	3	Rui	1	Programador

Deixando de fora os tuplos que não satisfazem a condição, isto é, não temos o empregado "Rita", já que este não tem categoria associada!

E qual o SQL que usaríamos para obter a resposta:

```
Select e.*, designacao
from Empregado as E INNER JOIN Categoria as C
ON E.CategoriaID = C.id_categoria
```

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

No caso de desejarmos obter uma listagem (neste caso apenas teórico!) de todos os empregados associados a todas as categorias?

O resultado seria:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Programador
3	3	Rui	Programador
4	4	Rita	Programador
5	1	Filipe	Analista
6	2	Fidalgo	Analista
7	3	Rui	Analista
8	4	Rita	Analista
9	1	Filipe	Gestor de Projeto
10	2	Fidalgo	Gestor de Projeto
11	3	Rui	Gestor de Projeto
12	4	Rita	Gestor de Projeto

De fato neste caso o resultado é estranho e nada contriui em termos de informação, mas representa o que pode ocorrer num produto cruzado entre duas tabelas.

E qual o SQL que usaríamos para obter a resposta:

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

No caso de desejarmos obter uma listagem (neste caso apenas teórico!) de todos os empregados associados a todas as categorias?

O resultado seria:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Programador
3	3	Rui	Programador
4	4	Rita	Programador
5	1	Filipe	Analista
6	2	Fidalgo	Analista
7	3	Rui	Analista
8	4	Rita	Analista
9	1	Filipe	Gestor de Projeto
10	2	Fidalgo	Gestor de Projeto
11	3	Rui	Gestor de Projeto
12	4	Rita	Gestor de Projeto

De fato neste caso o resultado é estranho e nada contriui em termos de informação, mas representa o que pode ocorrer num produto cruzado entre duas tabelas.

E qual o SQL que usaríamos para obter a resposta:

```
Select e.id_empregado, e.nome, designacao
from Empregado as E CROSS JOIN Categoria as C
```

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

Já observamos que existem empregados que não têm categoria associada (4 - Rita).
Mas se desejarmos listar todos os empregados com as suas respetivas categorias, incluindo os empregados sem categoria associada?

O resultado seria:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador
4	4	Rita	NULL

Já sabemos que se usarmos o INNER JOIN, obtemos os empregados onde há correspondência:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador

```
Select e.id_empregado, e.nome, c.designacao
from Empregado as E INNER JOIN Categoria as C
ON E.CategoriaID = C.id_categoria
```

Mas como adicionar os outros onde não há correspondência (neste caso a "Rita")?

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

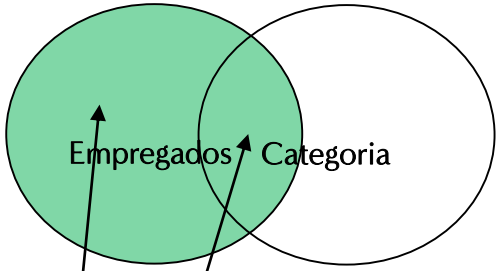
Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

É neste contexto que que queremos:

"todos os empregados + "comuns entre empregados categorias"



Temos então de adicionar ao:
- INNER JOIN (junção interna)

o
- OUTER JOIN (junção externa- neste caso à esquerda)

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

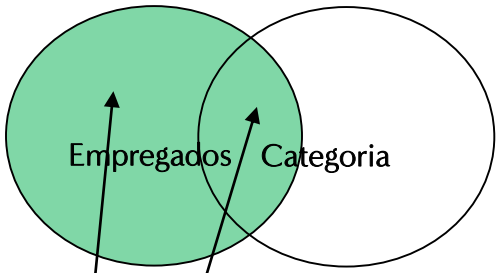
Qual o SQL:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador
4	4	Rita	NULL

Exemplos

É neste contexto que que queremos:

"todos os empregados + "comuns entre empregados categorias"



Temos então de adicionar ao:
- INNER JOIN (junção interna)

o
- OUTER JOIN (junção externa)

Qual o SQL:

```
Select e.id_empregado, e.nome, c.designacao
from Empregado as E LEFT OUTER JOIN Categoria as C
ON E.CategoriaID = C.id_categoria
```

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador
4	4	Rita	NULL

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

De forma análoga podemos pensar que existe uma categoria que não tem qualquer empregado associado (3 – Gestor de Projeto).
Mas se desejarmos listar todas as categoria com os seus respetivo empregados, incluindo as categoria sem empregados associados?

O resultado seria:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	3	Rui	Programador
3	2	Fidalgo	Analista
4	NULL	NULL	Gestor de Projeto

Já sabemos que se usarmos o INNER JOIN, obtemos as categorias onde há correspondência:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador

```
Select e.id_empregado, e.nome, c.designacao
from Empregado as E INNER JOIN Categoria as C
ON E.CategoriaID = C.id_categoria
```

Mas como adicionar as outras onde não há correspondência (neste caso a "Gestor de Projeto")?

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

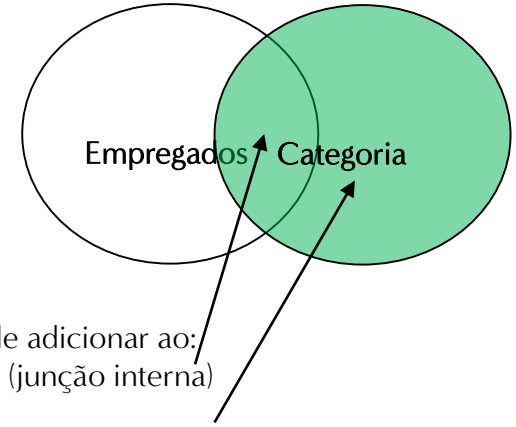
Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

É neste contexto que que queremos:

"todas as categorias + "comuns entre empregados categorias"



Temos então de adicionar ao:
- INNER JOIN (junção interna)

o
- OUTER JOIN (junção externa – neste caso à direita)

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

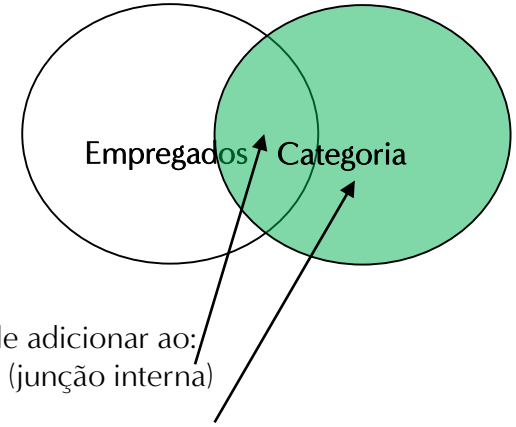
Qual o SQL:

	id_categoria	designacao	nome
1	1	Programador	Filipe
2	1	Programador	Rui
3	2	Analista	Fidalgo
4	3	Gestor de Projeto	NULL

Exemplos

É neste contexto que que queremos:

"todas as categorias + "comuns entre empregados categorias"



Temos então de adicionar ao:
- INNER JOIN (junção interna)

o
- OUTER JOIN (junção externa – neste caso à direita)

Qual o SQL:

```
Select c.id_categoria, c.designacao, e.nome
from Empregado as E RIGHT OUTER JOIN Categoria as C
ON E.CategoriaID = C.id_categoria
```

	id_categoria	designacao	nome
1	1	Programador	Filipe
2	1	Programador	Rui
3	2	Analista	Fidalgo
4	3	Gestor de Projeto	NULL

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

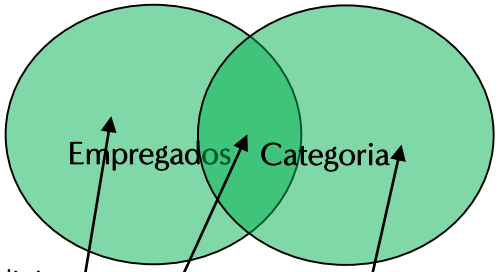
Finalmente podemos ainda pretender ter no resultado os empregados sem correspondências (isto é, empregados sem categoria), e as categorias sem correspondências (isto é, categorias sem empregados)

O resultado seria:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador
4	4	Rita	NULL
5	NULL	NULL	Gestor de Projeto

É neste contexto que que queremos:

"todas os empregados" + "comuns entre empregados categorias" + "todas as categorias"



Temos então de adicionar ao:
- INNER JOIN (junção interna)

o
- OUTER JOIN (junção externa – neste caso à esquerda)

o
- OUTER JOIN (junção externa – neste caso à direita)

Categoria

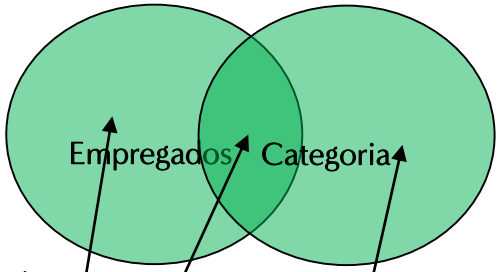
	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

"todas os empregados" + "comuns entre empregados categorias" + "todas as categorias"



Temos então de adicionar ao:
- INNER JOIN (junção interna)

o
- OUTER JOIN (junção externa – neste caso à esquerda)

o
- OUTER JOIN (junção externa – neste caso à direita)

Qual o SQL:

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador
4	4	Rita	NULL
5	NULL	NULL	Gestor de Projeto

Categoria

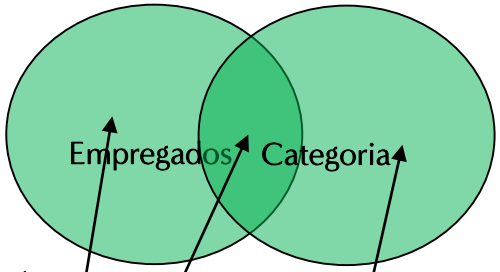
	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

"todas os empregados" + "comuns entre empregados categorias" + "todas as categorias"



Temos então de adicionar ao:
- INNER JOIN (junção interna)

o
- OUTER JOIN (junção externa – neste caso à esquerda)

o
- OUTER JOIN (junção externa – neste caso à direita)

Qual o SQL:

```
Select e.id_empregado, e.nome, c.designacao
from Empregado as E FULL OUTER JOIN Categoria as C
ON E.CategoriaID = C.id_categoria
```

	id_empregado	nome	designacao
1	1	Filipe	Programador
2	2	Fidalgo	Analista
3	3	Rui	Programador
4	4	Rita	NULL
5	NULL	NULL	Gestor de Projeto

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Exemplos

Consideremos agora tabela adicional Empregado1:

```
create table empregado1(
id_empregado int,
nome varchar(50),
CategoriaID int,
constraint TB_Empregado1_PK primary key(id_empregado)
);

insert into empregado1 values (1, 'Filipe', 1);
insert into empregado1 values (2, 'Ana', 2);
insert into empregado1 values (3, 'Miguel', 2);
```

No caso de pretendermos juntar o conteúdo da tabela "Empregado" e "Empregado1", iriamos obter:

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Ana	2
3	2	Fidalgo	2
4	3	Miguel	2
5	3	Rui	1
6	4	Rita	NULL

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Empregado1

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Ana	2
3	3	Miguel	2

Exemplos

Observemos que no resultados temos 6 linhas, mas a tabela Empregado tem 4 linhas e a tabela Empregado1 tem 3 linhas....
A junção das duas tabelas não deveria ter dado 7 linhas?

Se executarmos o sql:

```
select * from empregado
UNION
select * from empregado1
```

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Ana	2
3	2	Fidalgo	2
4	3	Miguel	2
5	3	Rui	1
6	4	Rita	NULL

A resposta é que não, porque a linha 1 de ambas as tabelas é igual!
Sempre que usarmos o comando UNION, ele encarrega-se de eliminar os duplicados.

Mas e se quisermos incluir as linhas repetidas?

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL
5	1	Filipe	1
6	2	Ana	2
7	3	Miguel	2

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Empregado1

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Ana	2
3	3	Miguel	2

Exemplos

Bom neste caso o comando UNION tem de ser complementado com o palavra reservada ALL:

Assim, se executarmos o sql:

```
select * from empregado
UNION ALL
select * from empregado1
```

Obtendo desta forma a união das duas tabelas incluindo as linhas repetidas!

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL
5	1	Filipe	1
6	2	Ana	2
7	3	Miguel	2

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Empregado1

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Ana	2
3	3	Miguel	2

Exemplos

Adicionalmente poderíamos ainda procurar os elementos que são comuns às duas tabelas Empregado e Empregado1 (elementos que estão ao mesmo tempo na tabela Empregado e na tabela Empregado1):

	id_empregado	nome	CategoriaID
1	1	Filipe	1

Neste caso temos apenas uma linha comum às duas tabelas!

Assim, qual o sql necessário?

```
select * from empregado
UNION ALL
select * from empregado1
```

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Empregado1

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Ana	2
3	3	Miguel	2

Exemplos

Adicionalmente poderíamos ainda procurar os elementos que são comuns às duas tabelas Empregado e Empregado1 (elementos que estão ao mesmo tempo na tabela Empregado e na tabela Empregado1):

	id_empregado	nome	CategoriaID
1	1	Filipe	1

Neste caso temos apenas uma linha comum às duas tabelas!

Assim, qual o sql necessário?

```
select * from empregado
INTERSECT
select * from empregado1
```

	id_empregado	nome	CategoriaID
1	1	Filipe	1

Categoria

	id_categoria	designacao
1	1	Programador
2	2	Analista
3	3	Gestor de Projeto

Empregado

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Fidalgo	2
3	3	Rui	1
4	4	Rita	NULL

Empregado1

	id_empregado	nome	CategoriaID
1	1	Filipe	1
2	2	Ana	2
3	3	Miguel	2

Resumo

- Existem vários tipos de junções, em sql:
  - **Inner Join** (junta linhas que satisfaçam as condições de junção);
  - **Outer Join** (Left, Rigth, Full) (junta linhas que satisfaçam as condições de junção e as que não as verificam, de uma ou de ambas);
  - **Cross Join** (juntar duas tabelas sem condição de junção... Produto cruzado);
  - **Self Join** (juntar duas vezes a mesma tabela);
  - **Union e Union All** (unir duas tabelas, desde que sejam compatíveis. Para incluir os duplicados temos de juntar a clausula ALL);
  - **Intersect** (procurar elementos que estejam ao mesmo tempo em duas tabelas).

TRIGGERS

Conjunto de instruções que podem ser associadas a uma tabela ou a uma vista, para um determinado evento.

Um trigger é disparado quando tabela é alvo de uma determinada operação.

Exercício

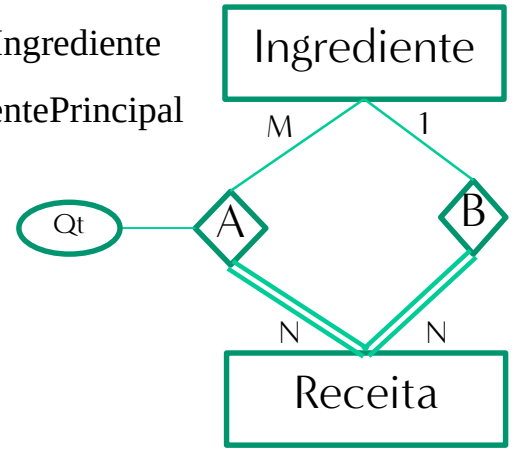
Considere o modelo de entidade e relacionamentos representado na figura ao lado.

Atendendo a que o total de calorias (KalTotal) de uma dada receita é obtido através do produto da quantidade (qt) pelo valor unitário das calorias (KalUnidade) do respetivo Ingrediente, crie um objeto que lhe permita atualiza o valor do total de calorias, de cada vez que:

a) É inserido um novo ingrediente na composição da respetiva receita.

A – ReceitaIngrediente

B – IngredientePrincipal



Ingrediente(id_ingrediente, nome_ingrediente, KalUnidade, stock)

Receita (id_receita, nome_receita, descrição_receita, preparação, data_criacao, KalTotal)

Exercício

Suponhamos que as tabelas têm os seguintes registos:

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

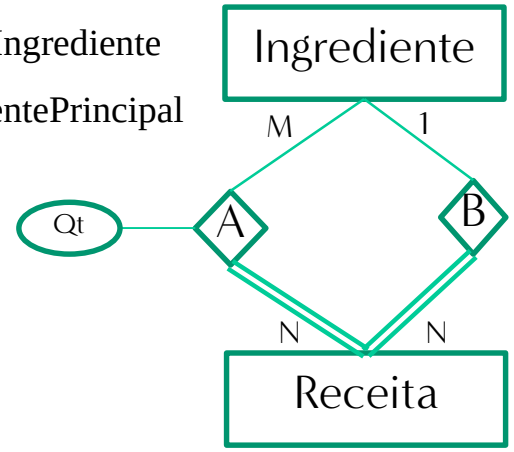
Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		? (Null)	2
,,,				

ReceitaIngrediente

ReceitaID	IngredienteID	Qt
1	1	2
1	2	3
1	3	4
…		

A – ReceitaIngrediente

B – IngredientePrincipal



Ingrediente(id_ingrediente, nome_ingrediente, KalUnidade, stock)

Receita (id_receita, nome_receita, descrição_receita, preparação, data_criacao, KalTotal)

Exercício

Suponhamos que as tabelas têm os seguintes registos:

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredienteID	Qt
1	1	2
1	2	3
1	3	4
…		

Assim, qual seria o valor do "KalTotal" para a receita 1?

Teríamos de calcular, para cada linha da composição da receita, o produto da quantidade (Qt) pelo valor unitário das calorias (KulUnidade) do respetivo Ingrediente:

Como a receita tem na sua composição os Ingredientes 1, 2 e 3:

Qt * KalUnidade = 2 * 2 = 4 (para o Ingrediente 1)

Qt * KalUnidade = 3 * 4 = 12 (para o Ingrediente 2)

Qt * KalUnidade = 4 * 6 = 24 (para o Ingrediente 3)

Depois somá-los:

4 + 12 +24 = 40

No final atualizar o valor total de calorias (KalTotal) na respetiva receita:

KalTotal = 40

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredienteID	Qt
1	1	2
1	2	3
1	3	4
…		

Comecemos:

Quando criamos um trigger temos de decidir (para além do nome) à partida "três" coisas:

- Onde?
  - Qual a tabela onde vai ocorrer a ação, e que vai ativar o trigger;
- Quando?
  - Em que momento o trigger vai atuar (pode ser "AFTER" e "INSTED OF";
- Qual a ação?
  - Se foi porque foi feito um {INSERT [,] UPDATE [,] DELETE}.

Assim:

```
CREATE TRIGGER ACT_KalTotal
ON ReceitaIngrediente
AFTER INSERT
....
```

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredineteID	Qt
1	1	2
1	2	3
1	3	4
…		

Cont:

Cada vez que se usa um Trigger do tipo INSERT, ficamos com acesso aos "novos" (através da tabela INSERTED) valores inseridos na zona da execução do Trigger (ie, entre o BEGIN e o END). Neste caso como a alteração foi feita na tabela "ReceitaIngrediente", ficamos com acesso a todos os seus atributos: ReceitaID, IngredienteID, QT

Vejamos por exemplo a inserção da 1ª linha da tabela ReceitaIngrediente, ficamos no Trigger com acesso aos valores:

RceitaID = 1;

IngredienteID = 1;

Qt = 2

No caso da operação que ativa o Trigger ter sido um UPDATE ou um DELETE, ficávamos com acesso aos valores alterados ou apagados (através da tabela (DELETED)

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredineteID	Qt
1	1	2
1	2	3
1	3	4
…		

Cont:

Mas para que esse acesso possa acontecer, temos de usar clausula

CREATE TRIGGER ACT_KalTotal

ON ReceitaIngrediente

AFTER INSERT

....

Vejamos agora o que fazer na execução do Trigger:

1) Temos de ir selecionar o valor "KalUnidade", para a respetiva linha inserida (neste caso seria o KalUnidade=2, porque a linha inserida continha o ~~Ingrediente~~ IngredienteID=1);

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredineteID	Qt
1	1	2
1	2	3
1	3	4
…		

Vamos usar uma variável para armazenar o valor da "KalUnidade", para a respetiva linha inserida (neste caso seria o KalUnidade=2, porque a linha inserida continha o Ingrediente IngredienteID=1);

```
CREATE TRIGGER ACT_KalTotal
ON ReceitaIngrediente
AFTER INSERT

AS
BEGIN

        update receita
        set Kaltotal = Kaltotal + (select quantidade * (select
KalUnidade from ingrediente where id_ingrediente = (select
ingredienteID from inserted)) from inserted)
        where id_receita = (select receitaID from inserted)

END
```

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredineteID	Qt
1	1	2
1	2	3
1	3	4
…		

```
CREATE TRIGGER ACT_KalTotal
ON ReceitaIngrediente
AFTER INSERT

AS
BEGIN

        update receita
        set Kaltotal = Kaltotal + (select quantidade * (select
KalUnidade from ingrediente where id_ingrediente = (select
ingredienteID from inserted)) from inserted)
        where id_receita = (select receitaID from inserted)

END
```

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredineteID	Qt
1	1	2
1	2	3
1	3	4
…		

Depois de Inserirmos a 1ªlinha da tabela ReceitaIngrediente, o valor do atributo KalTotal da Receita 1 seria: "4"

Depois de Inserirmos a 2ªlinha da tabela ReceitaIngrediente, o valor do atributo KalTotal da Receita 1 seria: "16" (4 + 12)

Depois de Inserirmos a 2ªlinha da tabela ReceitaIngrediente, o valor do atributo KalTotal da Receita 1 seria: "40" (16 + 24)

Assim, por cada linha inserida na tabela ReceitaIngrediente, o valor do atributo KalTotal da Receita correspondente, é automaticamente aualizado!

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredineteID	Qt
1	1	2
1	2	3
1	3	4
…		

b) Construa um trigger para atualizar o valor de KalTotal, no caso de uma dada linha da tabela ReceitaIngrediente ser alterada (por exemplo, imagine que na 3ªlinha em vez de usar a quantidade 4, passou a usar a quantidade 3. Assim em vez do valor KalTotal ser 40, passaria a ser: 34!)

Exercício

Ingrediente

Id_ingredinete	Nome_Ing	KalUnidade	Stock
1	Batata	2	100
2	Polvo	4	100
3	Azeite	6	100
…			

Receita

Id_Receita	Nome_Ing	…	KalTotol	IngPrin
1	Polvo à Lagareiro		?	2
,,,				

ReceitaIngrediente

ReceitaID	IngredineteID	Qt
1	1	2
1	2	3
1	3	4
…		

c) Suponha agora que eliminou uma linha da tabela ReceitaIngrediente. Crie um objeto que faça refletir essa situação no valor KalTotal (se por exemplo a linha eliminada tivesse sido a 3ª, o valor do KalTotal passaria a ser: 16).

NOTA:
Seria ainda possível juntar todas as opções das alíneas anteriores num único Trigger (exercício!)

Procedimentos

Existem blocos modulares de código, précompilados e armazenados no servidor como parte de uma BD, conhecidos como stored procedures e triggers.
Cada um destes blocos pode conter instruções de controle de fluxo, comandos DML ou simples consultas. Algumas das vantagens da sua utilização:
Centralização: Qualquer mudança nas regras de negócio implicaria uma actualização em dezenas, às vezes centenas de binários espalhados pela empresa, pela cidade ou pelo mundo, desta forma basta fazer uma única actualização ao nível do servidor;
Segurança: A tarefa do DBA fica facilitada, ao invés de ter que controlar diferentes níveis de privilégios sobre os diferentes objectos, pode definir a segurança a partir de stored procedures;
Trafico de rede: Os stored procedure ou os trigger são armazenados ao nível do servidor, ao cliente basta executá-los, implícita ou explicitamente, com eventuais parâmetros, e receber de volta algum possível retorno.
Clientes magros (Thin Client): Como as regras de negócio estão no servidor, as aplicações podem ser menores, exigindo menos recursos do sistema (disco, memória e CPU) e consequentemente hardware de menor custo;
As consultas embutidas dentro de stored procedures podem melhorar o desempenho das consultas repetitivas, já que o SGBD possui o seu plano se execução pré-compilado.

Procedimentos

Com recurso a T-Sql, crie um procedimento "ConsultaReceita", que receba como parâmetro de entrada o identificador de uma dada receita e que:

- Liste toda a informação sobre a receita (id_receita, nome, preparacao, total de calorias, tempo de preparacao), com uma label inicial para cada atributo, por exemplo:
  - Numero Receita: 1;
  - Nome Receita: "Jardineira"
  - ...
- Liste os ingredientes e as respetivas quantidades que a que compõem essa receita, antecedidas da palavra ingrediente com o respetivo número, por exemplo:
  - Ingrediente1: "Vitela" – Quantidade: "10"
  - Ingrediente2: "Cenoura" – Quantidade: "3"
  - ...
- Indique a data em que a consulta foi feita, por exemplo:
  - Consulta em: 17 de Novembro de 2018

```
create procedure ConsultaReceita @vp_id_receita int
as

Declare @v_nome varchar(30), @v_total_calorias varchar(30),
@v_preparacao varchar(30), @v_prepacacao varchar(30)
Declare @i int, @v_nome_ingrediente varchar(30)

DECLARE @lista_ingredientes CURSOR
SET @lista_ingredientes = CURSOR FOR
    SELECT ingredienteID, quantidade FROM ReceitaIngrediente
where receitaID = @vp_id_receita

Declare @v_id_ingrediente int, @v_quantidade int
```

```
Begin
    set @i=1

    select @v_nome = nome_receita, @v_total_calorias =
Kaltotal, @v_preparacao = preparacao, @v_prepacacao =
preparacao
    from receita where id_receita = @vp_id_receita

    print 'Num Receita: ' + cast(@vp_id_receita as varchar)
    print 'Receita: ' + @v_nome
    print 'Preparacao: ' + @v_preparacao
    print 'Total Calorias: ' + cast(@v_total_calorias as varchar)
    print 'Tempo de Preparação: ' + @v_preparacao
    print ''
```

```
OPEN @lista_ingredientes
   FETCH NEXT FROM @lista_ingredientes INTO @v_id_ingrediente, @v_quantidade

      While @i <= 10 AND @@FETCH_STATUS = 0

        begin

                select @v_nome_ingrediente = (select nome_ingrediente from ingrediente
where id_ingrediente = @v_id_ingrediente)
                print 'Ingredinete'+cast(@i as varchar)+': '+@v_nome_ingrediente+ ' -- ' +
'Quantidade: ' + cast(@v_quantidade as varchar)

                FETCH NEXT FROM @lista_ingredientes INTO @v_id_ingrediente,
@v_quantidade
                set @i = @i + 1
        end

      print ''
      print 'Data: ' + cast(getdate() as varchar)
END
```