Universidade de Coimbra - Faculdade de Ciências e Tecnologia
Departamento de Matemática

BASES DE DADOS - 2005/2006
4º ANO

PROJECTO – COZINHA

*** RELATÓRIO ***

Rita Margarida Ferreira Coimbra – tc01074@mat.uc.pt
Sara Joana Fino dos Santos Rodrigues de Carvalho – tc01037@mat.uc.pt
Sara Margarida Gaspar da Silva – tc01038@mat.uc.pt

Coimbra, 16 de Dezembro de 2005

Índice

Capítulo 1. Enunciado

Departamento de Matemática da Universidade de Coimbra	
Bases de Dados	
Projectos	2005/2006

Projectos, elaborados por grupos de dois ou três alunos, devem implementar uma base de dados de acordo com um dos itens abaixo.

Superliga contendo (pelo menos) os clubes, jogadores, jogos, . . .

Base de dados por livros contendo (pelo menos) livros, autores, assunto, língua…

Cadeiras contendo (pelo menos) cadeiras, alunos, exames…

Parlamento contendo (pelo menos) os partidos, resultados das eleições, deputados…

Uma base de dados à escolha, mas de complexidade semelhante às acima sugeridas

O projecto deve conter

- Um conceito de base de dados no modelo *Entity-Relationship* explicando entidades, atributos e associações.
- Uma lista das tabelas (simplificadas) correspondentes no modelo relacional.
- Uma lista de comandos em SQL que permita construir a base de dados.
- Uma implementação em MySQL contendo um exemplo concreto na base de dados escolhida (contendo um número razoável de entradas).
- Várias perguntas relevantes e de complexidade razoável, formuladas em

  - linguagem corrente

  - modelo relacional

  - MySQL

Nota. É preciso algum cuidado com a dimensão e dificuldade do trabalho. Por um lado, o trabalho não deve ser demasiado pequeno/simples (tipo gestão da lista de números de telefone dos meus amigos), sob pena de, mesmo que o trabalho fique muito bem feito, não seja suficiente para obter nota positiva no mesmo. Mas também não precisa ser muito grande/complexo (tipo gestão completa dos dados relativos a inscrições de alunos, lançamento de notas, turmas, horário, etc, duma faculdade).

Capítulo 2. Introdução

O objectivo deste trabalho é implementar uma base de dados de complexidade referida no enunciado do projecto. Deve constar ainda, um conceito de base de dados no modelo *Entity-Relationship* explicando entidades, atributos e associações; uma lista das tabelas (simplificadas) correspondentes no modelo relacional; uma lista de comandos em SQL que permita construir a base de dados; uma implementação em MySQL contendo um exemplo concreto na base de dados escolhida (contendo um número razoável de entradas; várias perguntas relevantes e de complexidade razoável, formuladas em linguagem corrente, álgebra relacional e em SQL.

Optámos pela construção de uma base de dados à qual iremos chamar COZINHA. Pretendemos assim implementar uma base de dados útil e eficiente que faça a gestão de uma cozinha de um restaurante, incluindo informações relevantes para a organização desta, tais como todas as informações respeitantes a receitas, produtos, stock, compras e fornecedores.

Capítulo 3. Bases de Dados

De uma forma simplista podemos dizer que uma base de dados consiste numa colecção de dados estruturados, organizados, inter-relacionados e armazenados de forma persistente.

Desde o início é necessário ter-se consciência que uma boa visão dos dados é fundamental para a realização de uma boa e útil base de dados.

Os programas de gestão de bases de dados devem permitir um fácil acesso e manipulação das bases de dados ao utilizador para que assim, haja um maior universo de aplicações das respectivas bases de dados.

O modelo de dados deve conter um conjunto de ferramentas para descrever os dados, as relações entre os dados, a semântica dos dados e as restrições sobre os dados. Para a descrição da nossa base de dados, vamos utilizar o modelo entidade-associação (ER- Entity Relationship). A modelação deste tipo é constituída por

- entidades (objectos)
- associações entre entidades

A linguagem de definição de dados (DDL) utilizada deve conter a especificação da notação para a definição do esquema da base de dados. O compilador da DDL gera um conjunto de tabelas armazenadas num dicionário de dados. As estruturas de armazenamento e os métodos de acesso utilizados pela base de dados são especificados por esta linguagem.

A linguagem de manipulação de dados (DML), também conhecida por linguagem de consulta, serve para aceder e manipular os dados organizados de acordo com o respectivo modelo de dados. Dentro desta linguagem temos duas classes de linguagens: procedimental (onde o utilizador especifica quais os dados que se pretendem assim como a maneira de os obter) e não-procedimental (onde o utilizador especifica quais os dados pretendidos mas não especifica a maneira de os obter).

O SQL (*Structured Query Language*) é a linguagem de consulta (não-procedimental) mais utilizada e é a ela que vamos recorrer na realização deste projecto. Vamos ainda utilizar, como sistema aplicação, o sistema MySQL para demonstrar as potencialidades da base de dados que vamos construir e da linguagem SQL, embora este programa contenha algumas variantes comparando com a versão original da linguagem SQL (é de ter em conta que existem outras aplicações muito utilizadas, tais como MS Access, Oracle entre outras). Duas razões da grande importância e utilização desta linguagem é que é possível utilizar o SQL embedido (aplicação que coloca comandos SQL dentro do código de outras linguagens de programação) e a existência de interfaces de aplicações (por exemplo ODBC/JDBC) permitindo o envio de consultas SQL para a base de dados.

Um modelo de base de dados é um modelo lógico de representação dos dados. Num modelo, não temos que nos preocupar com questões de implementação física, formato de dados, etc. Existem modelos específicos para a representação de dados ou da estrutura de dados numa base de dados. O mais famoso e utilizado é o modelo relacional. Num modelo relacional a informação é guardada em tabelas. Cada tabela é estruturada de forma a conter os dados referentes a entidades ou relacionamentos que, na situação prática, produzem a informação que a base de dados deve registar, actualizar e manter.

Qualquer base de dados tem duas componentes fundamentais: uma estrutura lógica e física que permite que a informação seja organizada; e um sistema de gestão da base de dados que assegura a gestão da informação. Uma base de dados tem o objectivo de registar, actualizar, manter e

disponibilizar informação. É também um conjunto de informações relacionadas com um determinado assunto ou finalidade.

Uma base de dados não tem, necessariamente, de estar informatizada (pode por exemplo consistir num conjunto de *post-it* colados num painel de parede com a informação necessária). É, no entanto, necessário que os dados tenham algum tipo de significado e organização. O Sistema Gestor de Bases de Dados (SGBD) é uma aplicação informática (falamos portanto em software) que fornece a *interface* entre os dados que são armazenados fisicamente na base de dados e o utilizador. Desta forma, o utilizador deixa de ter de se preocupar com a forma como os dados são armazenados, pesquisados ou ordenados, pois é o SGBD que tem a responsabilidade dessa tarefa. Quando falamos em utilizador estamo-nos a referir a alguém (pessoa) ou a uma aplicação informática.

Capítulo 4. Modelo ER

O primeiro passo na criação de uma base de dados consiste no seu desenho lógico utilizando ferramentas de análise.

4.1. Conjunto de Entidades e Atributos

Uma *base de dados* pode ser modelada como:

- uma colecção de entidades,
- relações (ou associações) entre entidades.

Uma *entidade* é um objecto existente e que se distingue de todos os outros objectos.
Cada entidade é representada por um conjunto de *atributos*, ou seja, propriedades descritivas possuídas por todos os membros de um conjunto de entidades.

Vamos construir uma base de dados que gere uma cozinha de um restaurante. Para que esta base de dados seja útil e eficaz fizemos um estudo prévio para escolher as entidades que vão ser utilizadas. Passamos então a enumerar as **entidades** da nossa base de dados:

- **cozinhado**
- **receitas**
- **produtos**
- **familia_p**
- **stock_peso**
- **stock_outro**
- **compras**
- **fornecedores**

Optámos por utilizar duas entidades respeitantes ao stock, stock_peso (onde utilizamos a medida kg ou litro) e stock_outro (onde utilizamos outras medidas como por exemplo lata, garrafa), pois há diversos produtos que são medidos em unidades de medida diferentes (kg, garrafas, molhos, latas etc), ou pode até ser que o mesmo produto seja medido em medidas diferentes (como é o caso por exemplo do azeite ao qual nos podemos referir em garrafas ou em litros). Para além disso, nas compras pode-nos ser mais útil referirmo-nos a certo produto numa determinada medida (por exemplo, no caso do leite condensado facilita referirmo-nos a latas em vez de nos referirmos a kg).

A entidade familia_p dá-nos a informação respeitante à família a que o produto pertence (vegetal, lacticínio, etc).

Todos os nomes dados às restantes entidades são suficientemente esclarecedores dispensando quaisquer descrições por agora.

Mas, as entidades não têm qualquer valor se não lhes for atribuído qualquer característica. É aqui que surge a importância dos atributos. Os atributos servem para descrever as entidades. Passamos a enumerar as **entidades** que fazem parte da nossa base de dados e os respectivos *atributos*:

→ **cozinhado**

- *id_coz :* é um número inteiro que identifica univocamente o cozinhado, funciona como o "número do bilhete de identidade" do cozinhado.
- *n_pessoas :* é o número de pessoas para que o cozinhado é feito.
- *duracao :* é o tempo que demora a preparar o cozinhado.
- *tipo :* é o tipo do cozinhado (pode ser um cozido, um grelhado, um frito, etc).

→ **receitas**

- *id_r :* é um número inteiro que identifica univocamente a receita, funciona como o "número do bilhete de identidade" da receita.
- *nome_r :* é o nome da receita.
- *familia_r :* é a família da receita, pode ser uma sobremesa, um prato de carne, peixe, uma entrada, uma bebida ou desconhecida.

→ **produtos**

- *id_p :* é um número inteiro que identifica univocamente o produto, funciona como o "número do bilhete de identidade" do produto.
- *nome_p :* é o nome do produto.
- *localizacao :* dá-nos o local onde esta armazenado o produto (numa das prateleiras, no congelador da carne, no congelador do peixe, na fruteira etc).

→ **familia_p**

- *id_familia :* é um número inteiro que identifica univocamente a família do produto, funciona como o "número do bilhete de identidade" da receita.
- *nome_familia :* dá-nos o nome da família a que pertence o produto (pode ser um vegetal, carne, peixe etc)

→ **stock_peso**

- *pmedida :* é a medida de peso usada para medir a quantidade de produto em stock (litro ou kg).
- *punidade :* é o número de medidas de produto que há em stock.

→ **stock_outro**

- *omedida :* é a medida usada para medir a quantidade de produto em stock (garrafas, pacotes, latas, molhos etc).
- *ounidade :* é o número de medidas de produto que há em stock.

→ **compras**

- *n_compra :* é o número da compra, é um número inteiro que identifica univocamente a compra.
- *datac :* dá-nos a data (ano-mês-dia) em que foi efectuada a compra.
- *total :* é a conta em euros do total gasto na compra.

→ **fornecedores**

- *id_forn :* é um número inteiro que identifica univocamente o fornecedor, funciona como o "número do bilhete de identidade" do fornecedor.
- *nome_forn :* é o nome do fornecedor.
- *descricao_forn :* é o tipo de que é o fornecedor (hipermercado, grossista, retalhista etc).

4.2. Conjunto de Relações

As relações servem para inter-ligar as diversas entidades, são associações entre as várias entidades. As relações facultam à base de dados um maior campo de manipulação pois tornam estes objectos menos estáticos.

Na nossa base de dados utilizamos as seguintes **relações**:

- **r_coz** – liga as entidades receitas e cozinhado.
- **fazer_receita** – liga as entidades receitas e produtos.
- **p_f** – liga as entidades produtos e familia_p.
- **prod_compra** – liga as entidades produtos e compras
- **fornecido** – liga as entidades compras e fornecedores

Notemos que temos **relações** com *atributos*:

→ **fazer_receita**

- *un_quant :* dá-nos informação acerca da quantidade de produto utilizado para fazer determinada receita.
- *med_quant :* dá-nos a respectiva medida.

→ **prod_compra**

- *preco_unid :* fornece-nos o preço por unidade de produto.
- *marca :* dá-nos a marca do produto comprado.
- *quant_unidade :* dá-nos informação acerca da quantidade de produto comprado
- *quant_medida :* dá-nos a respectiva medida.

4.3. Restrições de Mapeamento

As restrições de mapeamento dão-nos informação acerca do tipo de relação que existe entre as entidades ligadas. Restringem o número de entidades com as quais pode estar associada uma outra entidade num determinado conjunto de relações. Assim sendo temos restrições do tipo: um para um (1:1), um para muitos (1:N), muitos para um (N:1) e muitos para muitos (N:M).

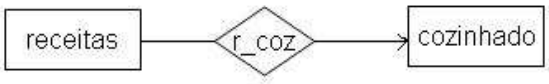
As restrições de mapeamento são expressas desenhando:

- uma seta (→), significando "um",
- uma linha (—), significando "muitos",

entre o conjunto de relações e o conjunto de entidades.

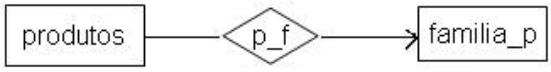
Na nossa base de dados temos as seguintes restrições:

→ **relação r_coz:** temos uma relação muitos para um, pois cada cozinhado pode referir-se a mais que uma receita mas cada receita diz respeito a um só cozinhado.



→ **relação fazer_receita:** temos uma relação muitos para muitos visto que, um produto pode entrar na composição de várias receitas e cada receita pode ter mais do que um produto.

→ **relação p_f:** aqui temos uma relação muitos para um, uma vez que cada família pode ter mais que um produto, por outro lado cada produto apenas pode pertencer a uma família.



→ **relação prod_compra:** neste caso estamos perante uma relação muitos para muitos pois um produto pode ser comprado mais do que uma vez e cada compra pode ter mais do que um produto comprado.

→ **relação fornecido:** temos aqui uma relação muitos para um visto que se pode fazer mais do que uma compra a um mesmo fornecedor e por outro lado cada compra diz respeito a um e um só fornecedor.

4.4. Chaves

No modelo relacional a única forma que existe para relacionar dados que existem numa tabela, com os dados de outra tabela, faz-se através de atributos comuns às duas tabelas. É nesta perspectiva que vemos a importância de atributos especiais (*chaves estrangeiras*) que vão fazer a ligação a outras tabelas em que esses mesmos atributos, identificam univocamente, cada uma das linhas (*chaves primárias*).

Temos então vários tipos de chaves que passamos a definir:

Superchave – associação de um ou mais atributos que, em conjunto, identificam univocamente um dos tuplos[1] (nota: no limite a associação de todos os campos de uma tabela constitui uma superchave).
Exemplo: Na nossa base de dados o conjunto de atributos (nome_p, id_p) constitui uma superchave da entidade produtos.

Chave candidata – subconjunto dos atributos de uma superchave que, sendo ainda superchave, não pode ser reduzido sem perder essa qualidade, ou seja, é uma superchave minimal.
Exemplo: (nome_p) e (id_p) são chaves candidatas da superchave anterior.

Chave primária – chave seleccionada entre as diferentes chaves candidatas para, efectivamente identificar cada tuplo.

Chave estrangeira – atributo, ou conjunto de atributos de uma relação, que é chave primária noutra relação.

[1] tuplo é cada instancia do esquema de relação, ou seja, cada linha de uma tabela é um tuplo da relação.

4.4.1. Chaves Primárias

Chaves primárias e respectivas entidades, da nossa base de dados:

ENTIDADES	CHAVE PRIMÁRIA
cozinhado	id_coz
receitas	id_r
produtos	id_p
familia_p	id_familia
stock_peso	id_p
stock_outro	id_p
compras	n_compra
fornecedores	id_forn

4.4.2. Chaves Estrangeiras

Após a simplificação de tabelas que iremos tratar mais tarde, obtemos várias chaves estrangeiras que passamos a enumerar:

ENTIDADE ou RELAÇÃO	CHAVES ESTRANGEIRAS
receitas	id_coz
fazer_receita	id_r e id_p
produtos	id_familia
prod_compra	id_p e n_compra
compras	id_forn

4.5. Participação

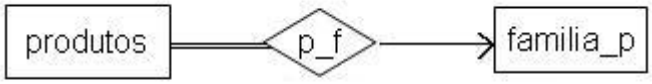
A participação de um conjunto de entidades num conjunto de relação pode ser de dois tipos:

* **participação total** (indicado por linha dupla no DER) onde toda a entidade do conjunto de entidades participa em pelo menos uma relação do conjunto de relações.

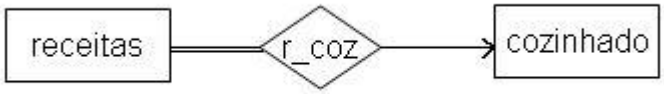
* **participação parcial** em que algumas entidades podem não participar em qualquer relação do conjunto de relações.

Vejamos as participações dos conjuntos de entidades nos conjuntos de relações usados na nossa base de dados:

Neste caso temos uma participação total do lado das compras e uma participação parcial do lado dos produtos isto porque, todas as compras têm pelo menos um produto e no caso dos produtos pode haver produtos que não digam respeito a qualquer compra.



Neste caso temos uma participação total do lado dos produtos e uma participação parcial do lado da familia_p pois todo o produto faz parte de uma família de produtos e, por outro lado, podem existir famílias de produtos que não digam respeito a qualquer produto usado na cozinha.

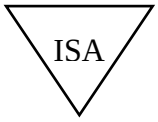


Aqui temos a participação total do lado das receitas e parcial do lado do cozinhado visto que, toda a receita se relaciona com um cozinhado mas pode haver cozinhados que não digam respeito a qualquer receita.

Estamos perante uma participação total das compras pois, todas as compras dizem respeito ao seu respectivo fornecedor, por outro lado, pode haver fornecedores que não tenham qualquer compra associada.

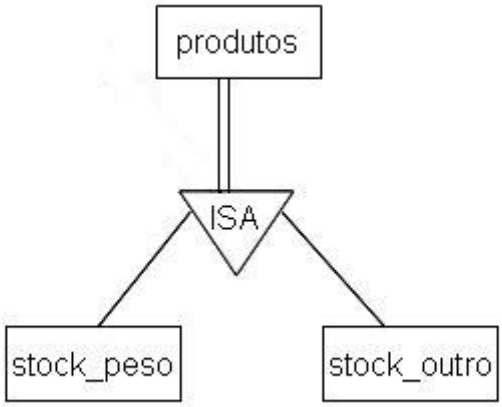
4.6. Especialização/ Generalização

Ao usar o método de desenho descendente, designamos subgrupos dentro de um conjunto de entidades que são distintas de outras entidades nesse conjunto. Estes subgrupos tornam-se conjuntos de entidades de menor nível que têm atributos ou participam em relações que não se aplicam ao conjunto de entidades de maior nível. No diagrama ER representamos a especialização com um triângulo anotado com ISA:



Nestes casos há herança de atributos: o conjunto de entidades de menor nível herda todos os atributos e participa em todas as relações do conjunto de entidades de maior nível ao qual está ligado.

Na nossa base de dados temos uma especialização em que a entidade de maior nível é a entidade produto e as entidades de menor nível são stock_peso e stock_outro.



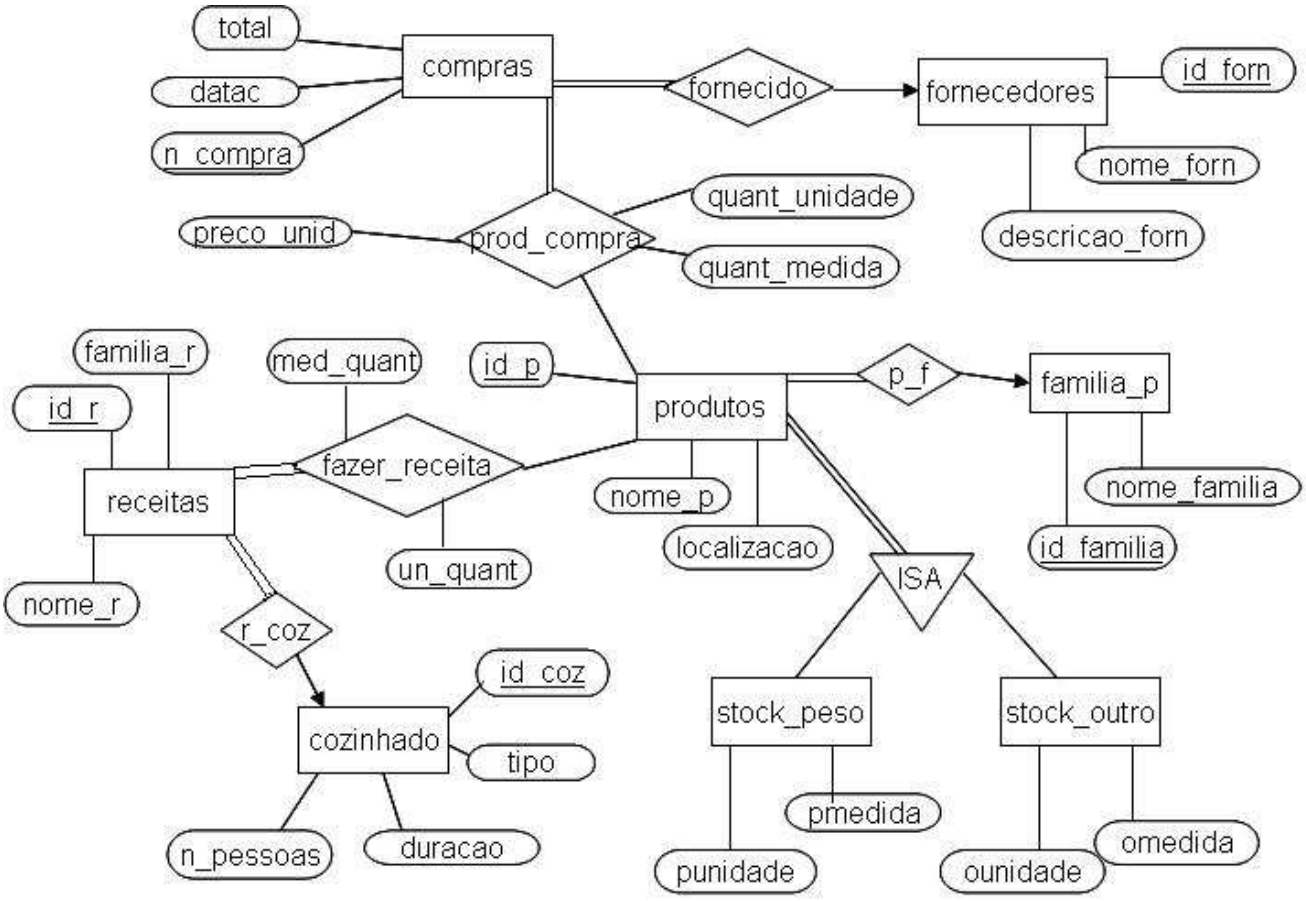
Nota: Neste caso temos a participação total dos produtos, pois todo o produto tem de estar associado a pelo menos um dos tipos de stock existentes.

4.7. Diagrama Entity-Relationship (DER)

Cada base de dados pode ser representada por um diagrama de entidade relacional. Este tipo de diagramas utiliza vários objectos geométricos para diferenciar os diferentes tipos de objectos, relações e restrições existentes na base de dados. Temos assim:

* **Rectângulos** para representar conjuntos de entidades.
* **Losangos** para representar conjuntos de associações.
* **Linha** que vão ligar atributos aos conjuntos de entidades e conjuntos de entidades a associações.
* **Elipses** para representar atributos:
  - **elipses duplas** para representar atributos multivalor.
  - **elipses tracejadas** para representar atributos derivados.
* **Sublinhado** que vai representar os atributos que fazem parte da chave primária.

4.7.1. DER para COZINHA



Capítulo 5. Tabelas do Modelo Relacional

O modelo relacional estabelece claramente as regras para a divisão da informação entre tabelas, de forma a evitar a duplicação de informação. Sendo assim a manipulação de dados da base de dados torna-se mais simples.

Uma base de dados que seja representável por um DER pode ser também representada por intermédio de um conjunto de relações. Para cada conjunto de entidades e para cada conjunto de associações gera-se uma única relação (ou tabela) com o nome do conjunto de entidades ou conjunto de associações respectivo.

5.1. Tabelas

Apresentamos de seguida as tabelas da base de dados COZINHA:

cozinhado ({ id_coz, n_pessoas, duracao, tipo })

r_coz ({id_coz, id_r})

receitas({id_r, nome_r, familia_r })

fazer_receita({id_r, med_quant, un_quant, id_p})

produtos({id_p, nome_p, localizacao})

stock_peso({id_p, pmedida, punidade})

stock_outro({id_p, omedida, ounidade})

p_f({id_p, id_familia})

familia_p({id_familia, nome_familia})

prod_compra({id_p, preco_unid, marca, quant_unidade, quant_medida, n_compra})

compras({n_compra, datac, total})

fornecido({n_compra, id_forn})

fornecedores({id_forn, nome_forn, descricao_forn})

OBERVAÇÕES: Note-se que os campos de uma tabela correspondente a uma entidade, do DER, são os atributos dessa entidade. No caso de cada relação do DER, as tabelas correspondentes têm como campos as chaves primárias das entidades ligadas por essa relação.

5.2. Tabelas Simplificadas

Vamos agora simplificar as tabelas tendo em conta os seguintes aspectos:

- Um conjunto de associações N:M é representado com uma tabela com colunas para as chaves primárias dos dois conjuntos de entidades participantes, com colunas adicionais para os atributos próprios (ou descritivos) do conjunto de associações.
- Conjuntos de associações 1:N e N:1, totais no lado muitos podem ser representados adicionando atributos extra ao lado muitos contendo a chave primária do outro conjunto participante.
- No caso da generalização, forma-se uma tabela para a entidade de maior nível (mais geral) e cria-se uma tabela para cada conjunto de entidades de nível abaixo, incluindo a chave primária da entidade acima e os atributos locais.

cozinhado ({ id_coz, n_pessoas, duracao, tipo })

receitas({id_r, nome_r, família_r, id_coz })

fazer_receita({id_r, med_quant, un_quant, id_p})

produtos({id_p, nome_p, localizacao, id_familia})

stock_peso({id_p, pmedida, punidade})

stock_outro({id_p, omedida, ounidade})

familia_p({id_familia, nome_familia})

prod_compra({id_p, preco_unid, marca, quant_unidade, quant_medida, n_compra})

compras({n_compra, datac, total, id_forn})

fornecedores({id_forn, nome_forn, descricao_forn})

Capítulo 6. Listagem dos dados das tabelas

6.1. Tabela produtos

id_p	nome_p	localizacao	id_familia
1	cuscus	prateleira1	1
2	esparguete	prateleira1	1
3	espiral	prateleira1	1
4	buzios	prateleira1	1
5	arroz agulha	prateleira1	2
6	arroz carolino	prateleira1	2
7	batata	despensa1	6
8	oregaos	gaveta	8
9	pimenta	gaveta	8
10	agua	prateleira2	9
11	lima	frigorifico	10
12	pao	cesto	7
13	alheira	congelador1	11
14	cenouras	frigorifico	5
15	cebolas	despensa1	5
16	sumo	prateleira2	9
17	coca-cola	prateleira2	9
18	bacalhau	congelador2	3
19	pescada	congelador2	3
20	coelho	congelador1	4
21	carne picada	congelador1	4
22	chocos	congelador2	3
23	febra de porco	congelador1	4
24	banana	fruteira	10
25	laranja	fruteira	10
26	alface	frigorifico	5
27	azeite	prateleira3	0
28	oleo	prateleira3	0
29	sal	prateleira3	0
30	alho	despensa1	5

id_p	nome_p	localizacao	id_familia
31	alho frances	frigorifico	5
32	ovos	frigorifico	0
33	polvo	congelador2	3
34	leite condensado	prateleira2	12
35	leite	prateleira2	12
36	acucar	prateleira1	0
37	sardinha	frigorifico	3
38	fiambre	frigorifico	4
39	milho	prateleira1	5
40	agriao	frigorifico	5
41	feijao verde	frigorifico	5
42	feijao preto	frigorifico	5
43	feijao vermelho	frigorifico	5
44	vinagre	prateleira3	0
45	natas	frigorifico	12
46	bolacha	prateleira2	7
47	chocolate	prateleira2	7
48	espinafres	frigorifico	5
49	gelatina	prateleira2	7
50	couve galega	frigorifico	5
51	ervilhas	congelador	5
52	cachaca	garrafeira	9
53	cerveja	garrafeira	9
54	vinho tinto	garrafeira	9
55	vinho branco	garrafeira	9
56	galinha	congelador1	4
57	maca	fruteira	10
58	ananas	frigorifico	10
59	carne de porco	congelador1	4

6.2. Tabela receitas

id_r	nome_r	familia_r	id_coz
1	baba de camelo	sobremesa	9
2	coelho estufado	carne	6
3	alheira com ovo	carne	5
4	empadao de carne	carne	13
5	caldo verde	entrada	4
6	chocos	peixe	3
7	sopa de espinafres	entrada	4
8	gelatina	sobremesa	9
9	sopa de legumes	entrada	4
10	canja	entrada	11
11	salada de fruta	sobremesa	14
12	ovos escalfados	NULL	10
13	salada primavera	*NULL*	14
14	mousse	sobremesa	9
15	sardinha assada	peixe	3
16	sopa de agriao	entrada	11
17	arroz de polvo	peixe	6
18	prego no prato	carne	5
19	massa a bolonhesa	carne	6
20	pudim	sobremesa	12
21	caipirinha	bebida	15
22	natas do ceu	sobremesa	12
23	pescada cozida	peixe	2

6.3. Tabela fazer_receita

id_r	med_quant	un_quant	id_p
1	dúzia	1.00	32
1	lata	2.00	34
1	kg	0.50	36
2	kg	0.50	5
2	kg	0.50	15
2	kg	2.00	20
2	litro	0.10	27
3	kg	2.50	13
3	litro	0.20	28
3	duzia	6.00	32
4	kg	3.50	7
4	kg	0.25	15
4	kg	2.00	21
4	kg	0.25	26
4	duzia	0.50	32
5	kg	1.50	7
5	litro	0.25	27
5	kg	0.02	29
5	kg	0.50	50
6	kg	0.50	7
6	kg	1.50	22
6	kitro	0.01	27
7	kg	1.50	7
7	kg	0.75	14
7	kg	0.25	15
7	litro	0.02	27
7	kg	0.02	29
7	kg	0.75	48
8	pacote	1.00	49
9	kg	1.50	7
9	kg	0.25	15
9	litro	0.25	27
9	kg	0.02	29
9	kg	1.00	31
9	lata	1.00	43
10	pacote	1.00	1
10	pacote	0.01	29
10	kg	1.00	56
11	kg	0.50	24
11	pacote	1.00	25
11	kg	0.75	57

id_r	med_quant	un_quant	id_p
11	kg	0.75	58
12	kg	0.01	29
12	duzia	0.50	32
12	pacote	1.00	51
12	kg	0.50	59
13	kg	0.25	26
13	duzia	0.50	32
13	kg	0.25	38
13	lata	1.00	39
13	kg	0.25	58
14	duzia	0.50	32
14	kg	0.50	36
14	tablete	1.00	47
15	kg	0.75	7
15	litro	0.01	27
15	kg	0.01	29
15	kg	1.00	37
16	kg	0.50	7
16	kg	0.25	14
16	litro	0.01	27
16	kg	0.01	29
16	kg	0.01	30
16	molho	1.00	40
16	lata	0.50	43
17	kg	0.75	6
17	kg	0.20	15
17	litro	0.02	28
17	kg	0.02	29
17	kg	0.01	30
17	kg	1.00	33
18	kg	0.75	5
18	kg	1.00	7
18	kg	1.25	23
18	litro	0.10	28
18	kg	0.01	29
18	kg	0.01	30
18	duzia	0.50	32
19	pacote	1.00	2
19	kg	0.25	15
19	kg	1.00	21
19	kg	0.01	29

id_r	med_quant	un_quant	id_p
19	litro	0.10	55
20	duzia	1.00	32
20	litro	1.00	35
20	kg	0.75	36
21	kg	0.50	11
21	litro	1.50	52
22	duzia	1.00	32
22	kg	0.75	36
22	pacote	2.00	45
22	pacote	0.50	46
23	kg	0.75	7
23	kg	0.50	14
23	kg	1.00	19
23	litro	0.01	27
23	kg	0.02	29

6.4. Tabela cozinhado

id_coz	n_pessoas	duracao	tipo
1	10	60	cozido
2	4	20	cozido
3	4	30	grelhado
4	10	60	sopa
5	6	45	frito
6	6	60	estufado
7	8	60	assado
8	8	120	assado
9	4	30	doce
10	6	30	escalfado
11	4	30	sopa
12	8	60	doce
13	10	90	outro
14	4	20	outro
15	6	10	bebida

6.5. Tabela stock_peso

id_p	pmedida	punidade
1	kg	1.00
2	kg	6.00
3	kg	3.00
4	kg	1.00
5	kg	20.00
6	kg	5.00
7	kg	70.00
10	litro	100.00
11	kg	1.00
13	kg	4.00
14	kg	5.00
15	kg	10.00
16	litro	12.00
17	litro	6.00
18	kg	25.00
19	kg	4.00
20	kg	4.00
21	kg	4.00
22	kg	3.00
23	kg	7.00
24	kg	2.00
25	kg	6.00
26	kg	1.00
27	litro	3.00
28	litro	7.00
29	kg	10.00
30	kg	4.00
31	kg	2.00
33	kg	6.00
35	litro	30.00
36	kg	15.00
37	kg	5.00
38	kg	1.00
41	kg	2.00
44	litro	9.00
48	kg	0.00
50	kg	0.00
52	litro	5.00
53	litro	10.56
54	litro	3.75
55	litro	1.50
56	kg	15.00
57	kg	3.00
58	kg	5.00
59	kg	20.00

6.6. Tabela stock_outro

id_p	omedida	ounidade
1	pacote	4.00
2	pacote	12.00
3	pacote	6.00
4	pacote	2.00
5	pacote	5.00
6	pacote	6.00
8	pacote	3.00
9	pacote	1.00
10	garrafa	100.00
12	unidade	0.00
16	garrafa	8.00
17	garrafa	4.00
27	garrafa	3.00
28	garrafa	7.00
32	duzia	2.00
34	lata	5.00
35	pacote	30.00
39	lata	5.00
40	molho	0.00
42	lata	2.00
43	lata	4.00
44	garrafa	9.00
45	pacote	5.00
46	pacote	10.00
47	tablete	6.00
49	pacote	5.00
51	pacote	4.00
52	garrafa	5.00
53	garrafa	32.00
54	garrafa	5.00
55	garrafa	2.00

6.7. Tabela compras

n_compra	total	datac	id_forn
1	233.20	30-11-2005	4578
2	195.50	30-11-2005	1127
3	105.95	02-12-2005	5212
4	18.93	05-12-2005	7137
5	16.74	06-12-2005	6912
6	130.05	07-12-2005	5920
7	17.46	09-12-2005	5563
8	32.15	13-12-2005	7798
9	37.20	14-12-2005	2472
10	15.40	17-12-2005	9727
11	12.49	18-12-2005	2465
12	57.50	20-12-2005	2465
13	54.20	22-12-2005	7137

6.8. Tabela prod_compra

id_p	preco_unid	marca	quant_unidade	quant_medida	n_compra
2	0.44	nacional	5	pacote	4
3	0.35	triunfo	4	pacote	11
5	0.85	cigala	15	kg	3
7	0.42	NULL	50	kg	1
8	0.60	auchan	3	pacote	11
9	0.70	auchan	4	pacote	11
10	0.35	caramulo	250	garrafa	1
11	2.10	NULL	1	kg	11
12	0.10	NULL	50	unidade	4
13	3.70	NULL	3	kg	10
15	0.40	NULL	10	kg	3
16	1.10	sumol	12	garrafa	9
17	1.20	cocacola	6	garrafa	9
18	10.25	ribeiralves	20	kg	1
19	2.12	NULL	3	kg	4
20	8.50	NULL	4	kg	3
21	4.20	NULL	4	kg	9
22	11.40	NULL	3	kg	3
24	1.80	NULL	2	kg	11
25	0.90	NULL	5	kg	4

id_p	preco_unid	marca	quant_unidade	quant_medida	n_compra
26	0.82	NULL	1	kg	4
27	3.50	oliveira da serra	6	garrafa	3
27	5.00	galo	10	litro	13
28	0.99	fula	6	litro	5
28	0.84	vaqueiro	5	litro	13
29	0.20	vatel	5	kg	10
30	3.40	NULL	2	kg	8
31	1.90	NULL	2	kg	6
32	1.20	NULL	3	duzia	5
33	15.00	NULL	6	kg	6
35	0.90	gresso	24	litro	8
36	0.95	rar	10	kg	7
37	7.25	NULL	3	kg	6
39	0.94	bonduelle	4	lata	7
40	1.80	NULL	4	kg	1
41	2.00	NULL	2	kg	5
44	0.80	continente	4	garrafa	5
45	0.55	mimosa	6	pacote	10
47	1.40	nestle	3	tablete	7
50	0.79	NULL	1	kg	11
51	1.25	pescanova	3	pacote	8
52	7.30	velho barreiro	5	garrafeira	12
53	0.30	sagres	12	garrafa	1
54	1.25	borba	30	litro	1
55	1.15	borba	30	litro	1
55	3.50	casal garcia	5	garrafa	12

6.9. Tabela fornecedores

id_forn	nome_forn	decricao_forn
1127	carrefur	grossista
2465	pingo doce	hipermercado
2472	Tofasil	retalhista
3530	Lidl	hipermercado
4578	continente	hipermercado
5212	modelo	hipermercado
5563	recheio	grossista
5920	Jumbo	hipermecado
6912	ze manel	mini_mercado
7137	Frescos e companhia	supermercado
7798	Makro	grossista
8372	caves de coimbra	retalhista
9727	intermarche	hipermercado

6.10. Tabela familia_p

id_familia	nome_familia
0	outro
1	massa
2	arroz
3	peixe
4	carne
5	vegetal
6	batata
7	pastelaria
8	especiarias
9	bebida
10	fruta
11	enchidos
12	lacticinios

Capítulo 7. Álgebra Relacional versus SQL

Na matemática, uma álgebra é um conjunto de objectos e um conjunto de operações sobre estes objectos.

A álgebra relacional foi desenvolvida para descrever operações sobre uma base de dados relacional. O conjunto de objectos são as tabelas e uma operação possui como operandos e como resultado tabelas.

A compreensão da álgebra relacional permite-nos ter uma melhor visão sobre a linguagem SQL, pois o SQL incorpora cada vez mais conceitos de álgebra relacional.

O SQL (*Structured Query Language*) é uma linguagem descritiva de manipulação de bases de dados que se baseia em operações de conjuntos de álgebra relacional. Podemos utilizar o SQL para consultar, actualizar, e gerir bases de dados relacionais. Para isso vamos usar o programa *MySQL Server 5.0.16.*

Observação: Um aspecto relevante da linguagem SQL é a utilzação de ***null*** , é possível que um tuplo tenha um valor nulo, denotado por *null*, para algum dos seus atributos, o que significa que o seu valor é desconhecido ou não existe. O resultado de qualquer expressão aritmética envolvendo um *null* é *null.*

Este capítulo é puramente teórico, sendo dados exemplos aplicados à nossa base de dados mais tarde.(capitulo 10)

7.1. Operadores Básicos

Existem operadores básicos de álgebra relacional, são eles:

* Selecção,
* Projecção,
* União,
* Diferença de conjuntos,
* Produto cartesiano,
* Renomeação.

Os comandos principais de SQL:

* SELECT,
* FROM,
* WHERE.

7.1.1. Selecção / WHERE

Álgebra Relacional:

A Selecção tem como operando uma tabela. O resultado é uma tabela que contém as linhas que obedecem a um determinado critério p.

Notação: $\sigma_p(r)$ Em que *p* é uma fórmula do cálculo proposicional constituída por termos ligados por: $\wedge$ (e), $\vee$ (ou), $\neg$ (não)

Comandos SQL:

A cláusula *WHERE* corresponde ao predicado de selecção da álgebra relacional. É formada por um predicado envolvendo atributos de relações que aparecem na cláusula *FROM*.

7.1.2. Projecção / SELECT

Álgebra Relacional:

A Projecção tem como operando uma tabela. O resultado é uma tabela que contém apenas as colunas indicadas, $A_1, ..., A_k$.

Notação: $\Pi_{A1, A2, ..., Ak}(r)$ em que $A_1, ..., A_k$ são nomes de atributos e *r* é uma relação.

Comandos SQL:

A cláusula *SELECT* corresponde à operação de projecção da álgebra relacional. É utilizada para listar os atributos pretendidos no resultado da consulta.

Para que no resultado da utilização da cláusula *select* não haja resultados repetidos usamos a palavra DISTINCT depois da palavra SELECT.

Mas, quando desejamos que nenhum dos duplicados seja removido, utilizamos a palavra ALL depois da palavra SELECT (no caso de não pormos nem DISTINCT nem ALL, o resultado é o mesmo aquando a utilização da palavra ALL).

7.1.3. União / UNION

Álgebra Relacional:

A União possui duas tabelas como operandos. As tabelas devem ser compatíveis para união:

→ Possuir o mesmo número de colunas,

→ O domínio da i -ésima coluna de uma tabela deve ser idêntico ao domínio da i -ésima coluna de outra.

Notação: $r \cup s$

Comandos SQL:

Uma união não é propriamente uma ligação entre tabelas. A cláusula *union* permite juntar o conteúdo de dois ou mais comandos SELECT.

→ Numa *UNION* o nome das colunas apresentado no resultado é o nome das colunas seleccionadas na primeira instrução SELECT,

→ Numa *UNION* o número de campos a seleccionar em cada um dos comandos SELECT tem de ser igual. O nome dos campos não é relevantes, mas o tipo de dados que pode ser agrupado depende de sistema para sistema.

7.1.4. Diferença de Conjuntos / EXCEPT

Álgebra Relacional:

A Diferença de Conjuntos possui duas tabelas como operandos. As tabelas devem ser compatíveis tal como na união.

Notação: $r - s$

Comandos SQL:

Em SQL a cláusula usada para a diferença de conjuntos é *except*. A cláusula *except* permite fazer a diferença do conteúdo de dois comandos SELECT.

7.1.5. Produto Cartesiano

Álgebra Relacional:

O Produto Cartesiano possui como operandos duas tabelas. O resultado é uma tabela cujas linhas são a combinação das linhas das tabelas operandas, tomando-se uma linha de uma das tabelas e concatenando-a com uma linha da outra tabela.

Total de colunas do produto cartesiano = Nº de colunas da 1ª tabela + Nº de colunas da 2ª tabela
Total de linhas do produto cartesiano = Nº de linhas da 1ª tabela × Nº de linhas da 2ª tabela

Notação: $r \times s$

Comandos SQL:

Para se recorrer ao produto cartesiano utiliza-se a cláusula SELECT ... FROM ..., mas declarando várias tabelas nos argumentos de FROM.

7.1.6. Renomeação / AS

Álgebra Relacional:

Operador para atribuir (dentro de uma consulta) um novo nome a uma tabela.

Notação: $\rho_x(E)$

Comandos SQL:

A linguagem SQL permite a renomeação de relações e atributos recorrendo à cláusula ***AS***. Na utilização do produto cartesiano, as relações têm de ser disjuntas. Assim, a renomeação é útil nestes casos.

7.2. Operadores Adicionais

Definem-se outras operações que não aumentam o poder expressivo da álgebra relacional, mas simplificam algumas consultas habituais.

* Intersecção de Conjuntos,
* Junção Natural,
* Divisão,
* Atribuição.

Em SQL temos outros comandos de grande utilidade como por exemplo:

* INTERSECT
* INNER JOIN
* ORDER BY,
* entre outros….

7.2.1. Intersecção de Conjuntos / INTERSECT

Álgebra Relacional:

A Intersecção possui duas tabelas como operandos. As tabelas devem ser compatíveis para intersecção:

→ Possuir o mesmo número de colunas,

→ O domínio da i -ésima coluna de uma tabela deve ser idêntico ao domínio da i -ésima coluna de outra.

Notação: $r \cap s$

Comandos SQL:

A cláusula *INTERSECT* permite juntar o resultado de dois comandos SELECT, apresentando apenas as linhas que resultam de ambos os comandos.

7.2.2. Junção Natural / INNER JOIN

Álgebra Relacional:

A combinação de uma operação de selecção aplicada sobre uma operação de produto cartesiano é usual em aplicações de bases de dados. É através dela que dados de tabelas relacionadas são associados. Por isso, foi criada a operação de junção, que corresponde exactamente à sequência de operações em questão.

O resultado da junção natural é uma relação no esquema $R \cup S$ que é obtido considerando cada par de tuplos t_r de r e t_s de s.

Notação: r ⋈ s

Comandos SQL:

Utiliza-se a cláusula *INNER JOIN* para efectuar a junção natural entre tabelas. Reparemos que a cláusula *on* serve para explicitar o argumento de base para a associação das tabelas. Existem algumas variantes na utilização desta função.

7.2.3. Divisão

Álgebra Relacional:

Como a Junção, a Divisão é uma operação de álgebra relacional que pode ser construída a partir de outras. O seu resultado é uma relação no esquema $R - S = (A_1, \ldots, A_m)$

A divisão é adequada para consultas que incluam a frase “para todo”.

Notação: r ÷ s

Comandos SQL:

Não existe um comando específico que corresponda ao operador ÷ da álgebra relacional. De qualquer forma, é possível fazer consultas que nos dêem o resultado do operador ÷, como veremos num exemplo do capítulo 10.

7.2.4. Atribuição

Álgebra Relacional:

A operação de atribuição permite-nos expressar consultas complexas de uma forma muito conveniente. Escreve-se a consulta como um programa sequencial constituído por uma sequência de atribuições terminada com uma expressão cujo valor é o resultado da consulta.

Notação: $\leftarrow$

7.2.5. ORDER BY

Comandos SQL:

É possível ordenar os resultados em função de um determinado atributo. Esta listagem pode ser efectuada por ordem ascendente ou descendente. Para listarmos de Z para A depois de ORDER BY podemos escrever DESC, para listarmos de A para Z depois de ORDER BY podemos escrever ASC.

7.2.6. SOME, IN, ALL, EXISTS

- F <op> SOME $r \Leftrightarrow \exists\, t \in r$: (F <op> t) , em que <op> pode ser : <, ≤, >, =, ≠
  (= SOME) ≡ IN

- F <op> ALL $r \Leftrightarrow \forall t \in r$: (F <op> t)
  (≠ ALL) ≡ NOT IN

- A construção EXISTS devolve o valor **true** se a subconsulta é não vazia.
  EXISTS $r \Leftrightarrow r \neq \varnothing$
  NOT EXISTS $r \Leftrightarrow r = \varnothing$

7.3. Funções de Agregação e Junção Externa

Existem operações estendidas da álgebra relacional que aumentam a sua expressividade:

* Projecção Generalizada,
* Funções de Agregação,
* Junção Externa.

↳ FUNÇÕES DE AGREGAÇÃO

Álgebra Relacional:

As funções de agregação aplicam-se a uma colecção de valores e devolvem um único valor como resultado. Temos como funções de agregação:

- avg: calcula a média de valores
- min: calcula mínimo de um conjunto de valores
- max: calcula máximo de um conjunto de valores
- sum: calcula a soma de um determinado conjunto de valores
- count: conta o número de valores de um conjunto

Notação: A operação de agregação tem a seguinte notação:

$$_{G1, G2, ..., Gn}\mathcal{G}_{F1(A1), F2(A2),..., Fn(An)}(E)$$

onde,

E é uma expressão de álgebra relacional
*G*1, *G*2 …, *G*n é uma lista de atributos de agrupamento (pode ser vazia)
Cada *Fi* é uma função de agregação
Cada *Ai* é um nome de um atributo

Comandos SQL:

* As funções de agregação têm por objectivo obter informação sobre conjuntos de linhas especificados na cláusula WHERE.
* A cláusula HAVING serve para fazer restrições ao nível dos grupos que são processados. No entanto, se pretendermos mostrar os grupos que apresentam uma característica em particular, não utilizamos a cláusula WHERE, pois esta destina-se à restrição das linhas. Utiliza-se a cláusula HAVING, que actua unicamente sobre o resultado dos grupos.
* A cláusula GROUP BY permite agrupar as informações guardadas numa determinada tabela.

↳ JUNÇÃO EXTERNA

As operações de junção retornam uma relação como resultado da combinação de duas outras relações. Estas operações adicionais são utilizadas habitualmente em subconsultas na cláusula FROM.

Uma extensão da operação de junção que evita a perda de informação. Calcula a junção e depois adiciona ao resultado os tuplos de uma relação que não estão relacionados com a outra relação na junção. Temos como exemplos LEFT JOIN e RIGHT JOIN.

7.4. Modificação da Base de Dados

Além das operações atrás referidas da álgebra relacional, é também possível modificar uma base de dados. Estas modificações são possíveis recorrendo às seguintes operações:

* Remoção,
* Inserção,
* Actualização.

Utilizamos o operador de atribuição para expressar todas estas operações.

7.4.1. Remoção

Álgebra Relacional:

Uma operação de remoção é expressa de uma maneira semelhante a uma consulta, sendo os tuplos seleccionados removidos da base de dados.

Só se podem remover tuplos integralmente; não se podem apagar valores de determinados atributos.

Notação: Uma remoção é expressa em álgebra relacional por:

$$r \leftarrow r - E$$

em que r é uma relação e E é uma operação de álgebra relacional.

Comandos SQL:

A remoção de tuplos de uma tabela é feita em SQL com a instrução

```
DELETE FROM  <tabela>
WHERE  <Condição>
```

7.4.2. Inserção

Álgebra Relacional:

Como era de esperar, a inserção baseia-se no uso da reunião. Para que esta operação fique bem definida deveremos especificar o objecto a ser inserido, ou descrever uma determinada consulta de maneira a que ela descreva o objecto ou conjunto de objectos a inserir.

Notação: Na álgebra relacional, uma inserção é expressa por: $r \leftarrow r \cup E$, em que r é uma relação e E é uma expressão de álgebra relacional.

Comandos SQL:

A inserção de tuplos numa tabela é feita em SQL com a instrução

```
INSERT INTO  <tabela>
VALUES  <Conjunto de tuplos>
```

7.4.3. Actualização

Álgebra Relacional:

Tal como as operações anteriores, esta também de baseia numa das operações básicas existentes para base de dados.

Notação: Esta operação é descrita por:

$$r \leftarrow \Pi_{F1, F2, \ldots, Fl, (r)}$$

onde cada *Fi* é um atributo de *r* no caso de o i-ésimo atributo não ser alterado, ou é uma expressão que indica a actualização que deve ser efectuada no i-ésimo atributo.

Comandos SQL:

A actualização de tuplos duma tabela é feita em SQL com a instrução

```
UPDATE  <tabela>
SET  <Atributo> = <Expressão>, <Atributo> = <Expressão>, ...
WHERE  <Condição>
```

Capítulo 8. Criação da Base de Dados COZINHA

Antes de inserir, remover ou alterar informação, ou mesmo efectuar qualquer tipo de consultas, temos de definir as tabelas existentes na base de dados. Ao criar essas tabelas temos de atribuir um domínio a cada campo, o que é feito recorrendo à instrução CREATE TABLE.

Temos de ter em conta a existência de restrições de integridade aquando a criação de tabelas, são elas:

- → **not null**
- → **primary key** (*A*1, ..., *An)*
- → **unique** (*A*1, ..., *An)*
- → **foreign key**(Ai) **references** <name_table> (Ai)
- → **check** *(P)*, em que *P* é um predicado

Como sabemos cada tabela deve conter uma chave primária, ou seja, um atributo ou um conjunto de atributos, que permitam identificar univocamente cada tuplo da base de dados. A selecção da chave é feita a partir da cláusula PRIMARY KEY.

8.1. produtos

```
CREATE TABLE produtos (
                        id_p          int(11)     NOT NULL UNIQUE,
                        nome_p        char(30)    NOT NULL,
                        localizacao   char(30)    default NULL,
                        id_familia    int(11)     NOT NULL,
                        PRIMARY KEY  (id_p)
                       );
```

8.2. receitas

```
CREATE TABLE  receitas  (
                         id_r          int(11)     NOT NULL UNIQUE,
                         nome_r        char(30)    NOT NULL,
                         familia_r     char(30)    default NULL,
                         id_coz        int(11)     NOT NULL,
                         PRIMARY KEY  (id_r)
                        );
```

8.3. cozinhado

```
CREATE TABLE cozinhado (
                        id_coz        int(11)    NOT NULL UNIQUE,
                        n_pessoas     int(11)    default NULL,
                        duracao       int(11)    default NULL,
                        tipo          char(30)   default NULL,
                        PRIMARY KEY  (id_coz),
                        CHECK(duracao>=0 and n_pessoas>=0)
                        );
```

8.4. stock_peso

```
CREATE TABLE stock_peso (
                          id_p          int(11)          NOT NULL UNIQUE,
                          pmedida       char(20)         default NULL,
                          punidade      double(10,2) default NULL,
                          PRIMARY KEY  (id_p),
                          CHECK(punidade>=0,0)
                        );
```

8.5. stock_outro

```
CREATE TABLE stock_outro (
                           id_p          int(11)          NOT NULL UNIQUE,
                           omedida       char(20)         default NULL,
                           ounidade      double(10,2)  default NULL,
                           PRIMARY KEY  (id_p),
                           CHECK(ounidade>=0,0)
                          );
```

8.6. compras

```
CREATE TABLE compras (
                       n_compra    int(11)          NOT NULL UNIQUE,
                       total       double(10,2)  NOT NULL,
                       datac       date             NOT NULL,
                       id_forn     int(11)          NOT NULL,
                       PRIMARY KEY  (n_compra),
                       CHECK(total>=0,0)
                      );
```

8.7. fornecedores

```
CREATE TABLE fornecedores (
                    id_forn          int(11)    NOT NULL UNIQUE,
                    nome_forn        char(30)   NOT NULL,
                    descricao_forn   char(40)   default NULL,
                    PRIMARY KEY  (id_forn)
                    );
```

8.8. familia_p

```
CREATE TABLE familia_p (
                    id_familia       int(11)    NOT NULL UNIQUE,
                    nome_familia     char(30)   NOT NULL,
                    PRIMARY KEY  (id_familia)
                    );
```

8.9. prod_compra

```
CREATE TABLE prod_compra (
                    id_p             int(11)          NOT NULL,
                    preco_unid       double(10,2)     NOT NULL,
                    marca            char(30)         default NULL,
                    quant_unidade    double(10,2)     NOT NULL,
                    quant_medida     char(20)         NOT NULL,
                    n_compra         int(11)          NOT NULL,
                    PRIMARY KEY (id_p, n_compra),
                    FOREIGN KEY(id_p) REFERENCES produtos(id_p),
                    FOREIGN KEY(n_compra) REFERENCES compras(n_compra),
                    CHECK(preco_unid>=0,0 and quant_unidade>=0,0),
                    );
```

8.10. fazer_receita

```
CREATE TABLE fazer_receita (
                    id_r             int(11)          NOT NULL,
                    med_quant        char(20)         NOT NULL,
                    un_quant         double(10,2)     NOT NULL,
                    id_p             int(11)          NOT NULL,
                    PRIMARY KEY  ( id_r, id_p),
                    FOREIGN KEY(id_r) REFERENCES receitas(id_r),
                    FOREIGN KEY(id_p) REFERENCES produtos(id_p),
                    CHECK(un_quant>=0,0)
                    );
```

Capítulo 9. Modificação da Base de Dados COZINHA

9.1. Remoção

- **Remoção da tabela receitas:**

SQL:

```
drop table cozinhado;
drop table receitas;
drop table fazer_receita;
```

- **Remoção de um tuplo: remover o produto azeite:**

Álgebra relacional

$t \leftarrow \sigma_{(nome\_p='azeite')}(produto)$
$stock\_outro \leftarrow stock\_outro - (\Pi_{\{id\_p, omedida, ounidade\}}(t \blacktriangleright\blacktriangleleft stock\_outro))$
$stock\_peso \leftarrow stock\_peso - (\Pi_{\{id\_p, pmedida, punidade\}}(t \blacktriangleright\blacktriangleleft stock\_peso))$
$produtos \leftarrow produtos - \Pi_{\{id\_p, nome\_p, localizacao,\}}(t)$

SQL:

```
DELETE FROM stock_outro
WHERE id_p IN
(
    SELECT produtos.id_p
    FROM produtos
    WHERE nome_p = 'azeite'
 );


DELETE FROM stock_peso
WHERE id_p IN
(
    SELECT id_p
    FROM produtos
    WHERE nome_p = 'azeite'
 );

 DELETE FROM produtos
  WHERE  nome_p = 'azeite';
```

9.2. Inserção

Voltar a inserir a tabela é processo que já foi explicado no capítulo criar tabelas.

- **Inserir o produto azeite, novamente:**

Álgebra Relacional:

produtos ← produtos ∪ {(27,'azeite','prateleira3',0)}
stock_outro← stock_outro ∪ {(27, 'garrafa', 3.00)}
stock_peso ← stock_peso ∪ {(27, 'azeite', 3.00)}

SQL:

```
INSERT INTO produtos VALUES (27, 'azeite',  'prateleira3', 0);
INSERT INTO stock_outro VALUES (27, 'garrafa',    3.00);
INSERT INTO stock_peso VALUES (27, 'litro',         3.00);
```

9.3. Actualização

- **Aumentar o stock dos produtos em 5 unidades:**

stock_outro ←stock_outro ∪ $\Pi_{\{id\_p,\ omedida,\ ounidade+5.0\}}$(stock_outro)
stock_peso ←stock_peso ∪ $\Pi_{\{id\_p,\ pmedida,\ punidade+5.0\}}$(stock_peso)

SQL:

```
UPDATE stock_outro SET ounidade = ounidade +5.0;
UPDATE stock_peso SET punidade = punidade +5.0;
```

- **Aumentar em 10 minutos todos os cozinhados com duração superior a 30 minutos:**

Álgebra relacional:

cozinhado ← cozinhado ∪ $\Pi_{\{id\_coz,\ n\_pessoas,\ duracao+10,\ tipo\ \}}$($\sigma_{(\ duracao>30)}$(cozinhado))

SQL:

```
UPDATE cozinhado SET duracao = duracao +10  WHERE duracao >30;
```

Capítulo 10. Consultas

Todas as consultas aqui exemplificadas foram testadas em *MySQL Server 5.0.16*.

PERGUNTA 1

Linguagem corrente:

Quais os produtos comprados no fornecedor continente?

Álgebra relacional:

$\Pi_{\{nome\_p\}}$(produtos⋈prod_compra⋈compras⋈ $\sigma_{(nome\_forn='continente')}$(fornecedores))

ou

$\Pi_{\{nome\_p\}}$($\sigma_{(nome\_forn='continente')}$[produtos⋈prod_compra⋈compras⋈ fornecedores])

SQL:

```
SELECT nome_p
FROM produtos, prod_compra, compras, fornecedores
WHERE produtos.id_p = prod_compra.id_p
AND prod_compra.n_compra = compras.n_compra
AND compras.id_forn = fornecedores.id_forn
AND fornecedores.nome_forn = 'continente';
```

ou

```
SELECT nome_p
FROM produtos
INNER JOIN prod_compra
INNER JOIN compras
INNER JOIN fornecedores ON ( produtos.id_p = prod_compra.id_p
AND prod_compra.n_compra = compras.n_compra
AND compras.id_forn = fornecedores.id_forn )
WHERE fornecedores.nome_forn = 'continente';
```

PERGUNTA 2

Linguagem corrente:

Qual a media do valor total de todas as compras?

Álgebra relacional:

$\mathcal{G}_{avg(total)}$(compras)

SQL:

```
SELECT avg( total )
FROM compras;
```

PERGUNTA 3

Linguagem corrente:

Quais os produtos e respectivas quantidades utilizadas na massa à bolonhesa?

Álgebra relacional:

$\Pi_{\{nome\_p,\ un\_quant,\ med\_quant\}}(\sigma_{(nome\_r='massa\ a\ bolonhesa')}[produtos \blacktriangleright\blacktriangleleft fazer\_receita \blacktriangleright\blacktriangleleft receitas])$

ou

$\Pi_{\{nome\_p,\ un\_quant,\ med\_quant\}}(\Pi_{\{id\_r\}}(\sigma_{(nome\_r='massa\ a\ bolonhesa')} (receitas)) \blacktriangleright\blacktriangleleft fazer\_receita \blacktriangleright\blacktriangleleft produtos)$

SQL:

```
SELECT nome_p, fazer_receita.un_quant, fazer_receita.med_quant
FROM produtos, fazer_receita, receitas
WHERE fazer_receita.id_p = produtos.id_p
AND fazer_receita.id_r = receitas.id_r
AND receitas.nome_r = 'massa a bolonhesa';
```

ou

```
SELECT nome_p, fazer_receita.un_quant, fazer_receita.med_quant
FROM produtos
INNER JOIN fazer_receita
INNER JOIN receitas ON ( fazer_receita.id_p = produtos.id_p
AND fazer_receita.id_r = receitas.id_r )
WHERE receitas.nome_r = 'massa a bolonhesa';
```

PERGUNTA 4

Linguagem corrente:

Quais as sobremesas que levam leite condensado?

Álgebra relacional:

$\Pi_{\{receitas.id\_r,\ nome\_r\}}(\sigma_{(familia\_r='sobremesa')}(receitas) \blacktriangleright\blacktriangleleft fazer\_receitas \blacktriangleright\blacktriangleleft \sigma_{(nome\_p='leite\ condensado')}(produtos))$

ou

$\Pi_{\{receita.id\_r,\ nome\_r\}}(\sigma_{(nome\_p='leite\ condensado'\ and\ familia\_r='sobremesa')} (produtos \blacktriangleright\blacktriangleleft fazer\_receita \blacktriangleright\blacktriangleleft receitas))$

SQL:

```
SELECT receitas.id_r, nome_r
FROM produtos, fazer_receita, receitas
WHERE produtos.id_p = fazer_receita.id_p
AND fazer_receita.id_r = receitas.id_r
AND nome_p = 'leite condensado'
AND familia_r = 'sobremesa';
```

PERGUNTA 5

Linguagem corrente:

Qual o nome da receita, o respectivo código e a família da receita que tem como ingrediente leite condensado?

Álgebra relacional:

$\Pi_{\{receitas.id\_r,\ nome\_r,\ familia\_r\}}$(receitas⋈fazer_receitas⋈ $\sigma_{(nome\_p='leite\ condensado')}$(produtos))

SQL:

```
SELECT receitas.id_r, nome_r, familia_r
FROM produtos, fazer_receita, receitas
WHERE produtos.id_p = fazer_receita.id_p
AND fazer_receita.id_r = receitas.id_r
AND nome_p = 'leite condensado';
```

ou

```
SELECT receitas.id_r, nome_r
FROM produtos
INNER JOIN fazer_receita
INNER JOIN receitas ON ( produtos.id_p = fazer_receita.id_p
AND fazer_receita.id_r = receitas.id_r )
WHERE nome_p = 'leite condensado';
```

PERGUNTA 6

Linguagem corrente:

Qual o preços dos molhos de agrião comprados nos diversos fornecedores?

Álgebra relacional:

$\Pi_{\{preço\_unid\}}(\sigma_{(nome\_p='agriao')}$ (prod_compra ⋈ produtos))

ou

$\Pi_{\{preço\_unid\}}$($\sigma_{(nome\_p='agriao')}$(prod_compra) ⋈ produtos)

SQL:

```
SELECT preco_unid
FROM prod_compra, produtos
WHERE prod_compra.id_p = produtos.id_p
AND nome_p = 'agriao';
```

PERGUNTA 7

Linguagem corrente:

Listar o número de vezes que se fizeram compras em cada fornecedor.

Álgebra relacional:

nome_forn $\mathcal{G}_{count(n\_compra)}$ (compras ⋈ fornecedores)

SQL:

```
SELECT nome_forn, count( n_compra )
FROM fornecedores, compras
WHERE fornecedores.id_forn = compras.id_forn
GROUP BY nome_forn;
```

PERGUNTA 8

Linguagem corrente:

Quais os fornecedores onde se gastou mais de 100€?

Álgebra relacional:

$\Pi_{\{nome\_forn\}}(\sigma_{(total>100)}(fornecedores \bowtie compras))$

SQL:

```
SELECT nome_forn
FROM fornecedores, compras
WHERE fornecedores.id_forn = compras.id_forn
AND total >100;
```

PERGUNTA 9

Linguagem corrente:

Quantas garrafas de azeite há em stock?

Álgebra relacional:

$\Pi_{\{nome\_p,\ omedida,\ ounidade\}}(\sigma_{(nome\_p='azeite')}(stock\_outro \bowtie produtos))$

SQL:

```
SELECT nome_p, ounidade, omedida
FROM produtos, stock_outro
WHERE produtos.id_p = stock_outro.id_p
AND nome_p = 'azeite';
```

PERGUNTA 10

Linguagem corrente:

Qual o local onde estão armazenadas as batatas?

Álgebra relacional:

$\Pi_{\{nome\_p,\ localizacao\}}(\sigma_{(nome\_p='batata')}(produtos))$

SQL:

```
SELECT nome_p, localizacao
FROM produtos
WHERE nome_p = 'batata';
```

PERGUNTA 11

Linguagem corrente:

Para quantas pessoas dá a receita de natas do céu?

Álgebra relacional:

$\Pi_{\{nome\_r,\ n\_pessoa\}}(\sigma_{(nome\_r='natas\ do\ ceu')} (receitas) \bowtie cozinhado)$

SQL:

```
SELECT nome_r, n_pessoas
FROM receitas, cozinhado
WHERE receitas.id_coz = cozinhado.id_coz
AND nome_r = 'natas do ceu';
```

PERGUNTA 12

Linguagem corrente:

Em que dia se comprou óleo da marca vaqueiro?

Álgebra relacional:

$\Pi_{\{\ datac\}}(compras \bowtie \sigma_{(marca='vaqueiro'\ )} (prod\_compra) \bowtie \sigma_{(nome\_p='oleo')} (produtos))$

SQL:

```
SELECT nome_p, marca, datac
FROM produtos, prod_compra, compras
WHERE compras.n_compra = prod_compra.n_compra
AND prod_compra.id_p = produtos.id_p
AND nome_p = 'oleo'
AND marca = 'vaqueiro';
```

PERGUNTA 13

Linguagem corrente:

Quais as receitas que utilizam vegetais?

Álgebra relacional:

$\Pi_{\{nome\_r\}}$(receitas ⋈ fazer_receitas ⋈ produtos ⋈ $\sigma_{(nome\_familia='vegetal')}$(familia_p))

SQL:

```
SELECT DISTINCT nome_r
FROM receitas, fazer_receita, produtos, familia_p
WHERE receitas.id_r = fazer_receita.id_r
AND fazer_receita.id_p = produtos.id_p
AND produtos.id_familia = familia_p.id_familia
AND nome_familia = 'vegetal ';
```

PERGUNTA 14

Linguagem corrente:

Das receitas que utilizam vegetais, quais os vegetais utilizados em cada uma?

Álgebra relacional:

$\Pi_{\{nome\_r\ ,\ nome\_p\}}$($\sigma_{(nome\_familia='vegetal')}$(familia_p) ⋈ receitas⋈fazer_receita⋈produtos)

SQL:

```
SELECT nome_r, nome_p
FROM receitas, fazer_receita, produtos, familia_p
WHERE receitas.id_r = fazer_receita.id_r AND fazer_receita.id_p = produtos.id_p
AND produtos.id_familia = familia_p.id_familia  AND nome_familia = 'vegetal';
```

PERGUNTA 15

Linguagem corrente:

Qual o valor total gasto em todas as compras efectuadas no fresco e companhia?

Álgebra relacional:

$\mathcal{G}$ sum(total) as 'total de compras no fresco e companhia' ($\sigma_{(nome\_forn='fresco\ e\ companhia')}$ (compras⋈fornecedores))

SQL:

```
SELECT sum( total ) AS 'total de compra no frescos e companhia'
FROM compras INNER JOIN fornecedores ON compras.id_forn = fornecedores.id_forn
WHERE nome_forn = 'frescos e companhia'
```

PERGUNTA 16

Linguagem corrente:

Quais as marcas de óleo que já compramos?

Álgebra relacional:

$\Pi_{\{marca\}}(\sigma_{(nome\_p='oleo')} (prod\_compra \triangleright\triangleleft produtos))$

SQL:

```
SELECT marca
FROM produtos, prod_compra
WHERE produtos.id_p = prod_compra.id_p
AND nome_p = 'oleo';
```

PERGUNTA 17

Linguagem corrente:

Quais os cozinhados que tem como ingredientes batata ou cenoura?

Álgebra relacional:

$\Pi_{\{nome\_r\}}(\sigma_{(nome\_p='batata' \text{ v } nome\_p='cenoura')} (receitas \triangleright\triangleleft fazer\_receita \triangleright\triangleleft produtos))$

SQL:

```
SELECT DISTINCT nome_r
FROM receitas, fazer_receita, produtos
WHERE receitas.id_r = fazer_receita.id_r
AND fazer_receita.id_p = produtos.id_p
AND (nome_p = 'batata' OR nome_p = 'cenoura');
```

PERGUNTA 18

Linguagem corrente:

Listar os nomes dos produtos comprados no jumbo.

Álgebra relacional:

$\Pi_{\{nome\_p\}}(\sigma_{(nome\_forn='jumbo')} (produtos \triangleright\triangleleft fazer\_compras \triangleright\triangleleft compras \triangleright\triangleleft fornecedores))$

SQL:

```
SELECT nome_p
FROM produtos, prod_compra, compras, fornecedores
WHERE prod_compra.n_compra =compras.n_compra AND produtos.id_p = prod_compra.id_p
AND compras.id_forn = fornecedores.id_forn AND  nome_forn = 'jumbo';
```

PERGUNTA 19

Linguagem corrente:

Quantos produtos estão associados a cada família dos produtos?

Álgebra relacional:

$\Pi_{\{nome\_familia,\ nome\_familia\ g\ count(nome\_p)(produtos)\ \}}$(produtos ⋈ familia_p)

SQL:

```
SELECT nome_familia, count( nome_p )
FROM produtos
INNER JOIN familia_p ON ( produtos.id_familia = familia_p.id_familia )
GROUP BY (nome_familia);
```

PERGUNTA 20

Linguagem corrente:

Qual o produto mais caro comprado na makro?

Álgebra relacional:

$r \leftarrow \Pi_{\{nome\_p,\ preco\_unid\ \}}$(produtos ⋈ prod_compra⋈ $\sigma_{(nome\_forn='makro')}$(fornecedores))

$\Pi_{\{nome\_p,\ preco\_unid\ \}}$(produtos ⋈ prod_compra⋈ $\sigma_{(nome\_forn='makro')}$(fornecedores)) -
$\Pi_{\{A.nome\_p,\ A.preco\_unid\ \}}$($\sigma_{(B.preco\_unid>A.preco\_unid)}$($\rho_A(r) \times \rho_B(r)$)
)

SQL:

```
SELECT DISTINCT nome_p, C.preco_unid
FROM produtos, prod_compra AS C, compras, fornecedores
WHERE produtos.id_p = C.id_p
AND C.n_compra = compras.n_compra
AND compras.id_forn = fornecedores.id_forn
AND nome_forn = 'makro'
AND C.preco_unid NOT IN
(
   SELECT DISTINCT A.preco_unid
   FROM produtos, prod_compra AS A, prod_compra AS B, compras, fornecedores
   WHERE produtos.id_p = A.id_p
   AND A.n_compra = compras.n_compra
   AND B.n_compra = compras.n_compra
   AND compras.id_forn = fornecedores.id_forn
   AND nome_forn = 'makro'
   AND B.preco_unid > A.preco_unid
)
```

PERGUNTA 21

Linguagem corrente:

Quais os cozinhados que demoram mais tempo a preparar e que têm o ingrediente azeite?

Álgebra relacional:

$r \leftarrow \Pi_{\{nome\_r\}}(cozinhado \bowtie receitas \bowtie fazer\_receita \bowtie \sigma_{(nome\_p='azeite')}(produtos))$

$\Pi_{\{nome\_r\}}(cozinhado \bowtie receitas \bowtie fazer\_receita \bowtie \sigma_{(nome\_p='azeite')}(produtos)) - \Pi_{\{C2.nome\_p\}}(\sigma_{(C2.duracao < C1.duracao)}(\rho_{C1}(r) \times \rho_{C2}(r)))$

SQL:

```
SELECT nome_r
FROM fazer_receita
INNER JOIN receitas
INNER JOIN cozinhado
INNER JOIN produtos ON ( fazer_receita.id_r = receitas.id_r
AND receitas.id_coz = cozinhado.id_coz
AND fazer_receita.id_p = produtos.id_p )
WHERE nome_p = 'azeite'
AND nome_r NOT IN
(
   SELECT DISTINCT B.nome_r
   FROM
     (
     SELECT receitas.id_r, nome_r, duracao
     FROM fazer_receita, receitas, cozinhado, produtos
     WHERE fazer_receita.id_r = receitas.id_r
     AND receitas.id_coz = cozinhado.id_coz
     AND fazer_receita.id_p = produtos.id_p
     AND nome_p = 'azeite'
     ) AS A,
     (
     SELECT receitas.id_r, nome_r, duracao
     FROM fazer_receita, receitas, cozinhado, produtos
     WHERE fazer_receita.id_r = receitas.id_r
     AND receitas.id_coz = cozinhado.id_coz
     AND fazer_receita.id_p = produtos.id_p
     AND nome_p = 'azeite'
      ) AS B
  WHERE B.duracao < A.duracao
 );
```

PERGUNTA 22

Linguagem corrente:

Discrimine os ingredientes usados na canja, incluindo o código do produto, a medida e a quantidade.

Álgebra relacional:

$\Pi_{\{fazer\_receitas.id\_p,\ nome\_p,\ med\_quant,\ un\_quant\}}(fazer\_receita \bowtie \sigma_{(nome\_r='canja')}(receitas) \bowtie produtos)$

SQL:

```
SELECT fazer_receita.id_p, nome_p, med_quant, un_quant
FROM receitas, fazer_receita, produtos
WHERE receitas.id_r = fazer_receita.id_r
AND produtos.id_p = fazer_receita.id_p
AND nome_r = 'canja';
```

PERGUNTA 23

Linguagem corrente:

Conte o número de produtos que existe em cada local (despensa, frigorifico, etc.)

Álgebra relacional:

$\Pi_{\{localizacao,\ count(nome\_p)\}}(localizacao\mathcal{G}\ count(nome\_p) (produtos))$

SQL:

```
SELECT localizacao, count( nome_p )
FROM produtos
GROUP BY ( localizacao );
```

PERGUNTA 24

Linguagem corrente:

Quais as receitas que não usam o ingrediente sal?

Álgebra relacional:

$\Pi_{\{nome\_r\}}(receitas) - \Pi_{\{nome\_r\}}(receitas \triangleright\triangleleft fazer\_receita \triangleright\triangleleft \sigma_{(nome\_p='sal')}(produtos))$

SQL:

```
SELECT nome_r
FROM receitas
WHERE nome_r NOT IN
 (
    SELECT nome_r
    FROM receitas
    INNER JOIN fazer_receita
    INNER JOIN produtos ON ( receitas.id_r = fazer_receita.id_r
    AND fazer_receita.id_p = produtos.id_p )
    WHERE nome_p = 'sal'
);
```

ou

```
SELECT R.nome_r
FROM receitas AS R
WHERE NOT EXISTS
(

  SELECT nome_r
  FROM receitas
  INNER JOIN fazer_receita
  INNER JOIN produtos ON ( receitas.id_r = fazer_receita.id_r
  AND fazer_receita.id_p = produtos.id_p )
  WHERE nome_p = 'sal'
  AND R.id_r = receitas.id_r
 )
```

PERGUNTA 25

Linguagem corrente:

Dos ingredientes da sopa de legumes, qual é o produto mais barato?

Álgebra relacional:

$r \leftarrow \Pi_{\{nome\_p,\ preco\_unid\ \}}(\ prod\_compra \rhd\lhd produtos \rhd\lhd fazer\_receita \rhd\lhd \sigma_{\ (nome\_r='sopa\ de\ legumes')}(receitas))$

$\Pi_{\{nome\_p,\ preco\_unid\ \}}(\ prod\_compra \rhd\lhd produtos \rhd\lhd fazer\_receita \rhd\lhd \sigma_{\ (nome\_r='sopa\ de\ legumes')}(receitas))\ -$
$\Pi_{\ \{A2.nome\_p\ \}}(\ \sigma_{(A1.preco\_unid<A2.preco\_unid)}(\ \rho_{A1}(r) \times \rho_{A2}(r)\ )$

SQL:

```
SELECT nome_p, preco_unid
FROM receitas, fazer_receita, produtos, prod_compra
WHERE receitas.id_r = fazer_receita.id_r
AND fazer_receita.id_p = produtos.id_p
AND produtos.id_p = prod_compra.id_p
AND nome_r = 'sopa de legumes'
AND nome_p NOT IN
(
   SELECT B.nome_p
   FROM
      (
      SELECT nome_p, preco_unid
      FROM fazer_receita, receitas, produtos, prod_compra
      WHERE fazer_receita.id_r = receitas.id_r
      AND produtos.id_p = prod_compra.id_p
      AND fazer_receita.id_p = produtos.id_p
      AND nome_r = 'sopa de legumes'
      ) AS A, (
      SELECT nome_p, preco_unid
      FROM fazer_receita, receitas, produtos, prod_compra
      WHERE fazer_receita.id_r = receitas.id_r
      AND produtos.id_p = prod_compra.id_p
      AND fazer_receita.id_p = produtos.id_p
      AND nome_r = 'sopa de legumes'
      ) AS B
   WHERE B.preco_unid > A.preco_unid
);
```

PERGUNTA 26

Linguagem corrente:

Listar os números das compras de todos os fornecedores que são hipermercados.

Álgebra relacional:

$\Pi_{\{n\_compra,\ id\_forn,\ descricao\_forn\}}(compras \blacktriangleright\blacktriangleleft fornecedores) \div \Pi_{\{id\_forn,\ descricao\_forn\}}(\sigma_{(descricao\_forn='hipermercados')}(fornecedores))$

SQL:

```
SELECT A.n_compra
FROM
   (
    SELECT n_compra, compras.id_forn, descricao_forn
    FROM compras, fornecedores
    WHERE compras.id_forn = fornecedores.id_forn
    AND compras.id_forn  IN
       (
       SELECT id_forn
       FROM fornecedores
       WHERE descricao_forn = 'hipermercado'
       )
    ) AS A;
```

ou

```
SELECT A.n_compra
FROM
(
    SELECT n_compra, C.id_forn, descricao_forn
    FROM compras AS C, fornecedores
    WHERE C.id_forn = fornecedores.id_forn
    AND EXISTS
    (

      SELECT id_forn
      FROM fornecedores
      WHERE descricao_forn = 'hipermercado'
      AND C.id_forn = fornecedores.id_forn
  )
) AS A ;
```

PERGUNTA 27

Linguagem corrente:

Qual o nome dos produtos que estão todos no congelador1 ou no congelador2? (Em SQL ordene por ordem crescente e decrescente o nome dos produtos)

Álgebra relacional:

$\Pi_{\{nome\_p,\ localizacao\}}(\ produtos\ ) \div \Pi_{\{localizacao\}}(\ \sigma_{(localizacao='congelado1' \vee localizacao='congelador2)}(produtos))$

SQL:

→ *Ordem decrescente*

```
SELECT A.nome_p
FROM
   (
   SELECT nome_p, localizacao
   FROM produtos
   WHERE localizacao IN
      (
      SELECT localizacao
      FROM produtos
      WHERE localizacao = 'congelador1'
      OR localizacao = 'congelador2'
      )
   ) AS A
ORDER BY ( A.nome_p ) DESC;
```

→ *Ordem Crescente*

```
SELECT A.nome_p
FROM
   (
   SELECT nome_p, localizacao
   FROM produtos
   WHERE localizacao IN
      (
      SELECT localizacao
      FROM produtos
      WHERE localizacao = 'congelador1'
      OR localizacao = 'congelador2'
      )
   ) AS A
ORDER BY ( A.nome_p) ASC ;
```

NOTA: No caso de ordenar por ordem crescente pôr *asc* ou não é igual, pois por defeito o *order by* ordena por ordem crescente.

PERGUNTA 28

Linguagem corrente:

Listar as localizações cujo número de produtos, por localização, seja maior ou igual a 5.

Álgebra relacional:

$\Pi_{\{localizacao,\ total\_produtos\}}(\sigma_{(total\_produtos>=5)}\ (_{localizacao}\mathcal{G}\ count(nome\_p)\ as\ total\_produtos\ (produtos)\ )$

SQL:

```
SELECT localizacao, count( nome_p ) AS total_produtos
FROM produtos
GROUP BY (localizacao)
HAVING count( nome_p ) >5;
```

PERGUNTA 29

Linguagem corrente:

Listar o nome dos produtos que estão na prateleira1 ou na prateleira2 e cuja medida é pacote.

Álgebra relacional:

$\Pi_{\{nome\_p\}}(\ \sigma\ (\ (localizacao='prateleira1' \vee localizacao='prateleira2')\ \wedge omedida='pacote')(produtos))$

SQL:

```
SELECT nome_p
FROM produtos, stock_outro
WHERE produtos.id_p = stock_outro.id_p
AND ( localizacao = 'prateleira1'
OR localizacao = 'prateleira2')
AND omedida = 'pacote';
```

PERGUNTA 30

Linguagem corrente:

Liste o nome das receitas que tenham a cadeia “sop” no nome da receita.

SQL:

```
SELECT nome_r
FROM receitas
WHERE nome_r LIKE '%sop%';
```

PERGUNTA 31

Linguagem corrente:

Listar a união de stock_outro e stock_peso (em SQL ordenar por código de produto).

Álgebra relacional:

$r \leftarrow \Pi_{\{stock\_outro.id\_p,\ omedida\ as\ medida,\ ounidade\ as\ unidade\}}(stock\_outro)$
$s \leftarrow \Pi_{\{stock\_peso.id\_p,\ pmedida\ as\ medida,\ punidade\ as\ unidade\}}(stock\_peso)$
$t \leftarrow s \cup r$

$\Pi_{\{stock.id\_p,\ stock.medida,\ stock.unidade\}}(\rho_{stock}(t))$

SQL:

```
SELECT  stock.id_p, stock.medida, stock.unidade
FROM (
    SELECT id_p, omedida AS medida, ounidade AS unidade
    FROM stock_outro
    UNION
    SELECT id_p, pmedida AS medida, punidade AS unidade
    FROM stock_peso
 ) AS stock
 ORDER BY stock.id_p;
```

PERGUNTA 32

Linguagem corrente:

Listar o nome dos produtos e as respectivas unidades e medidas que têm registos no stock_outro e no stock_peso.

Álgebra relacional:

$\Pi_{\{produtos.id\_p,\ nome\_p,\ omedida,\ ounidade\}}(stock\_outro \blacktriangleright\blacktriangleleft produtos) \cap$
$\Pi_{\{produtos.id\_p,\ nome\_p,\ pmedida,\ punidade\}}(stock\_peso \blacktriangleright\blacktriangleleft produtos)$

SQL:

```
SELECT produtos.id_p, nome_p, omedida, ounidade, pmedida, punidade
FROM stock_outro, stock_peso, produtos
WHERE stock_outro.id_p = stock_peso.id_p
AND produtos.id_p = stock_peso.id_p;
```

PERGUNTA 33

Linguagem corrente:

Quais os produtos que tem um custo superior a 1€ e inferior a 5€?

Álgebra relacional:

$\Pi_{\{ nome\_p \}}(\sigma_{(preço\_unid<5 \ \wedge \ preco\_unid>1)} (produtos \blacktriangleright\blacktriangleleft prod\_compra))$

SQL:

```
SELECT nome_p
FROM produtos
INNER JOIN prod_compra
USING ( id_p )
WHERE preco_unid BETWEEN 1 AND 5;
```

PERGUNTA 34

34.1 Linguagem corrente:

Quais os produtos que não tem marca do produto definida?

Álgebra relacional:

$\Pi_{\{ nome\_p \}}(\sigma_{(marca =null)} (produtos \blacktriangleright\blacktriangleleft prod\_compra))$

SQL:

```
SELECT nome_p
FROM produtos
INNER JOIN prod_compra
USING ( id_p )
WHERE marca IS NULL;
```

34.2 Linguagem corrente:

Quais os produtos que tem marca do produto definida?

Álgebra relacional:

$\Pi_{\{ nome\_p \}}(\sigma_{(marca <> null)} (produtos \blacktriangleright\blacktriangleleft prod\_compra))$

SQL:

```
SELECT nome_p
FROM produtos
INNER JOIN prod_compra
USING ( id_p )
WHERE marca IS NOT NULL;
```

PERGUNTA 35

35.1 Linguagem corrente:

Considerando os produtos associados ao stock_outro, liste os respectivos códigos de produtos, quantidades e medidas dos produtos em stock.

Álgebra relacional:

$$\Pi_{\{\text{ stock\_outro.id\_p, ounidade, omedida, stock\_peso.id\_p, punidade, pmedida}\}}(\text{ stock\_outro } \_\blacktriangleright\blacktriangleleft \text{ stock\_peso })$$

SQL:

```
SELECT stock_outro.id_p, ounidade, omedida, stock_peso.id_p, punidade, pmedida
 FROM stock_outro
 LEFT JOIN stock_peso ON ( stock_outro.id_p = stock_peso.id_p );
```

35.2 Linguagem corrente:

Considerando os produtos associados ao stock_peso, liste os respectivos códigos de produtos, quantidades e medidas dos produtos em stock.

Álgebra relacional:

$$\Pi_{\{\text{ stock\_outro.is\_p, ounidade, omedida, stock\_peso.id\_p, punidade, pmedida}\}}(\text{ stock\_outro } \blacktriangleright\blacktriangleleft\_ \text{ stock\_peso })$$

SQL:

```
SELECT stock_outro.id_p, ounidade, omedida, stock_peso.id_p, punidade, pmedida
 FROM stock_outro
 RIGHT JOIN stock_peso ON ( stock_outro.id_p = stock_peso.id_p );
```

PERGUNTA 36

Linguagem corrente:

Quais os produtos que estão na prateleira1 ou prateleira2 ou prateleira3?

Álgebra Relacional:

$\Pi_{\{nome\_p,\ localizacao\}}(\sigma_{(localizacao='prateleira1' \vee localizacao='prateleira2' \vee localizacao='prateleira3')}(produtos))$

SQL:

→ ***EXISTS***

```
SELECT nome_p, localizacao
FROM produtos AS P
WHERE EXISTS
(
   SELECT localizacao
   FROM produtos
   WHERE
   (
   localizacao = 'prateleira1'
   OR localizacao = 'prateleira2'
   OR localizacao = 'prateleira3'
   )
   AND produtos.id_p = P.id_p
);
```

→ ***SOME***

```
SELECT nome_p, localizacao
FROM produtos AS P
WHERE localizacao = SOME
(
    SELECT localizacao
    FROM produtos
    WHERE ( localizacao = 'prateleira1'
    OR localizacao = 'prateleira2'
    OR localizacao = 'prateleira3'
    )
);
```

→ ***IN***

```
SELECT nome_p, localizacao
FROM produtos AS P
WHERE localizacao IN
(   SELECT localizacao
   FROM produtos
   WHERE ( localizacao = 'prateleira1'
   OR localizacao = 'prateleira2'
   OR localizacao = 'prateleira3'));
```

PERGUNTA 37

Linguagem corrente:

Quais os produtos que não estão na prateleira1ou prateleira2 ou prateleira3

Álgebra Relacional:

$\Pi_{\{nome\_p,\ localizacao\}}(\sigma_{(\neg(localizacao='prateleira1' \vee localizacao='prateleira2' \vee localizacao='prateleira3'))}(produtos))$

SQL:

→ ***NOT IN***

```
SELECT nome_p, localizacao
FROM produtos AS P
WHERE localizacao NOT IN
(
    SELECT localizacao
    FROM produtos
    WHERE (
    localizacao = 'prateleira1'
    OR localizacao = 'prateleira2'
    OR localizacao = 'prateleira3'
    )
);
```

→***ALL***

```
SELECT nome_p, localizacao
FROM produtos AS P
WHERE localizacao <> ALL
(
    SELECT localizacao
    FROM produtos
    WHERE (
    localizacao = 'prateleira1'
    OR localizacao = 'prateleira2'
    OR localizacao = 'prateleira3'
    )
);
```

→ ***NOT EXISTS***

```
SELECT nome_p, localizacao
FROM produtos AS P
WHERE NOT EXISTS
 (
    SELECT localizacao
    FROM produtos
    WHERE (
    localizacao = 'prateleira1'
    OR localizacao = 'prateleira2'
    OR localizacao = 'prateleira3'
    )
    AND produtos.id_p = P.id_p
);
```

Capítulo 11. Conclusão

- Um assunto que não abordámos no nosso projecto foi ***vistas*** pois, a sua utilização não tem qualquer cabimento na nossa base de dados.

- É importante realçar que a versão de MySQL usada não suporta a operação INTERSECT.

- Apesar de termos pensado em colocar o modo de preparação dos cozinhados na nossa base de dados, achámos que não faria muito sentido visto que, a base de dados que criámos não serve a quem prepara os cozinhados mas sim à pessoa que governa a cozinha. Partimos então do princípio que o modo de preparação das receitas que existem na nossa base de dados está armazenado em livros de receitas para então os cozinheiros terem acesso a elas.

Capítulo 12. Bibliografia

- Luís Damas: SQL, FCA, [6]2005
- www.mysql.com
- Bases de Dados, Reinhard Kahle, Departamento de Matemática Universidade de Coimbra, 2005/2006

PROJECTO REALIZADO POR:

(Rita Margarida Ferreira Coimbra)

(Sara Joana Fino dos Santos Rodrigues de Carvalho)

(Sara Margarida Gaspar da Silva)