

# Matemática e Informática

## BASES DE DADOS RELACIONAIS

INSTITUTO SUPERIOR DE AGRONOMIA

DEPARTAMENTO DE MATEMÁTICA

Nestes apontamentos expõe-se a matéria que constitui a parte final do 4º Módulo da Unidade Curricular de *Matemática e Informática*, do 1º ano, do 1º ciclo de todos os cursos adaptados a Bolonha, do Instituto Superior de Agronomia.

O texto foi adaptado de “Apontamentos de Computadores e Sistemas de Informação” por Graça Abrantes.

Fernanda Valente

Marta Mesquita

Graça Abrantes

ISA, Abril de 2010

# Capítulo 1

## Bases de Dados Relacionais

Nas diversas áreas de conhecimento surge frequentemente a necessidade de recolher e processar uma grande quantidade de **dados** que se encontram interrelacionados. Nestes casos é necessário gerir esses dados de um modo eficaz, de tal forma que **não existam perdas** de informação, que a organização dos dados permita a sua **fácil compreensão** e que o seu processamento por meios informáticos seja possível com **eficiência**.

Durante a década de 70 do século XX eram já muitas as deficiências sentidas em diversas áreas como resultado da organização dos dados em simples ficheiros. Contudo, a tecnologia informática da época apenas permitiu que grandes organizações (como os Bancos, por exemplo) recorressem a um software específico, denominado **Sistema Gestor de Bases de Dados** (SGBD), bem diferente dos que existem actualmente, para suprir algumas dessas deficiências.

Nos últimos anos, o desenvolvimento da informática, quer no que respeita a *hardware* como a *software*, conjuntamente com a enorme baixa dos seus custos financeiros, permitiu que a utilização dos SGBD para criação, manutenção e processamento de Bases de Dados se banalizasse e que estes sistemas se tornassem auxiliares indispensáveis nos problemas que envolvem dados de vários tipos.

A capacidade, que muitos SGBD possuem, de permitir acessos de utilizadores em rede e de suportar dados distribuídos por vários computadores a funcionarem em rede, contribui também para a crescente utilização dos SGBD.

Uma **Base de Dados** é um conjunto de dados relacionados que é utilizado recorrendo a um Sistema Gestor de Bases de Dados (SGBD). Uma Base de Dados representa uma parte da realidade que é relevante para a resolução de um determinado conjunto de problemas.

Presentemente, existem no mercado diversos SGBD, os mais utilizados são os **SGBD relacionais** que permitem criar bases de dados relacionais. Os conceitos de Base de Dados relacional foram apresentados em 1970 e desde então vários fornecedores de *software* têm vindo a desenvolver SGBD relacionais.

O **OpenOffice.org Base** ([www.openoffice.org](http://www.openoffice.org)) inclui-se nesta categoria de SGBD e destina-se a ser utilizado em computadores pessoais com sistema operativo Windows ou Linux. Microsoft Access, MySQL e ORACLE constituem exemplos de outros SGBD relacionais.

### 1.1. Modelo conceptual de tipo Entidade-Associação

Para que seja possível extrair informação de qualquer conjunto de dados é necessário conhecer as regras que foram utilizadas na sua organização. O conjunto de regras utilizadas na criação de uma base de dados constitui o **modelo conceptual dos dados**.

A criação deste modelo é frequentemente uma tarefa difícil pelo que a procura de metodologias, métodos e técnicas para a sua construção é presentemente uma área de investigação no campo da Engenharia de Software.

Os métodos de desenvolvimento de bases de dados mais divulgados incluem uma fase de criação de um modelo conceptual de dados e, posteriormente, uma fase de conversão deste modelo para o SGBD escolhido para a sua implementação.

Em muitos dos modelos conceptuais, criados durante o desenvolvimento de bases de dados relacionais, têm sido utilizados com sucesso conceitos e regras de tipo **Entidade-Associação** (E-A). Uma das qualidades dos modelos criados desta forma reside no facto de poderem ser representados por diagramas recorrendo a uma notação muito simples.

Os modelos de tipo E-A são compostos por **entidades** e **associações**. As entidades representam objectos reais que possuem uma descrição que é determinada pelos problemas que se pretendem resolver por meio da Base de Dados. As associações representam relacionamentos relevantes entre entidades.

As entidades que possuem uma descrição comum são representadas por um **tipo de entidade**, o qual é caracterizado por um conjunto de **atributos**. Os **valores** que tomam cada um dos atributos de um tipo de entidade permitem descrever e distinguir entre si as entidades que pertencem a um mesmo tipo de entidade.

Exemplo 1:

Pretende-se criar uma base de dados para efectuar a gestão de uma empresa agrícola, onde seja registada informação relativa aos seus trabalhadores e à produção e venda. Sobre cada trabalhador pretende-se registar o seu nome, data de nascimento e especialidade. Sobre cada produto da exploração agrícola pretende-se conhecer a sua designação, área cultivada e qual o trabalhador responsável. Em relação aos compradores e aos fornecedores, pretende-se conhecer a sua identificação, morada e telefone. Para cada matéria-prima pretende-se registar a sua designação e quantidade armazenada. Para cada produto vendido deverá ser registado o comprador, a data da venda, a quantidade vendida, o valor da venda e a data do recebimento. Para a matéria-prima consumida pretende-se registar os produtos a que esta se destina, a data da utilização e a quantidade utilizada. Para cada aquisição de matéria-prima deve-se conhecer o fornecedor, a data da compra, a quantidade comprada, o valor da compra e a data de pagamento.

Neste exemplo, os trabalhadores da empresa, os produtos cultivados, as matérias primas utilizadas, os compradores e os fornecedores são exemplos de tipos de entidade do modelo de dados. Cada um destes tipos de entidade é identificado por um nome: TRABALHADOR, PRODUTO, MATÉRIA\_PRIMA, COMPRADOR, FORNECEDOR.

O nome do trabalhador, a especialidade e a data do nascimento são exemplos de atributos do tipo de entidade TRABALHADOR; a designação do produto cultivado e a área de cultura são exemplos de atributos do tipo de entidade PRODUTO.

A um atributo de um tipo de entidade cujos valores permitam identificar de forma inequívoca cada uma das entidades desse tipo dá-se o nome de **chave primária**. Cada tipo de entidade deve possuir uma chave primária.

No exemplo atrás citado pode ser criado um número identificativo para cada trabalhador de forma a constituir uma chave primária do tipo de entidade TRABALHADOR e pode ser utilizado um código como chave primária do tipo de entidade PRODUTO. Para cada um dos restantes tipos de entidade deve também ser criado um código que irá ser a respectiva chave primária.

Um **tipo de associação** representa as associações com características comuns que se estabelecem entre as várias entidades. Um tipo de associação pode possuir atributos que permitam descrever as características próprias de cada associação.

No exemplo considerado pode ser criado um tipo de associação, com o nome UTILIZA, que representa as associações que se estabelecem entre as entidades do tipo PRODUTO e as entidades do tipo MATÉRIA\_PRIMA. Como atributos deste tipo de associação podem incluir-se a data em que a matéria-prima foi utilizada e a quantidade usada. Pode ainda ser criado um tipo de associação VENDA para representar as associações existentes entre as entidades do tipo PRODUTO e as entidades do tipo COMPRADOR e outro tipo de associação COMPRA para representar as associações existentes entre MATÉRIA\_PRIMA e FORNECEDOR.

Um tipo de associação caracteriza-se pelo número de tipos de entidade que envolve, podendo ser unária, ou binária (o caso mais frequente), ou ternária, ou ...

A **multiplicidade** do tipo de associação indica o modo como podem ser estabelecidas as associações. Um tipo de **associação binária** diz-se:

- de **um para um** (1:1) quando cada entidade só pode participar numa única associação desse tipo;
- de **um para muitos** (1:n) quando cada entidade de um tipo só pode participar numa associação, mas as entidades do outro tipo podem participar em mais do que uma associação desse tipo;
- de **muitos para muitos** (n:n) quando não existe nenhuma restrição ao número de vezes que cada entidade pode participar em associações desse tipo.

Assim, por exemplo, pode dizer-se que o tipo de associação RESPONSÁVEL existente entre os tipos de entidade TRABALHADOR e PRODUTO é de 1:n, o que significa que cada trabalhador pode ser responsável por um ou mais produtos mas cada produto é da responsabilidade de um único trabalhador. O tipo de associação VENDA ser de n:n significa que cada produto pode ter diversos compradores e que um mesmo comprador pode adquirir diversos produtos.

Geralmente, é vantajoso definir a multiplicidade de um tipo de associação de uma forma mais detalhada, indicando dois números: o mínimo e o máximo de vezes que uma entidade pode participar em associações de um mesmo tipo. A multiplicidade é frequentemente 0,1 ou 1,1 ou 0,n ou 1,n, mas pode ser diferente (2,2 ou 2,n, por exemplo).

O tipo de associação RESPONSÁVEL tem multiplicidade 0,n relativamente ao tipo de entidade TRABALHADOR, o que significa que podem existir trabalhadores que não são responsáveis por nenhum produto e outros que são responsáveis por mais do que um produto.

A multiplicidade do tipo de associação RESPONSÁVEL é 1,1 relativamente ao tipo de entidade PRODUTO, o que significa que para cada produto existe sempre um e só um trabalhador responsável.

As multiplicidades dos tipos de associação de um modelo representam **restrições de integridade** dos dados, isto é, condições que têm que ser verificadas para que a base de dados seja fiel à realidade. No entanto, nem todas as restrições de integridade dos dados podem ser representadas no modelo pela multiplicidade dos tipos de associação; por exemplo, a soma das áreas onde são cultivadas os produtos não pode exceder a área agrícola total da empresa.

A Fig. 1 representa o modelo conceptual de dados que tem vindo a ser analisado, numa notação diagramática muito divulgada. Nesta notação os rectângulos representam tipos de entidade e os losangos representam tipos de associação. No interior encontra-se um nome identificativo do tipo. As linhas que unem um losango a dois (ou mais) rectângulos denotam as entidades que se encontram envolvidas na associação. Os atributos são representados por um nome junto do tipo a que pertencem. A chave de cada tipo de entidade é representada por um atributo sublinhado. A multiplicidade dos tipos de associação, na sua forma mais detalhada, é representada junto à linha que une um tipo de entidade a um tipo de associação.

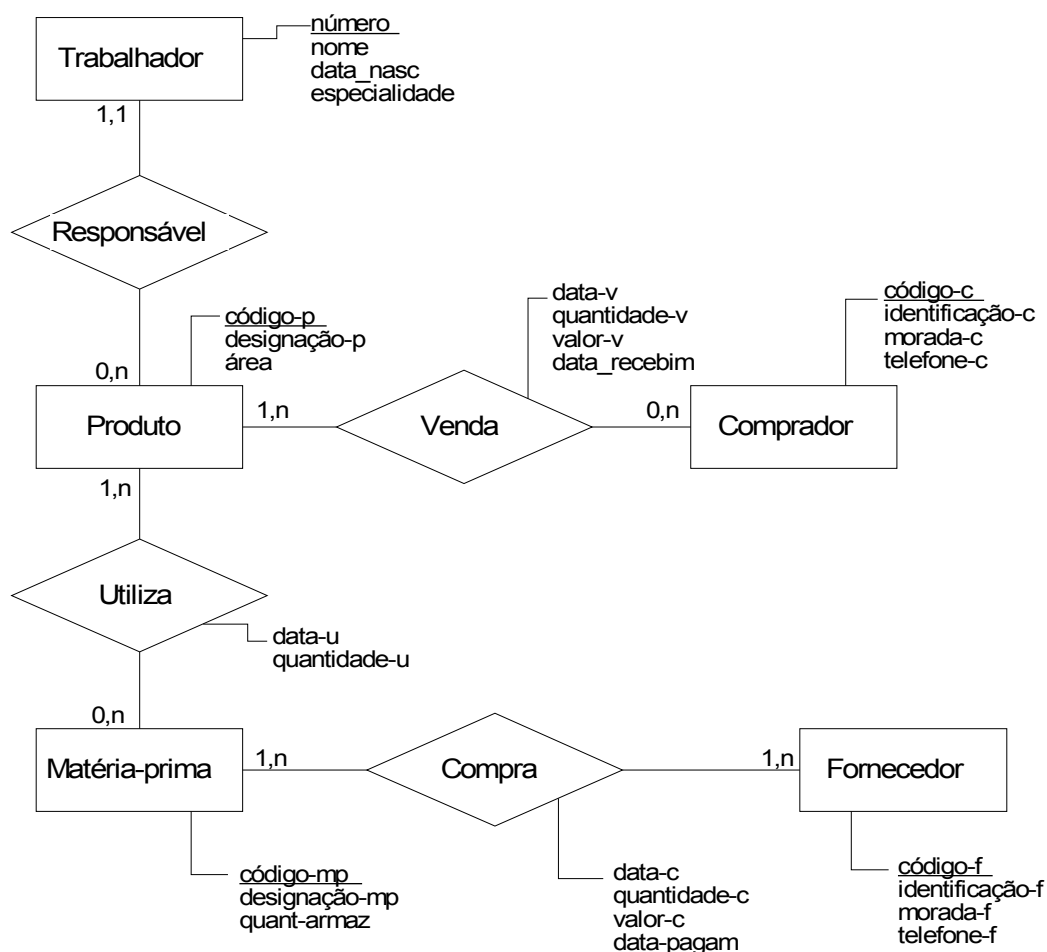


Fig. 1 - Modelo conceptual de dados do Exemplo 1 (tipo Entidade-Associação).

A primeira etapa da criação de um modelo conceptual de dados deve ser a identificação das entidades e dos atributos. Para atingir este objectivo podem seguir-se as seguintes regras gerais:

1. As entidades representam objectos que podem possuir uma descrição; os atributos não possuem informação descritiva.
2. Quando a uma entidade pode corresponder mais do que um valor de um dado atributo, este deve ser considerado como uma entidade mesmo que sobre ele não exista mais nenhuma informação descritiva. Assim, mesmo que não se pretendesse incluir qualquer informação descritiva relativamente às matérias primas utilizadas na produção, estas não poderiam ser consideradas atributos das entidades do tipo PRODUTO

e deveriam ser sempre consideradas como entidades, uma vez que em cada produto podem ser utilizadas várias matérias primas.

3. Os atributos devem ser usados nas entidades a que mais directamente dizem respeito.

4. Como atributo identificador de entidades deve evitar-se utilizar conjuntos de atributos.

Depois de identificadas as entidades e os atributos resta identificar as associações. Estas, por exclusão de partes, deverão representar os objectos que não foram considerados nem entidades nem atributos. Nesta fase devem, ainda, ser respeitadas as duas regras seguintes:

1. As associações redundantes devem ser eliminadas. Consideram-se redundantes as associações que representam um mesmo aspecto da realidade; as redundâncias mais frequentes resultam da transitividade das associações do tipo *pertence*.

2. Só devem ser usadas associações ternárias quando for impossível representá-las por meio de várias associações binárias. Quanto maior for o número de entidades envolvidas numa dada associação mais difícil será manter a integridade do sistema após as alterações a que será sujeito.

a) Uma loja pretende construir uma base de dados com informação relevante sobre os discos que vende. Relativamente a cada disco, pretende registar na base de dados um código identificativo, o título, nº de cópias vendidas, data de gravação e o grupo/intérprete. Pretende também registar as músicas que cada disco inclui. Sobre cada música, pretende registar o título que identifica a música e o primeiro autor. Uma mesma música pode figurar em discos distintos.

b) Uma empresa agrícola pretende construir uma base de dados com informação relevante sobre os animais que possui. Relativamente a cada variedade de animal, pretende registar na base de dados um nome identificativo e o número de animais dessa variedade que possui. Pretende também registar os produtos derivados de cada variedade de animal, em particular a sua designação, a quantidade produzida e o seu preço de venda. Cada um destes produtos é produzido apenas por uma variedade de animal. Cada variedade de animal consome vários alimentos e um dado alimento pode ser incluído na alimentação de diversas variedades de animal. É necessário registar os alimentos utilizados, sendo cada alimento caracterizado por um nome identificativo e pelo seu preço de aquisição. A base de dados deve ainda conter informação relativa à quantidade de cada alimento consumida por cada variedade de animal.

## **1.2. Os SGBD relacionais**

Contrariamente a outros tipos de SGBD utilizados sobretudo entre 1970 e 1985, os SGBD relacionais não surgiram na sequência do aperfeiçoamento das técnicas de processamento de ficheiros, mas sim a partir de um conjunto de conceitos teóricos apresentados em 1970

por E. F. Codd. Os conceitos que Codd introduziu foram, posteriormente, desenvolvidos por outros autores e actualmente são adoptados pelos SGBD mais frequentemente utilizados.

As vantagens dos SGBD relacionais devem-se essencialmente a três factores:

- simplicidade dos conceitos que utilizam;
- existência de definições formais para os conceitos que permitiram uma rápida divulgação e a adesão de diversos fabricantes de software;
- adequação à representação de muitos dos aspectos que constituem a realidade.

### 1.2.1. Relações, atributos e domínios

Nas bases de dados relacionais a estrutura fundamental é a **relação** ou **tabela**. Uma relação é definida por um **esquema** que é composto pelo **nome da relação** e pelo nome de um ou mais **atributos**; a definição de cada atributo depende do tipo de dados (inteiro, real, alfanumérico, data, lógico ...) que o atributo irá armazenar.

Um atributo  $A_i$  pode ser definido como uma variável que toma valores num conjunto  $D_i$  chamado **domínio** do atributo. O domínio de um atributo determina o conjunto de valores que o atributo pode tomar.

**Definição:** Dado um conjunto de atributos  $U=\{A_1, A_2, \dots, A_n\}$ , define-se uma relação  $R$  sobre  $U$  como um subconjunto do produto cartesiano  $D_1 \times D_2 \times \dots \times D_n$ . A cada tuplo  $(a_1, a_2, \dots, a_n)$  deste produto cartesiano dá-se o nome de **instância** da relação  $R$ .

O conjunto das instâncias de uma relação constitui uma **tabela** em que as linhas são as instâncias (ou **registos**) e as colunas são os atributos (ou **campos**).

Na Fig. 2 encontra-se representado o exemplo de uma relação ou tabela.

Atributos ou campos		
códigoP	designação	área
102	batata	2.5
101	cenoura	1.0
103	feijão verde	

← Instância ou registo

Fig. 2 - Exemplo de uma relação ou tabela.

É importante notar desde já que:

- numa instância o valor de um atributo é sempre atómico; isto é, numa tabela, no cruzamento de uma linha com uma coluna só pode existir um valor de atributo;
- numa relação não podem existir instâncias iguais;

- a ordem porque se encontram as instâncias de uma relação e os seus atributos é irrelevante;
- os valores de cada atributo pertencem sempre a um mesmo domínio;
- podem existir instâncias sem valores em alguns dos seus atributos; neste caso o atributo diz-se opcional e o seu valor é *null*;
- os nomes (ou identificadores) dos atributos que constituem o esquema de uma relação são únicos nessa relação.

### 1.2.2. Atributos chave

Quando um atributo de uma dada relação toma valores diferentes para cada instância dessa relação diz-se que esse atributo é uma **chave candidata** da relação. Deste modo, cada instância pode ser identificada pelo valor de um atributo chave porque dado um determinado valor dessa chave só existe uma instância em que o atributo possui esse valor. Por vezes, uma chave pode ser composta por mais do que um atributo. E, no caso extremo, uma chave pode até ser composta por todos os atributos que definem a relação.

Considera-se **chave primária** de uma relação um subconjunto mínimo de atributos cujos valores permitam identificar de modo único cada uma das instâncias dessa relação.

Nos SGBD relacionais, para representar as associações existentes entre as várias entidades utilizam-se esquemas de relações em que figuram atributos comuns.

Uma **chave estrangeira** de uma relação é um conjunto de atributos que é chave primária de outra relação.

### 1.2.3. Restrições de integridade

Para que uma base de dados represente correctamente uma determinada realidade não é suficiente indicar o conjunto de atributos que constitui o esquema das relações. É ainda necessário indicar as condições que devem ser verificadas para que um elemento do produto cartesiano dos domínios dos atributos possa pertencer à relação. Estas condições são geralmente chamadas **restrições de integridade**.

Algumas destas restrições são consequência imediata das definições anteriores. Encontram-se nesta categoria as restrições:

- de **integridade de domínio** impostas pelas definições em compreensão ou em extensão dos domínios dos atributos;

- de **integridade de entidade** que impede a existência de duas instâncias de uma relação cujos atributos chave tenham o mesmo valor e que o valor de uma chave primária (ou mesmo de uma parte dela) seja *null*;
- de **integridade de referência** que obriga a que o valor de qualquer chave estrangeira de uma relação seja um valor da chave primária da relação que refere.

Uma outra categoria de restrições importante estabelece as condições que devem ser verificadas pelos valores de dois ou mais atributos de uma mesma instância da relação. Dentro desta última categoria de restrições existem vários tipos. Algumas restrições requerem a utilização de fórmulas matemáticas ou lógicas. Mas, as mais importantes, e que se encontram em quase todos os casos práticos, são as designadas por **dependências funcionais**.

Dada um relação R definida sobre um conjunto de atributos  $U=\{A_1, A_2, \dots, A_n\}$ , diz-se que o atributo  $A_j$  **depende funcionalmente do atributo**  $A_i$  ( $A_i \rightarrow A_j$ ) quando é impossível terem-se duas instâncias da relação com o mesmo valor para  $A_i$  e diferentes valores para  $A_j$ .

#### 1.2.4. Esquema

Chama-se **esquema** de uma base de dados relacional ao conjunto das relações que a compõem, definidas por meio dos seus atributos, e das restrições de integridade.

Para cada base de dados pode existir um número quase ilimitado de esquemas. Isto não significa que todos os esquemas sejam igualmente bons.

Pode-se dizer que um esquema é tanto melhor quanto mais fácil for o seu manuseamento, nomeadamente no que se refere à manutenção da sua coerência após as actualizações que venha a sofrer. De um modo mais geral, pode dizer-se que um esquema é tanto melhor quanto menor for o número de dependências funcionais nele existente.

Na notação utilizada na representação do esquema relacional, um duplo sublinhado indica que o conjunto de atributos é a chave primária da relação e um sublinhado simples tracejado indica uma chave estrangeira.

#### 1.2.5. Normalização

Os conceitos rigorosos utilizados pelos SGBD relacionais permitiram estabelecer algumas condições que um esquema deve observar. O objectivo das regras de normalização é

definir regras para o agrupamento de atributos de forma a minimizar o número de dependências funcionais existentes numa base de dados.

### **Primeira forma normal**

Uma relação R encontra-se na primeira forma normal se e só se todos os seus atributos contêm valores atômicos.

De acordo com a definição de relação, todas as relações respeitam a primeira forma normal.

Na prática, a primeira forma normal apenas estabelece a seguinte regra: todas as instâncias de uma relação têm o mesmo número de atributos e não existem duas instâncias de uma relação que sejam iguais. Resumindo, esta normalização é **indispensável** para que um esquema possa ser considerado do tipo relacional.

Por exemplo,

Produto (códigoP, designaçãoP, matériaPrima1, ..., matériaPrimaJ, ..., área)

não respeita a 1ª forma normal (nem é sequer uma relação) porque possui um número variável de atributos.

### **Segunda forma normal**

Uma relação R encontra-se na segunda forma normal se e só se verifica a primeira forma normal e os atributos que não são chave de R são dependentes da **totalidade** da chave primária de R.

É especialmente importante observar a segunda forma normal quando a chave de uma relação é composta por mais do que um atributo. De facto, esta regra estabelece que numa relação um atributo que não pertence à chave primária não pode depender apenas de uma parte dessa chave.

Quando uma relação não verifica esta forma normal é necessário retirar da relação o atributo dependente e criar uma nova relação sobre esse atributo e sobre a parte da chave de que ele depende.

Esta normalização visa diminuir a duplicação de dados.

Por exemplo,

Venda (códigoP, códigoC, dataV, quantidadeV, valorV, dataRecebi, identificaçãoC)

não se encontra na 2ª forma normal porque o atributo **identificaçãoC** depende do atributo **códigoC** que é apenas uma parte da chave primária da relação. A aplicação da 2ª forma normal a esta relação tem como consequência a definição de duas relações:

Comprador (códigoC, identificaçãoC)

Venda (códigoP, códigoC, dataV, quantidadeV, valorV, dataRecebim)

### **Terceira forma normal**

Uma relação R encontra-se na terceira forma normal se e só se verifica a segunda forma normal e todos os atributos que não são chave de R são não-transitivamente dependentes dessa chave.

Uma relação pode estar de acordo com a segunda forma normal e não respeitar a terceira forma normal quando um atributo que não pertence à chave primária da relação é funcionalmente dependente de outro atributo que também não pertence à chave da relação.

A aplicação da terceira forma normal conduz a que seja retirado da relação um dos atributos dependentes e a que se crie uma nova relação sobre os atributos que se verificou serem dependentes. O atributo que se mantém na relação inicial passa a ser uma chave estrangeira desta relação (visto figurar como chave principal de outra relação).

Por exemplo,

Produto (códigoP, designaçãoP, área, número, nome, especialidade)

não se encontra na 3ª forma normal porque os atributos **nome** e **especialidade** dependem do atributo **número** que não é chave primária da relação. Esta dependência existe porque os valores dos atributos **nome** e **especialidade** são determinados pelo valor do atributo **número** (uma vez que este atributo representa o número identificativo de cada trabalhador).

A aplicação da 3ª forma normal a esta relação conduz à definição de duas relações:

Trabalhador (número, nome, especialidade)

Produto (códigoP, designaçãoP, área, número)

Resumidamente, pode dizer-se que numa relação que verifica as primeiras três formas normais, qualquer atributo que não pertence à chave depende completamente e exclusivamente da totalidade da chave.

### **1.3. Passagem do modelo Entidade-Associação para relacional**

A importância de se iniciar a concepção do modelo de dados recorrendo a um modelo de tipo Entidade-Associação reside no facto destes modelos descreverem bem e de uma forma natural a realidade. Acresce ainda que, como não recorrem a um excesso de

formalismo, são suficientemente flexíveis para poderem ser utilizados eficazmente numa fase em que a estruturação dos dados é ainda confusa. A estas vantagens pode ainda adicionar-se a facilidade com que é possível efectuar a sua passagem para um SGBD relacional.

Para transformar um modelo Entidade-Associação num esquema relacional podem ser seguidas as seguintes regras:

1. Todo o tipo de entidades do modelo Entidade-Associação traduz-se por uma relação em que a chave primária e os atributos provêm do tipo de entidade.
2. Um tipo de associação de 1:n (um para muitos) entre dois tipos de entidades  $E_i$  e  $E_j$  que tenha uma multiplicidade igual a 0,1 ou 1,1 para um tipo de entidade  $E_j$  traduz-se por uma chave estrangeira na relação  $R$  que é tradução de  $E_j$ . Os atributos da associação traduzem-se por atributos na relação  $R$  que é tradução de  $E_i$ .
3. Um tipo de associação de 1:1 (um para um) entre dois tipos de entidades  $E_i$  e  $E_j$  é tratado como um caso especial do tipo de associação 1:n (explicado no ponto anterior): traduz-se por uma chave estrangeira na relação que é tradução de  $E_j$  ou de  $E_i$ ; se apenas para um destes tipos entidades a multiplicidade for 1,1 (sendo para a outra 0,1) dá-se preferência à relação que traduz este tipo de entidade, no caso contrário é indiferente qual a relação que é escolhida.
4. Um tipo de associação de n:n (muitos para muitos) traduz-se por uma relação em que a chave primária inclui as chaves estrangeiras que são chave primária dos tipos de entidade que a constituem; os outros atributos são a tradução dos tipos de atributos da associação (se esta possuir algum). No caso das chaves estrangeiras não serem suficientes para formar a chave primária da relação, na constituição desta devem também ser utilizados outros atributos de forma a ser obtida uma chave primária adequada à relação.

Estas regras permitem que se proceda de uma forma rápida (e quase automática) à passagem de um modelo Entidade-Associação em que só existam associações binárias para um esquema relacional.

Aplicando estas regras ao modelo E-A da Fig. 1 é possível obter o esquema relacional que se segue.

Aplicando a regra 1, obtêm-se:

Trabalhador (número, nome, dataNasc, especialidade)

Comprador (códigoC, identificaçãoC, moradaC, telefoneC)

MatériaPrima (códigoMP, designaçãoMP, quantArmaz)

Fornecedor (códigoF, identificaçãoF, moradaF, telefoneF)

Aplicando a regra 1 e a regra 2, obtêm-se:

Produto (códigoP, designaçãoP, área, número)

Aplicando a regra 3 e juntando um atributo às duas chaves estrangeiras para formar a chave primária, obtém-se:

Venda (códigoP, códigoC, dataV, quantidadeV, valorV, dataRecebi)

Utiliza (códigoP, códigoMP, dataU, quantidadeU)

Compra (códigoMP, códigoF, dataC, quantidadeC, valorC, dataPagam)

Note-se como nestas três últimas relações foram utilizados os sublinhados para distinguir entre os atributos que constituem a chave primária (representados com duplo sublinhado) aqueles que **também** são uma chave estrangeira (representados por um dos sublinhados tracejado e o outro contínuo).

Nota: Neste esquema foi evitada a utilização do carácter - nos identificadores de tabelas e atributos para facilitar a sua implementação e processamento no OpenOffice.org Base. Embora este carácter seja permitido, a sua utilização dificulta, por exemplo, a escrita de *queries* em SQL (1.5).

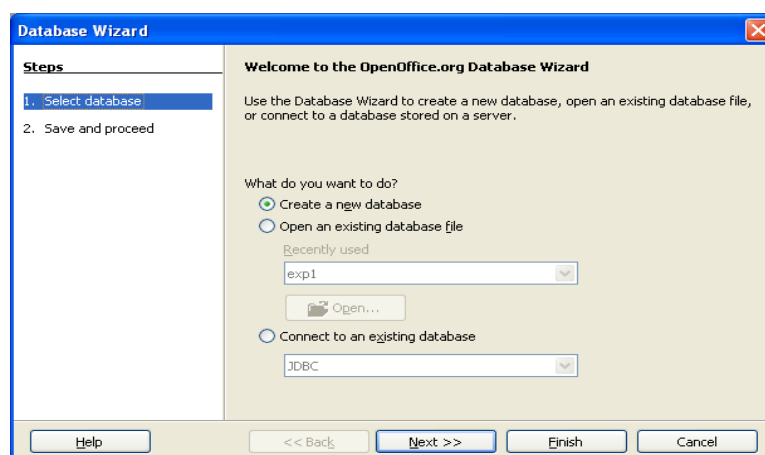
#### 1.4. OpenOffice.org Base

No **OpenOffice.org Base (Base)** ([www.openoffice.org](http://www.openoffice.org)) as tabelas são guardadas em ficheiros com a extensão .odb.

Para activar o **Base** seleccionar

Start → Programs → OpenOffice.org 2.1 → OpenOffice.org Base,

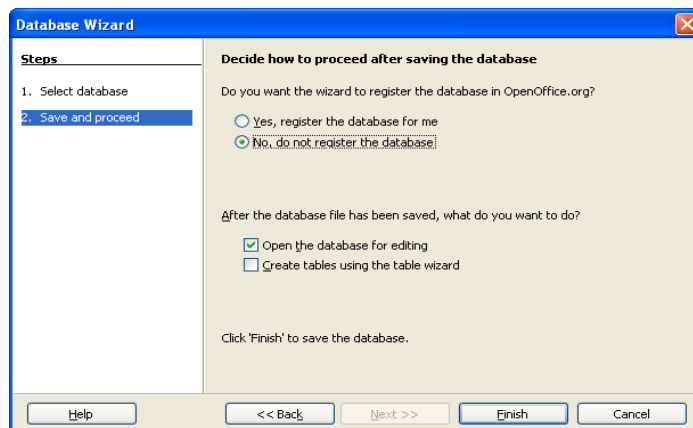
Para criar uma nova Base de Dados deve , na janela seguinte, escolher a opção **Create a new database** e carregar em **Next**.



Em seguida, deve seleccionar a opção **No, do not register the database** e manter seleccionada a opção **Open the database for editing**. Carregar em **Finish**.

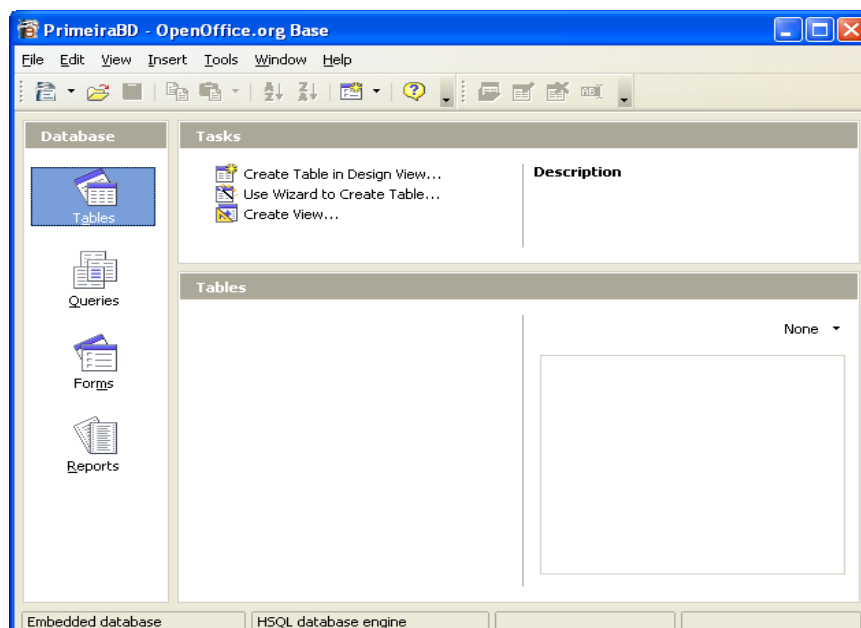
Posicione-se na directoria de trabalho e salve a base de dados com o nome de **PrimeiraBD**.

Para abrir uma base de dados já existente deveria, no primeiro quadro, ter seleccionado a opção **Open an existing data base file**.



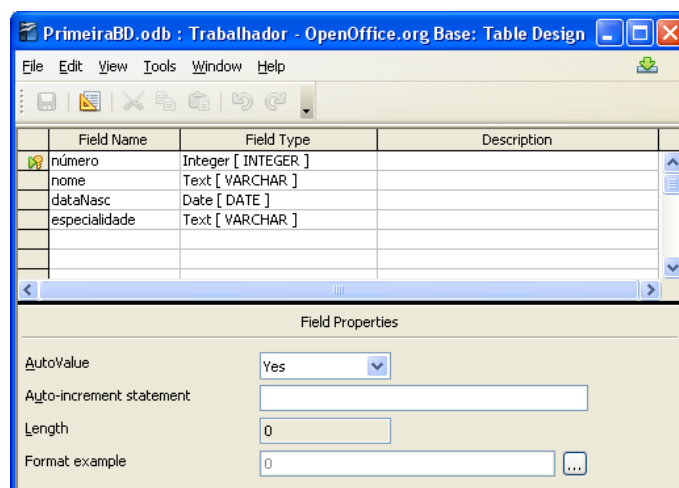
#### 1.4.1. Criação de tabelas

Antes de iniciar o registo dos dados de uma relação é necessário criar a respectiva tabela. O separador Tables permite criar, alterar e visualizar o esquema das relações que compõem a base de dados que está aberta; permite ainda aceder aos dados que se encontram guardados nas relações (ou tabelas).



Para criar a tabela **Trabalhador** deve seleccionar **Create Table in Design View**.

A janela de definição do esquema de uma relação inclui obrigatoriamente os identificadores dos atributos na coluna **Field Name** e o tipo dos dados que serão guardados em cada atributo na coluna **Field type**.



Os principais tipos de dados utilizados pelo Base são: **Integer [INTEGER]**, **Small Integer [SMALINT]**, **BigInt [BIGINT]**, para inteiros, **Double [DOUBLE]**, **Decimal [DECIMAL]** para reais, **Text [VARCHAR]** ou **Text [VARCHAR\_IGNORECASE]**, para caracteres, **YES/NO [BOOLEAN]** para booleanos e **Date [DATE]**, **Time [TIME]** (hh:mm:ss.00) para datas.

O tipo de dados de uma chave estrangeira deve ser exactamente o mesmo que foi definido na correspondente chave primária.

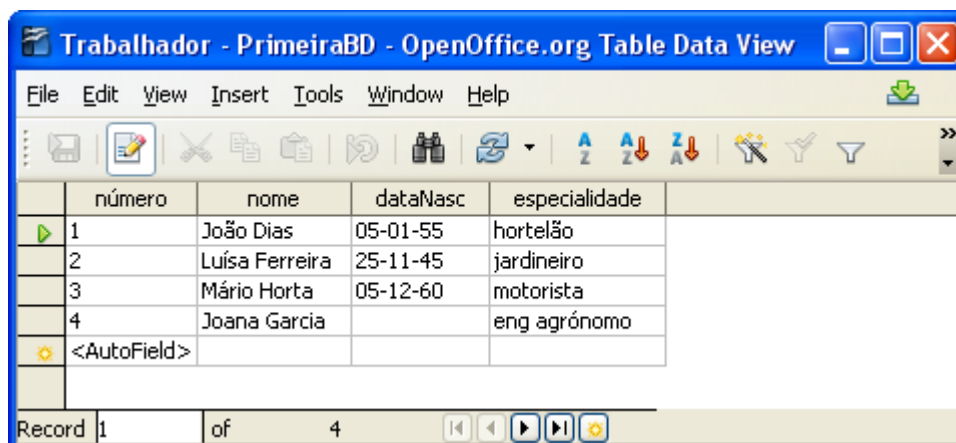
Na janela de definição do esquema das relações é ainda possível indicar outras propriedades dos atributos na área **Field Properties**. Por exemplo, **Entry required**, **Length**, **default value** e **Format example**.

Faz parte da definição de uma relação a indicação do(s) atributo(s) que é (são) chave primária da relação. Esta definição pode ser feita seleccionando o(s) referido(s) atributo(s) e clicando no botão direito do rato.

Para gravar o esquema da tabela fazer **File → Save** e atribuir o nome **Trabalhador**.

### 1.4.2. Introdução de dados

Depois de definir o esquema de uma relação e de o guardar é possível introduzir dados nessa relação, criando instâncias da relação ou registos. Para introduzir dados na tabela **Trabalhador** pode carregar no botão direito do rato e seleccionar **Open**.



	numero	nome	dataNasc	especialidade	
▶	1	João Dias	05-01-55	hortelão	
	2	Luísa Ferreira	25-11-45	jardineiro	
	3	Mário Horta	05-12-60	motorista	
	4	Joana Garcia		eng agrónomo	
☀	<AutoField>				

### 1.4.3. Implementação de restrições de integridade

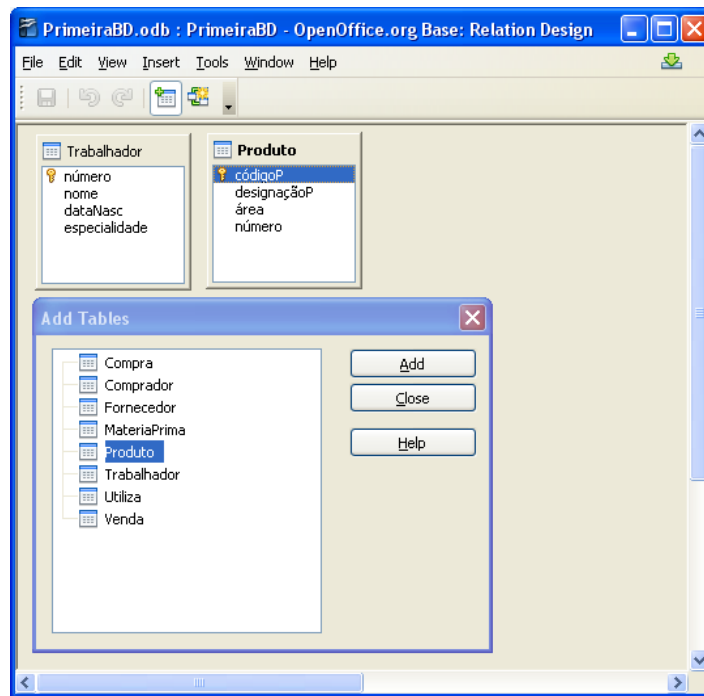
Como foi já referido (secção 1.2.4, pág. 12) o esquema de uma base de dados inclui a definição dos atributos de cada tabela e as restrições de integridade. As restrições de integridade de um modelo podem ser de vários tipos. Foi também já referido (secção 1.2.3, pág. 11) que o esquema de cada relação permite identificar restrições de três tipos: **integridade de domínio**, **integridade de entidade** e **integridade de referência**.

A implementação das restrições de integridade de domínio efectua-se pela definição do tipo de dados dos atributos, durante o processo de criação das tabelas.

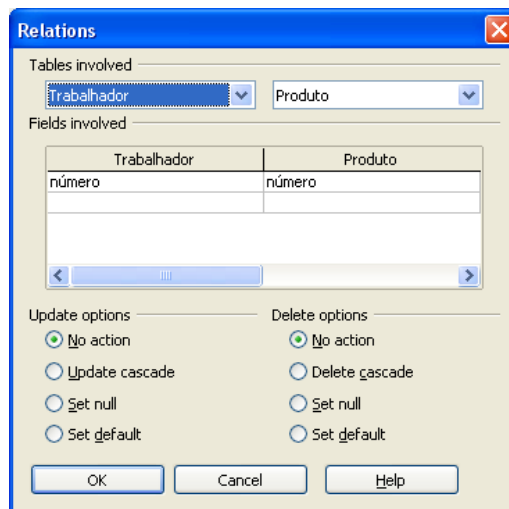
Analogamente, a implementação das restrições de integridade de entidade efectua-se, durante o mesmo processo, pela definição da chave primária de cada tabela.

Quanto à implementação das restrições de integridade de referência no **Base**, é necessário recorrer ao menu **Tools** → **Relationships**.

Para definir a restrição de integridade de referência da chave estrangeira **numero** da tabela **Produto** relativamente à correspondente chave primária, **numero** da tabela **Trabalhador**, deve seleccionar cada uma das tabelas fazendo **Add**.



Em seguida, seleccione **Insert** e **New Relation**. No menu **Relations** escolha as tabelas e os correspondentes campos para os quais quer definir a restrição de integridade de referência.

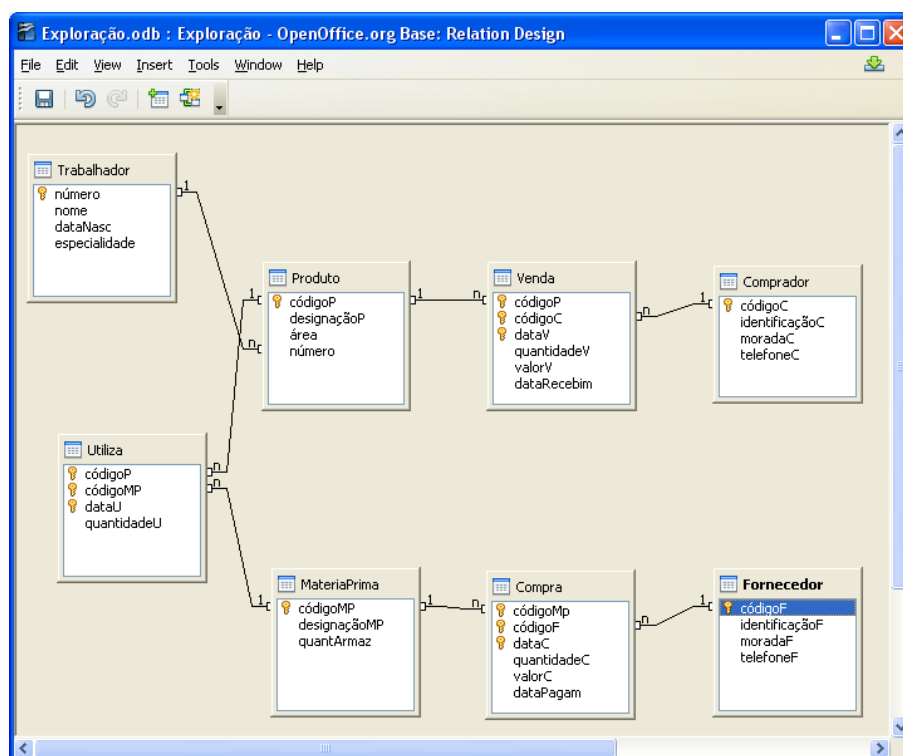


Em alternativa, pode seleccionar o atributo que é chave estrangeira e, com o botão esquerdo do rato, colocá-la sobre a correspondente chave primária.

Adicione um registo à tabela **Produto** e verifique que, depois da definição da restrição de integridade de referência e, contrariamente ao que sucedia anteriormente, no campo

**número** não é permitido um valor que não exista já no campo **número** da tabela **Trabalhador**.

Depois de criar a base de dados Exploração referente ao Exemplo1, com base no modelo relacional definido em 1.3 (pag. 15), e após terminar as definições das restrições de integridade de referência de todas as chaves estrangeiras do modelo, a janela **Relationships** deverá ser análoga à seguinte:

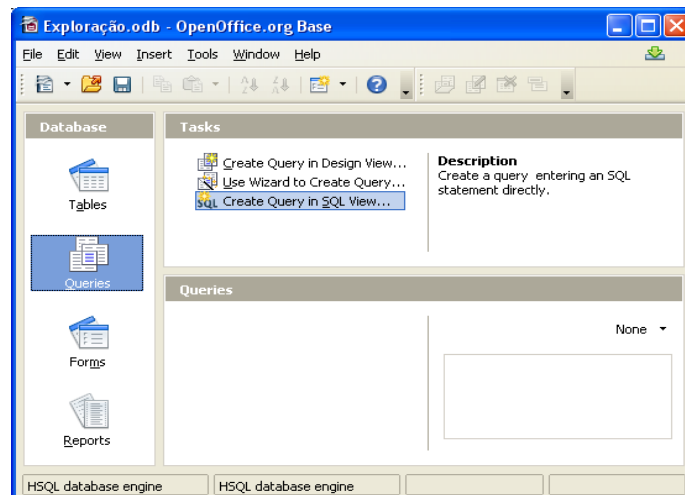


#### 1.4.4. Consultas e Actualizações

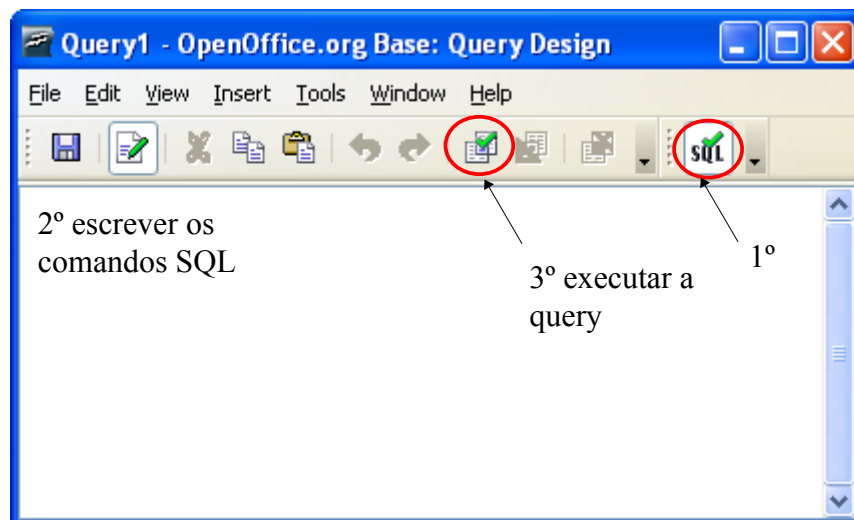
Geralmente, uma base de dados depois de criada é sujeita a inúmeras actualizações e consultas dos seus dados. Estas operações são genericamente denominadas **Queries**.

As regras para a criação do esquema de uma base de dados relacional permitiram o desenvolvimento da linguagem SQL que permite efectuar eficientemente diversos tipos de *queries*.

Para iniciar a criação de uma *query* em **SQL view** deve posicionar-se na janela **Queries** e carregar em **Creat Query in SQL View ...**



Na janela criada, seleccionar o icon **SQL** na barra de ferramentas de modo a executar os comandos da linguagem SQL directamente no motor da base de dados <sup>1</sup>.



## 1.5. Structured Query Language (SQL)

**SQL** é uma linguagem normalizada (ANSI) utilizada para manipular a informação de bases de dados relacionais. Com ela podemos criar, consultar e actualizar bases de dados relacionais.

### 1.5.1. A instrução SELECT

As consultas a uma base de dados relacional fazem-se em SQL recorrendo à instrução **SELECT**. Esta instrução permite criar conjuntos de registos de uma ou mais tabelas da base de dados seleccionados segundo diversos critérios.

<sup>1</sup> O OpenOffice.org Base utiliza como motor da base de dados o HSQLDB ([www.hsqldb.org](http://www.hsqldb.org))

A sintaxe completa da instrução **SELECT** é complexa. Por este motivo, primeiramente serão utilizados alguns exemplos de utilização de variantes desta instrução e só depois será apresentada a sua sintaxe completa.

A forma mais simples de utilização da instrução **SELECT** é aquela que utiliza apenas a cláusula **FROM** e que selecciona todos os registos de uma tabela.

Sintaxe — 1ª variante:

```
SELECT { * | table.* | [table.]field1 [, [table.]field2 [, ...]]}  
FROM table;
```

onde

- \*** especifica que todos os campos devem ser seleccionados
- table** especifica o nome da tabela que contém os campos e os registos seleccionados
- field** especifica os nomes dos campos que são seleccionados

Exemplo<sup>2</sup> 2 - Seleccionar todos os produtos e os valores de todos os seus atributos.

```
SELECT *  
FROM Produto;
```

Conjunto de registos seleccionados<sup>3</sup>:

<b>códigoP</b>	<b>designaçãoP</b>	<b>área</b>	<b>número</b>
5	cenoura	4.5	4
6	feijão verde	6.5	1
7	batata	2.6	4
8	rosa	1	2

### A cláusula **WHERE**

A cláusula **WHERE** da instrução **SELECT** permite especificar uma condição (simples ou composta) que os registos seleccionados verificam.

Sintaxe — 2ª variante:

```
SELECT fieldlist
```

2 Todos os exemplos que se seguem referem-se à base de dados Exploração (<\\Prunus\home\cadeiras\matinf\BDados\Exploração.odt>)

3 Nota: Para correr estas *queries* na versão 2.2 do OpenOffice.org Base é necessário colocar entre aspas os nomes das tabelas e dos campos, de modo a se distinguir caracteres maiúsculos de minúsculos. No entanto, para não tornar demasiado pesado o código SQL apresentado, as aspas foram omitidas neste texto.

<b>FROM table</b> <b>WHERE condition;</b>
--

onde

**fieldlist** especifica os nomes dos campos que são seleccionados

**table** especifica o nome da tabela que contém os campos e os registos seleccionados

**condition** uma condição que os registos seleccionados verificam; podem ser utilizados operadores relacionais (<, <=, >, >=, =, <>) , operadores lógicos (NOT, AND, OR) e os operadores IN, IS, BETWEEN e LIKE.

Exemplo 3 - Seleccionar os códigos dos produtos vendidos desde 1999-04-01, os códigos dos compradores que os compraram, as datas destas vendas e os respectivos valores.

```

SELECT códigoP, códigoC, dataV, valorV
FROM Venda
WHERE dataV > {D '1999-04-01'};

```

Conjunto de registos seleccionados:

códigoP	códigoC	dataV	valorV
5	1	1999-04-07	45000
6	2	1999-04-05	60000

Exemplo 4 - Seleccionar as vendas cuja quantidade seja superior a 50 e inferior a 200 ou cujo valor da venda seja não inferior a 60000, indicando os códigos dos produtos vendidos, os códigos dos compradores que os compraram, as quantidades vendidas e os respectivos valores.

```

SELECT códigoP, códigoC, quantidadeV, valorV
FROM Venda
WHERE (quantidadeV > 50 AND quantidadeV < 200)
OR valorV >= 60000;

```

Conjunto de registos seleccionados:

códigoP	códigoC	quantidadeV	valorV
6	2	100	60000
6	1	450	90000

Exemplo 5 - Seleccionar as vendas cuja quantidade não seja superior a 50 e inferior a 200 e cujo valor da venda seja inferior a 60000, indicando os códigos dos produtos vendidos, os códigos dos compradores que os compraram, as quantidades vendidas e os respectivos valores.

```

SELECT códigoP, códigoC, quantidadeV, valorV
FROM Venda
WHERE NOT ((quantidadeV > 50 AND quantidadeV < 200)

```

**OR** valorV >= 60000);

Conjunto de registos seleccionados:

<b>códigoP</b>	<b>códigoC</b>	<b>quantidadeV</b>	<b>valorV</b>
5	1	50	45000
5	2	5.6	1000

Exemplo 6 - Seleccionar o nome, a especialidade e a data de nascimento dos trabalhadores cuja especialidade é hortelão ou jardineiro.

```
SELECT nome, especialidade, dataNasc  
FROM Trabalhador  
WHERE especialidade IN ('hortelão', 'jardineiro');
```

Conjunto de registos seleccionados:

<b>nome</b>	<b>especialidade</b>	<b>dataNasc</b>
João Dias	hortelão	1955-01-05
Luísa Ferreira	jardineiro	1945-11-25

Exemplo 7 - Seleccionar o nome, a especialidade e a data de nascimento dos trabalhadores cuja especialidade é hortelão ou jardineiro ou cuja data de nascimento esteja entre 1955-01-01 e 1965-12-31.

```
SELECT nome, especialidade, dataNasc  
FROM Trabalhador  
WHERE especialidade IN ('hortelão', 'jardineiro') OR  
dataNasc BETWEEN {D '1955-01-01'} AND {D '1965-12-31'};
```

Conjunto de registos seleccionados:

<b>nome</b>	<b>especialidade</b>	<b>dataNasc</b>
João Dias	hortelão	1955-01-05
Luísa Ferreira	jardineiro	1945-11-25
Mário Horta	motorista	1960-12-05

Exemplo 8 - Seleccionar o nome, a especialidade e a data de nascimento dos trabalhadores cujo nome começa por J.

```
SELECT nome, especialidade, dataNasc  
FROM Trabalhador  
WHERE nome LIKE 'J%';
```

Conjunto de registos seleccionados:

<b>nome</b>	<b>especialidade</b>	<b>dataNasc</b>
João Dias	hortelão	1955-01-05
Joana Garcia	eng agrónomo	

Nota: % substitui um número qualquer de caracteres, enquanto \_ substitui um único carácter.

Exemplo 9 - Seleccionar o nome, a especialidade e a data de nascimento dos trabalhadores cujo nome não começa por *J*.

```
SELECT nome, especialidade, dataNasc
FROM Trabalhador
WHERE nome NOT LIKE 'J%';
```

Conjunto de registos seleccionados:

nome	especialidade	dataNasc
Luísa Ferreira	jardineiro	1945-11-25
Mário Horta	motorista	1960-12-05

Exemplo 10 - Seleccionar os valores de vendas que não estejam ainda pagas e os códigos dos respectivos compradores.

```
SELECT códigoC, valorV
FROM Venda
WHERE dataRecebim IS NULL;
```

Conjunto de registos seleccionados:

códigoC	valorV
2	1000
2	60000
1	56000

### A cláusula FROM

A cláusula **FROM** especifica o(s) nome(s) da(s) tabela(s) em que se encontram os registos a seleccionar. Quando é indicada mais do que uma tabela, a instrução SELECT produz um conjunto de registos cujos campos são os indicados na lista de campos. Os registos seleccionados correspondem aos tuplos do produto cartesiano das tabelas especificadas, isto é, cada registo é composto por um registo de cada uma dessas tabelas.

Sintaxe — 3ª variante: <b>SELECT fieldlist</b> <b>FROM tableexpression</b> <b>[ WHERE condition ] ;</b>
--

onde

**tableexpression** especifica o(s) nome(s) da(s) tabela(s) que contém os campos e os registos seleccionados

Exemplo 11:

```
SELECT designaçãoP, área, Produto.número, Trabalhador.número, nome,
       especialidade
FROM Produto, Trabalhador;
```

Conjunto de registos seleccionados:

DesignaçãoP	área	Produto.número	Trabalhador.número	nome	especialidade
cenoura	4.5	4	1	João Dias	hortelão
feijão verde	6.5	1	1	João Dias	hortelão
batata	2.6	4	1	João Dias	hortelão
rosa	1	2	1	João Dias	hortelão
cenoura	4.5	4	2	Luísa Ferreira	jardineiro
feijão verde	6.5	1	2	Luísa Ferreira	jardineiro
batata	2.6	4	2	Luísa Ferreira	jardineiro
rosa	1	2	2	Luísa Ferreira	jardineiro
cenoura	4.5	4	3	Mário Horta	motorista
feijão verde	6.5	1	3	Mário Horta	motorista
batata	2.6	4	3	Mário Horta	motorista
rosa	1	2	3	Mário Horta	motorista
cenoura	4.5	4	4	Joana Garcia	eng agrónomo
feijão verde	6.5	1	4	Joana Garcia	eng agrónomo
batata	2.6	4	4	Joana Garcia	eng agrónomo
rosa	1	2	4	Joana Garcia	eng agrónomo

No exemplo anterior a qualificação do atributo número com o identificador da tabela a que o atributo pertence (isto é, a referência a um dos atributos por meio de Trabalhador.número e a outro por meio de Produto.número) é obrigatória porque nas tabelas referidas pela cláusula FROM existem atributos que utilizam um identificador comum (número).

A instrução SELECT do exemplo anterior efectua o **produto cartesiano** das duas tabelas indicadas na cláusula FROM. De facto, considerando cada tabela (relação) como um conjunto cujos elementos são os tuplos que constituem as suas instâncias (ou registos), teremos os conjuntos seguintes:

Produto = {(5, "cenoura", 4.5, 4), (6, "feijão verde", 6.5, 1), ...}

Trabalhador = {(1, "João Dias", {D '1955-01-05'}, "hortelão"), (2, "Luisa Ferreira", {D '1945-11-25'}, "motorista"), ...}.

Assim, de acordo com a definição matemática, o produto cartesiano destes 2 conjuntos é o conjunto seguinte:

Produto × Trabalhador = {(5, "cenoura", 4.5, 4, 1, "João Dias", {D '1955-01-05'}, "hortelão"), (5, "cenoura", 4.5, 4, 2, "Luisa Ferreira", {D '1945-11-25'}, "motorista"), (6, "feijão verde", 6.5, 1, 1, "João Dias", {D '1955-01-05'}, "hortelão"), (6, "feijão verde", 6.5, 1, 2, "Luisa Ferreira", {D '1945-11-25'}, "motorista"), ...}.

Os atributos referidos na instrução SELECT modificam um pouco o resultado do produto cartesiano: restringem os elementos dos tuplos que são obtidos (no exemplo não são seleccionados os atributos dataNasc da tabela Trabalhador e códigoP da tabela Produto) e especificam a ordem pela qual o resultado deve apresentar os atributos.

Quando a cláusula FROM refere mais do que uma tabela, a cláusula WHERE é frequentemente utilizada para seleccionar no resultado do produto cartesiano os tuplos que correspondem a registos em que o valor de uma chave estrangeira é igual ao **valor** de uma chave primária. Neste caso, a instrução SELECT produz **um subconjunto do produto cartesiano** das tabelas referidas na cláusula FROM e é usual dizer-se que se efectua um *join* ou cruzamento de tabelas.

Exemplo 12 - Para cada produto seleccionar a designação, a área e o número, nome e especialidade do respectivo responsável

```
SELECT designaçãoP, área, Trabalhador.número, nome, especialidade
FROM Produto, Trabalhador
WHERE Produto.número=Trabalhador.número;
```

Conjunto de registos seleccionados:

designaçãoP	área	número	nome	especialidade
feijão verde	6.5	1	João Dias	hortelão
rosa	1	2	Luísa Ferreira	jardineiro
cenoura	4.5	4	Joana Garcia	eng agrónomo
batata	2.6	4	Joana Garcia	eng agrónomo

### Eliminação de registos duplicados

Algumas *queries* podem conduzir a conjuntos de registos com duplicações. Os registos duplicados podem ser eliminados recorrendo à utilização de **DISTINCT**.

Sintaxe — 4ª variante:

```
SELECT DISTINCT fieldlist
FROM tableexpression
WHERE condition;
```

Exemplo 13 - Seleccionar o número, nome e especialidade dos trabalhadores que são responsáveis por algum produto.

```
SELECT DISTINCT Trabalhador.número, nome, especialidade  
FROM Produto, Trabalhador  
WHERE Produto.número=Trabalhador.número;
```

Conjunto de registos seleccionados:

número	nome	especialidade
1	João Dias	hortelão
2	Luísa Ferreira	jardineiro
4	Joana Garcia	eng agrónomo

### Sub-query numa cláusula WHERE

Na cláusula WHERE pode ainda figurar uma *query*, neste caso denominada *sub-query*, constituída por outra instrução SELECT. O resultado da *sub-query* define um conjunto de valores que são utilizados na condição especificada pela cláusula WHERE.

Exemplo 14 - Seleccionar as designações dos produtos dos quais já se tenham efectuado vendas, e o nome e a especialidade dos respectivos responsáveis.

```
SELECT nome, especialidade, designaçãoP  
FROM Trabalhador, Produto  
WHERE Trabalhador.número=Produto.número  
AND Produto.códigoP IN (SELECT códigoP  
FROM Venda);
```

Conjunto de registos seleccionados:

nome	especialidade	designaçãoP
Joana Garcia	eng agrónomo	cenoura
João Dias	hortelão	feijão verde

### A cláusula ORDER BY

A cláusula **ORDER BY** apenas ordena os registos seleccionados pelos valores de um conjunto de campos especificado.

Sintaxe — 5ª variante:

```
SELECT fieldlist  
FROM tableexpression  
WHERE condition  
ORDER BY field1 [ASC | DESC ], field2 [ASC | DESC ], ...];
```

onde

**ASC** especifica a ordem ascendente  
**DESC** especifica a ordem descendente.

Exemplo 15 - Para cada produto seleccionar a designação, a área e o número, nome e especialidade do respectivo responsável. O resultado deve ser apresentado por ordem decrescente do número do trabalhador e por ordem crescente da área do produto.

```
SELECT designaçãoP, área, Trabalhador.número, nome, especialidade
FROM Produto, Trabalhador
WHERE Produto.número=Trabalhador.número
ORDER BY Trabalhador.número DESC, área ASC;
```

Conjunto de registos seleccionados:

designaçãoP	área	número	nome	especialidade
batata	2.6	4	Joana Garcia	eng agrónomo
cenoura	4.5	4	Joana Garcia	eng agrónomo
rosa	1	2	Luísa Ferreira	jardineiro
feijão verde	6.5	1	João Dias	hortelão

Exemplo 16 - Para cada produto cujo responsável tem a especialidade *eng agrónomo* seleccionar a designação, a área e o número, nome e especialidade do respectivo responsável. O resultado deve ser apresentado por ordem decrescente da área do produto.

```
SELECT designaçãoP, área, Trabalhador.número, nome, especialidade
FROM Produto, Trabalhador
WHERE Produto.número=Trabalhador.número
AND especialidade='eng agrónomo'
ORDER BY área DESC;
```

Conjunto de registos seleccionados:

designaçãoP	área	número	nome	especialidade
cenoura	4.5	4	Joana Garcia	eng agrónomo
batata	2.6	4	Joana Garcia	eng agrónomo

### Aliases

Nos exemplos anteriores os nomes das tabelas e dos campos utilizados na instrução SELECT e nos resultados das *queries* têm coincidido com os nomes definidos no esquema da base de dados. Mas, é possível definir nomes novos, chamados **Aliases**, numa instrução SELECT.

Exemplo 17 - Para cada produto, seleccionar a designação, a área e o nome e especialidade do respectivo responsável.

```

SELECT designaçãoP AS prod, área, nome AS responsável, especialidade
FROM Produto AS P, Trabalhador AS T
WHERE P.número=T.número;

```

Conjunto de registos seleccionados:

prod	área	responsável	especialidade
feijão verde	6.5	João Dias	hortelão
rosa	1	Luísa Ferreira	jardineiro
cenoura	4.5	Joana Garcia	eng agrónomo
batata	2.6	Joana Garcia	eng agrónomo

### Funções de agregação

Na instrução SELECT podem ser utilizadas funções de agregação em vez de uma lista de atributos. Estas funções possibilitam o resumo dos resultados de uma selecção de registos, como alternativa à selecção de registos. Podem ser utilizadas as funções SUM, AVG, MAX, MIN e COUNT.

Exemplo 18 - Seleccionar a área total de todos os produtos, a área média por produto e o número de produtos existentes.

```

SELECT SUM (área) AS "Área total", AVG (área) AS "Área média",
COUNT(*) AS "Nº de produtos"
FROM Produto;

```

Conjunto de registos seleccionados:

Área total	Área média	Nº de produtos
14. 6	3.65	4

Neste exemplo os caracteres " " são necessários porque se pretende utilizar identificadores que incluem espaços; sucederia o mesmo se os identificadores incluíssem outros caracteres especiais como o carácter – ou caso se pretenda manter caracteres maiúsculos e minúsculos.

### Cláusula GROUP BY ... HAVING

A utilização de funções de agregação é feita frequentemente em conjunto com a cláusula GROUP BY. Esta especifica os conjuntos de registos seleccionados que são objecto da(s) função(ões).

Nota: quando é utilizada a cláusula GROUP BY, só podem ser indicados na cláusula SELECT os atributos incluídos na cláusula GROUP BY (para além daqueles que são objecto de uma função).

Exemplo 19 - Para cada produto vendido, seleccionar o respectivo código, número de vendas e a quantidade total dessas vendas.

```
SELECT códigoP, COUNT(códigoP) AS "Nº de vendas",
      SUM(quantidadeV) AS "Quantidade Tot"
FROM Venda
GROUP BY códigoP;
```

Conjunto de registos seleccionados:

códigoP	Nº de vendas	Quantidade Tot
5	2	55.6
6	2	550

Exemplo 20 - Para cada produto com vendas não pagas, seleccionar o respectivo código, número de vendas por pagar e a quantidade total dessas vendas.

```
SELECT códigoP, COUNT(códigoP) AS "Nº de vendas",
      SUM(quantidadeV) AS "Quantidade Tot"
FROM Venda
WHERE dataRecebig IS NULL
GROUP BY códigoP;
```

Conjunto de registos seleccionados:

códigoP	Nº de vendas	Quantidade Tot
5	1	5.6
6	2	550

A cláusula **GROUP BY** pode ser utilizada em conjunto com **HAVING** para restringir os registos seleccionados àqueles que verificam a condição especificada em **HAVING**.

Exemplo 21 - Para cada produto com vendas não pagas numa quantidade total superior ou igual a 250, seleccionar o respectivo código, número de vendas por pagar e a quantidade total dessas vendas

```
SELECT códigoP, COUNT(códigoP) AS "Nº de vendas",
      SUM(quantidadeV) AS "Quantidade Tot"
FROM Venda
WHERE dataRecebig IS NULL
GROUP BY códigoP
HAVING SUM(quantidadeV)>=250;
```

Conjunto de registos seleccionados:

códigoP	Nº de vendas	Quantidade Tot
6	2	550

### Expressões aritméticas e funções

Nos exemplos anteriores foram utilizadas expressões lógicas em diversas das cláusulas da instrução SELECT mas, também é possível utilizar expressões aritméticas envolvendo dados de tipos numéricos e funções.

Exemplo 22 - Para cada venda efectuada e ainda não paga que em 1999-04-30 já excedesse os 30 dias de dívida, seleccionar o código do produto, o valor da venda, o IVA (supondo que o valor da venda inclui 17% de IVA), a data da venda e o número de dias de dívida contados até 1999-04-30.

```
SELECT códigoP, valorV, Round(valorV*0.17/1.17) AS IVA, dataV,  
       DateDiff('day', dataV, {D '1999-04-30'}) AS "Dias de dívida"  
FROM Venda  
WHERE dataRecebig IS NULL AND DateDiff('day', dataV, {D '1999-04-30'}) > 30;
```

Conjunto de registos seleccionados:

<b>códigoP</b>	<b>valorV</b>	<b>IVA</b>	<b>dataV</b>	<b>Dias de dívida</b>
5	1000	145	1999-02-20	69
6	90000	13077	1999-03-29	32

Algumas das funções que o OpenOffice.org Base disponibiliza para o processamento de datas:

CurDate(), Now(), CurTime() — retornam o valor actual da data, da data e da hora, e da hora, respectivamente.

DateDiff(string, date1, date2) — retorna o número de unidades de tempo (definida por *string*) que vão desde a data1 à data2.

### Instrução SELECT - sintaxe

A sintaxe genérica da instrução SELECT utilizada nos exemplos anteriores é a seguinte:

```
SELECT [predicate] { * | table.* | [table.]field1 [AS alias1] [,  
    [table.]field2 [AS alias2] [, ...] }  
FROM tableexpression [, ...]  
[WHERE... ]  
[GROUP BY...]
```

```
[HAVING... ]  
[ORDER BY... ];
```

onde

**predicate** pode ser ALL ou DISTINCT ou DISTINCTROW ou TOP n [PERCENT]

### **O operador UNION**

O operador UNION é usado para combinar o resultado de duas ou mais instruções SELECT num único conjunto de resultados. Cada instrução SELECT tem de ter como resultado o mesmo número de colunas e as colunas correspondentes, nas várias instruções SELECT, têm de ter tipos de dados compatíveis.

No resultado final, o nome das colunas corresponde ao do primeiro SELECT.

Exemplo 23 - Seleccionar os códigos e as identificações de todos os compradores e fornecedores indicando os que são clientes e os que são fornecedores.

```
SELECT códigoC AS código, identificaçãoC AS Nome, 'Cliente' AS Tipo  
FROM Comprador  
UNION  
SELECT códigoF, identificaçãoF, 'Fornecedor'  
FROM Fornecedor;
```

Conjunto de registos seleccionados:

<b>código</b>	<b>Nome</b>	<b>Tipo</b>
1	Manuel Maria	Cliente
2	Luisa Fraga	Cliente
3	Duarte Silva	Cliente
1	Ana Sousa	Fornecedor

### **1.5.2. Outras instruções SQL**

Além da instrução SELECT, a linguagem SQL inclui instruções para definição do esquema de uma base de dados (por exemplo, CREATE TABLE e ALTER TABLE) e para alteração dos dados (INSERT, DELETE e UPDATE).

A sintaxe e a semântica destas instruções não serão, no entanto, aqui referidas.

## 1.6. Alteração do esquema de uma base de dados

Pode ser necessário alterar o esquema de uma BD porque:

1. a realidade sofre alterações e é importante que a base de dados represente a nova realidade;
2. os objectivos da base de dados são modificados;
3. esquema inclui erros que não permitem que a base de dados cumpra os objectivos estipulados.

As alterações a um esquema que requerem apenas a criação de novas tabelas não constituem um problema difícil. As alterações a um esquema, que exigem a alteração de tabelas definidas anteriormente, podem constituir um problema mais difícil, sobretudo se a BD já contém um volume de dados significativo e se as alterações requerem a reorganização desses dados.

Assim, o esquema de uma base de dados que já se encontra em funcionamento apenas deve ser alterado quando tal se torne imprescindível. Dos três motivos anteriormente referidos, os dois primeiros são incontroláveis mas, o risco de ocorrer a terceira situação pode ser minimizado de várias formas. Neste sentido, convém realçar a importância da fase inicial de conceptualização da base de dados (também denominada fase de análise): é fundamental o desenho de um modelo conceptual de dados (por exemplo, de tipo Entidade-Associação) de modo a representar o mais correctamente possível a realidade que se pretende descrever.

No sentido de avaliar as consequências da alteração de um esquema, considere-se uma alteração à base de dados que tem vindo a ser utilizada. No modelo conceptual que foi criado ignorou-se que, usualmente, uma venda é identificada por meio de um documento (por exemplo, factura) onde são registadas as características de cada venda: a data, cada um dos produtos vendidos, respectivas quantidades, preços, ... . O mesmo sucede com as compras de matérias primas. O modelo E-A da Fig. 3 representa estes aspectos.

As alterações à BD introduzidas por este modelo de dados requerem, em primeiro lugar, que sejam adicionadas quatro tabelas ao esquema relacional anteriormente criado e que sejam modificados os esquemas de duas das tabelas já existentes. Se já existirem dados nestas duas tabelas, será ainda necessário criar os registos correspondentes nas novas tabelas e alterar todos os registos das duas tabelas modificadas (acrescentando-lhes os valores dos novos atributos). Finalmente, também algumas das *queries* utilizadas como exemplos de instruções em SQL necessitariam de ser alteradas, no caso de se pretender manter os resultados obtidos.

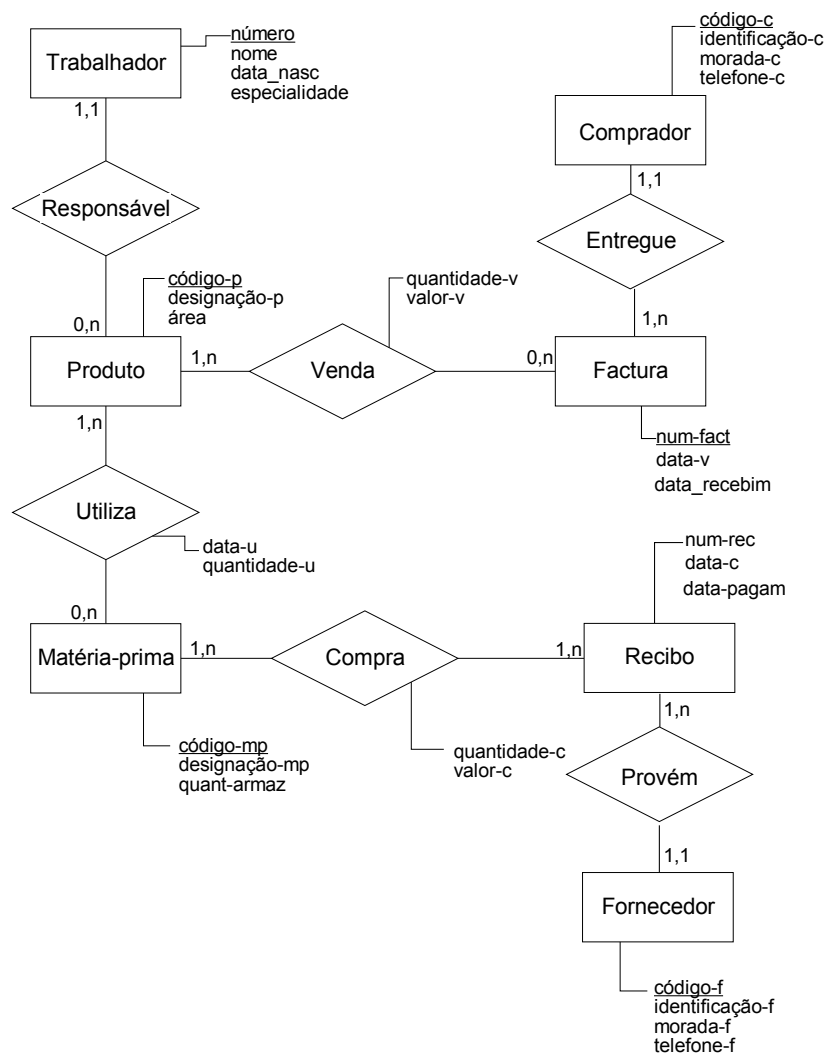


Fig. 3 – Alteração ao modelo conceptual de dados