Capítulo 4
Projecto de Bases de Dados

Versão provisória 2001-05-04

Conteúdos

Índice de Figuras

4.1 Introdução

Este capítulo trata principalmente das Bases de Dados e respectivos Sistemas de Gestão, ou SGBD, que se fundamentam teoricamente no modelo relacional [Codd 1970][1]. Serão no entanto também brevemente abordadas as Bases de Dados baseadas no modelo Objecto-Relacional e referidas as Bases de Dados assentes em outros modelos menos generalizados.

Embora se aborde em geral o tema das bases de dados, arquitecturas e modelos mais relevantes, o principal objectivo deste capítulo é o de apresentar de uma forma prática o modelo relacional e o processo de normalização de dados funcional, incluindo uma introdução à linguagem SQL. Dado um problema de gestão de informação, é essencial a identificação de uma estrutura de dados relacional de boa qualidade para se virem a aproveitar bem todas as capacidades dos SGBD comerciais, tais como o MS Access ou o Oracle. Uma boa compreensão do processo de normalização de dados funcional é fundamental para se conseguirem obter aplicações de elevada qualidade.

O processo de normalização de dados funcional só é aplicável a problemas de pequena e média dimensão, e tem algumas limitações. Para problemas reais complexos, recorre-se normalmente a processos de normalização iniciados com abordagens mais gerais baseadas no modelo entidade-associação, anteriormente descritas no Capítulo 3 sobre modelação conceptual de classes. No entanto, a compreensão do processo de normalização é indispensável, visto que os conceitos que esta técnica introduz são recorrentes em todos os domínios relacionados com o projecto e a administração de bases de dados relacionais, e também com a implementação e optimização do seu desempenho.

Na secção seguinte será apresentada uma base de dados muito simples, os principais tipos de operações que se podem efectuar em geral sobre base de dados e exemplos dessas operações na linguagem SQL. Desta forma inicia-se a apresentação das principais características dos sistemas de gestão de bases de dados, terminando com alguns exemplos concretos com base no Access e no Oracle.

A secção 4.3 introduz a álgebra relacional, ou seja as noções matemáticas em que se baseia o modelo relacional de bases de dados, em particular do ponto de vista das operações abstractas

---

[1] [Date 1998] refere que o primeiro trabalho de Codd sobre o modelo relacional deu origem ao artigo "Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks", um relatório de investigação da IBM com distribuição restrita (IBM Research Report RJ599, 1969-08-19). Este documento foi publicado em 1969, ou seja um ano antes da versão actualizada que apareceu na *Communications of the ACM* com o título "A Relational Model of Data for Large Shared Data Banks" [Codd 1970]. Este último trabalho tem sido creditado como a origem do modelo relacional. Nestes dois artigos Codd transformou radicalmente todo o estudo dos sistemas de gestão de base de dados, tornando-o numa ciência com bases matemáticas sólidas e elegantes. Surge assim o modelo relacional como base para uma teoria ou modelo de dados em geral. Codd realçou desde logo a separação entre *modelo* e *implementação*, fundamentou com a lógica de predicados a gestão de bases de dados e definiu uma álgebra e um cálculo relacional. Codd introduziu ainda informalmente a primeira linguagem de manipulação de dados relacional (ALPHA), definiu o conceito de dependência funcional, introduziu as noções de chave primária, chave alheia (*foreign key*) e chave candidata (embora com designações diferentes das utilizadas actualmente) e definiu as três primeiras formas normais (1FN, 2FN e 3FN).

sobre informação. A linguagem SQL, introduzida sucintamente na secção 4.4, baseia-se claramente na álgebra relacional e no cálculo relacional.

No capítulo 3 apresentou-se a modelação conceptual de classes, que permite obter por tradução simples o esquema conceptual de uma base de dados relacional. Para problemas de dimensão média ou grande é esse tipo de análise que deve ser seguida. No entanto para problemas de pequena dimensão, ou quando é necessário efectuar ajustes finos a um problema, pode ser seguido o método da normalização funcional para identificar as relações ou tabelas que a base de dados de modelo relacional deve conter. Este método de normalização funcional é explicado na secção 4.5, onde são claramente definidos os conceitos de chave primária e de formas normais. Quando se deseja que um sistema de gestão de informação tenha uma resposta muito rápida na presença de grandes volumes de dados pode ser necessário desnormalizar as relações. A compreensão plena deste processo não é fácil com modelação conceptual de classes.

A secção 4.6 ilustra, com base no caso de um sistema para uma empresa de venda de vinhos, todo o processo de definição da base de dados relacional, seguindo a normalização funcional.

Finalmente na secção 4.7, referem-se sucintamente outros modelos de bases de dados que existem e que podem ser utilizados em casos específicos, e conclui-se o presente capítulo com as perspectivas para as bases de dados em geral e do modelo relacional em particular.

As secções e discutem-se as vantagens e desvantagens relativas. Finalmente na secção 4.8 apresentam-se alguns exercícios.

4.2 Bases de Dados e Sistemas de Gestão de Bases de Dados

Os primeiros sistemas informáticos surgiram para realizar cálculos numerosos ou complexos e para arquivar e processar grandes volumes de informação. No início da informática os dados eram organizados em ficheiros, cada vez mais complexos. Rapidamente surgiram necessidades de criar e manter vários ficheiros, e de manter relações entre a informação neles contida. Esta informação era usada para dar resposta a múltiplos requisitos dos utilizadores e gestores, inicialmente com resposta através de relatórios ou outros documentos, produzidos a pedido, e mais tarde através de sistemas interactivos, com interfaces pré-programadas. A complexidade destas Bases de Dados deu origem à criação de programas especiais preparados para facilitar todas as tarefas de gestão e utilização dos dados, os designados Sistemas de Gestão de Bases de Dados ou SGBD. Estes sistemas basearam-se inicialmente na gestão de ficheiros com informação relacionada entre si através de apontadores nos respectivos registos. A consulta ou navegação nessas bases de dados requeria o conhecimento dos apontadores existentes e tornava complicado o acesso a quem não dominava a linguagem de programação e interface do SGBD.

Esta secção introduz as BD e os SGBD de modelo relacional, modelo que actualmente é utilizado por todos os sistemas comerciais mais divulgados, e que exige apenas do utilizador conhecimentos do esquema conceptual da BD e de uma única linguagem de acesso, a linguagem SQL. A secção apresenta sumariamente a noção de BD, a arquitectura actual de uma BD e do respectivo SGBD e termina com alguns exemplos de organização de sistemas comerciais Microsoft e Oracle.

4.2.1 Exemplo de uma Base de Dados muito simples

As primeiras bases de dados informáticas resultaram da passagem dos arquivos documentais mantidos em armários e gavetas para ficheiros com formatos electrónicos, mantidos nos discos ou outras unidades periféricas de computadores. Tipicamente os dados a guardar eram inicialmente pouco variados, incluindo por exemplo nomes, datas e números. Da mesma forma o tipo de operações era inicialmente pouco variado, começando pela adição ou correcção de informação, e pela produção de alguns relatórios simples. Imaginem-se por exemplo as operações de gestão de informação ao nível do ficheiro de livros de uma biblioteca ou da gestão de contas bancárias, antes e imediatamente após a introdução da informática. A evolução da informática ao nível dos equipamentos e das aplicações levou a uma especialização crescente dos sistemas destinados ao tratamento de informação, ou seja dos SGBD, e assistiu-se ao aparecimento de modelos e sistemas cada vez com características mais interessantes. Este desenvolvimento de modelos iniciou-se com o modelo hierárquico, que deu lugar ao modelo em rede, hoje em dia interessantes apenas do ponto de vista histórico. O desenvolvimento teórico culminou com o estabelecimento do modelo relacional já referido. A noção de uma base de dados relacional é complexa, pressupondo sempre a utilização de um SGBD relacional. A noção será apresentada gradualmente.

Apresenta-se de seguida um exemplo de uma base de dados contendo apenas um ficheiro ou uma tabela. Um problema tão simples não se consideraria hoje em dia uma base de dados a gerir por um SGBD, mas de momento irá supor-se que sim, para ilustrar alguns conceitos fundamentais.

Exemplo 4.1: A adega WiNet:

> "A adega WiNet armazena e vende vinhos de diversas origens. Para apoiar as suas actividades comerciais a WiNet utiliza um sistema de informação sobre vinhos. Parte dos dados são mantidos num ficheiro e estão ilustrados na Figura 4.1.
>
> Do ponto de vista do armazém, os vinhos são sempre guardados em caixas colocadas em paletas, guardadas em prateleiras de estantes. Cada estante é identificada por duas letras, cada prateleira por um número entre 0 e 99, e cada prateleira tem 10 locais únicos (**0** a **9**) para colocação de paletas, dando origem a identificadores de local, por exemplo "AB50**1**".

Adega

#Caixa	*nomeVinho*	*nomeCurtoProdutor*	*ano*	*nGarrafas*	*dLitros*	*corETipo*	*prazo*	*#Estante*
23	Foral Grande Escolha	Caves Aliança	1997	6	75	Maduro Tinto	2020	KG033
24	Charamba	Quinta da Aveleda	1999	6	75	Maduro Branco	2015	LO934
25	Quinta da Leda	Ferreira	1995	6	75	Maduro Tinto	2011	JU098
155	Borges	Vinhos Borges	1995	12	75	Maduro Tinto	2011	JU099
26	Coroa Douro	Vinhos Poças Júnior	1996	6	75	Maduro Tinto	2012	FU100
33	Muralhas de Monção	Adega Coop. de Monção	1999	9	75	Verde Branco	2015	AB501
44	Soalheiro	António Esteves Ferreira	1998	3	75	Verde Branco	2014	AC102
45	Quinta de Alderiz	Casa Pinheiro	1998	6	75	Verde Branco	2014	JV103
46	Quinta das Caldas	Quinta da Gaivosa	1996	3	75	Maduro Tinto	1020	JB344
27	Dona Paterna	Carlos Alberto Codesso	1997	6	75	Branco	2013	JG105
29	Redoma	Niepoort Vinhos	1999	1	75	Rosé	2015	CC206
444	Dorna Velha	Quinta do Silval	1995	1	75	Maduro Tinto	2011	RT107
456	Tuella	Cockburn's	1999	3	75	Branco	2015	RT158
78	Vinha dos Freires	Quinta do Valado	1997	6	75	Maduro Tinto	2016	RF009
56	Casa de Vila Boa	Maria Vitória Lencastre	1999	6	75	Verde Tinto	2015	ML110
77	Malvasia	Barbeito	1901	12	75	Aloirado Madeira	2200	AZ001
55	Morgadio da Torre	Sogrape	1999	6	75	Verde Branco	2018	GA001

Figura 4.1 Base de Dados da empresa WiNet: Tabela **Adega** *(o duplo sublinhado no atributo* #Caixa *significa que se trata da chave primária da relação ou tabela[2]).*

Tipos de operações e utilizadores

O **utilizador normal** da base de dados da empresa WiNet precisa de **consultar** informação na base de dados, bem como de **actualizar** informação, ou seja, introduzir, alterar ou apagar dados. Estes são os dois tipos de operações mais frequentes sobre uma base de dados, sendo utilizados de seguida os sinais de pontuação "?" e "!" para as representar.

1. ***Consultar** (?) informação: pesquisar ou interrogar a base de dados ou as suas tabelas*. Por exemplo: obter resposta a solicitações de clientes, fornecedores ou de outros departamentos da empresa sobre a existência e localização de um dado vinho, ou sobre o seu volume de vendas mensal. Como resultado de operações de interrogação é normalmente necessário produzir relatórios com formatos apropriados. Formatos que podem ser requeridos pelas interfaces com outros sistemas informáticos, designados por formatos informáticos. Para além disso, se os relatórios se destinarem a ser utilizados por pessoas, podem ser exigidas

[2] Este conceito, chave primária, é introduzido na secção 4.6.4.

também formatações gráficas específicas, por exemplo para a produção de etiquetas para cartas, ou para elaboração de relatórios mensais de gestão.

2. ***Actualizar*** *(!) informação: adicionar, alterar ou eliminar elementos ou conjuntos de informação na base de dados ou nas suas tabelas.* Por exemplo: introduzir os dados de uma nova caixa de vinho; neste caso será necessário acrescentar novas linhas e novos valores na tabela indicada.

Para permitir o acesso pelos utilizadores normais, alguém teve de definir a estrutura da base de dados e alguém teve de a construir. Irá designar-se esse alguém como o **projectista**. No caso anterior decidiu-se por exemplo que a empresa precisava de uma tabela **Adega** com 9 colunas, cada uma com os nomes indicados e possibilitando gerir informação relativa a cerca de 10.000 caixas de vinho. Pode também concluir-se que há outra informação importante a gerir que não se pode guardar na tabela **Adega**. Nesse caso o construtor terá por exemplo de alterar a estrutura da tabela **Adega** ou acrescentar novas tabelas à base de dados. Este é mais um outro tipo de operação que o SGBD deve suportar:

3. ***Actualizar estrutura****: definir a base de dados e suas tabelas, acrescentar novos tipos de informação à base de dados, alterar os tipos ou a estrutura existente, juntar ou partir tabelas, ou mesmo eliminar tipos de informação, definir restrições de integridade e outras regras*. Por exemplo: adicionar preços de aquisição, preços de venda ou contactos dos produtores. Para tal pode ser necessário adicionar novas colunas a tabelas existentes, ou mesmo acrescentar novas tabelas à base de dados. Pode também ser necessário alterar a estrutura das tabelas existentes, mudando colunas entre tabelas, o que exige uma grande atenção ao significado e relações entre tipos de informação, sobretudo se a base de dados já estiver em operação.

Finalmente os utilizadores normais esperam que a informação na base de dados esteja acessível de uma forma organizada e que as operações sejam rápidas. Por exemplo é mais fácil localizar um vinho a partir do seu nome se o ficheiro **Adega** for mantido ordenado pelo nome do vinho. É da responsabilidade do **administrador** ou gestor da base de dados garantir que ela funciona bem, optimizar todo o seu funcionamento, bem como assegurar que as regras empresariais de acesso sejam respeitadas.

4. ***Gerir*** *a base de dados: optimizar o acesso à informação, garantir a segurança das operações, manter a integridade, e em geral proteger a base de dados*. Por exemplo: manter a base de dados ordenada e garantir respostas rápidas para as consultas mais frequentes; classificar os utilizadores normais, atribuir-lhes senhas conforme os níveis de acesso à informação, e proteger as tabelas de acordo com essas permissões; garantir a realização de cópias de segurança e garantir o registo de todas as operações efectuadas para possibilitar a recuperação do estado actual.

Tipos de utilizadores e utilizações de um SGBD

Utilizadores	Tipos de Operações	Exemplos
Utilizador (normal)	Gestão da informação	Actualizar dados, Consultar dados
Projectista	Gestão da estrutura	Definir tabelas, Definir gestão da informação, Actualizar estrutura
Administrador	Gestão da base de dados	Implementar, Optimizar Proteger

A linguagem SQL

Na Figura seguinte apresentam-se alguns exemplos de operações dos utilizadores normais sobre a base de dados da WiNet, identificada de momento apenas com a tabela **Adega.** Estas operações estão escritas em SQL, linguagem que será apresentada em detalhe mais tarde. Operações realizadas pelos construtores e administradores serão referidas noutra ocasião.

Operação de consulta e respectivo resultado em forma de tabela:

```
SELECT   nomeVinho, #Caixa, ano
FROM     Adega
WHERE    ano = 1995 ;
```

Resultado:

nomeVinho	*#Caixa*	*ano*
Quinta da Leda	25	1995
Borges	155	1995
Dorna Velha	444	1995

Operação de introdução de novos valores:

```
INSERT
INTO     Adega (#Caixa, nomeVinho, nomeCurtoProdutor, ano, nGarrafas,
               dLitros, corETipo, prazo, #Estante)
VALUES         (404, 'Caves Velhas', 'Sograpе', 1998, 12,
               75, 'Maduro Tinto', 2015, 'DF111') ;
```

Operação de alteração de dados:

```
UPDATE   Adega
SET      Local = 'JAG15'
WHERE    Caixa = 444 ;
```

Operação de eliminação de uma linha de dados:

```
DELETE
FROM     Adega
WHERE    Caixa = 444 ;
```

Figura 4.2 Exemplos de operações sobre a base de dados WiNet com a linguagem SQL.

A linguagem SQL é relativamente simples e permite definir todas as operações de interface com um SGBD relacional, ao nível do utilizador, do projectista e do administrador do SGBD, embora apresente na prática algumas limitações. Basta de momento referir que existem normas internacionais de SQL, normas essas que são suportadas parcialmente e com algumas variações por quase todos os fabricantes, dando origem a vários dialectos de SQL. Na secção 4.4 será feita uma apresentação um pouco mais elaborada de SQL.

Generalizando um pouco mais, uma Base de Dados é inicialmente uma versão informática de um conjunto de ficheiros, relacionados entre si, destinando-se assim a apoiar toda a gestão de informação de uma instituição e dos seus utilizadores. Um Sistema de Gestão de Bases de Dados, ou simplesmente SGBD, é um conjunto de suportes lógicos que permitem construir e manter Bases de Dados de uma forma integrada. Muitas vezes utilizam-se os termos Base de Dados e SGBD como sinónimos, visto que não é possível manter uma Base de Dados em funcionamento independentemente de um SGBD, normalmente o que lhe deu origem. Assim é em particular para as bases de dados de modelos complexos, tais como as relacionais. A noção de base de dados está actualmente associada a um sistema complexo, devido ao número de ficheiros que o constituem, ao número muito elevado de dados que mantém, ao número de utilizadores, ao número de funções que suporta e ao facto de ser um sistema que teve o início de operação num determinado momento, mas que não tem um fim previsto.

Na secção seguinte apresenta-se um conjunto de arquitecturas normalizadas para Bases de Dados seguindo as propostas em [Date 1995].

4.2.2 Modelos e Arquitecturas Genéricas para SGBD

Esta secção introduz o modelo relacional para um SGBD, o que o caracteriza e distingue de outros sistemas de gestão de informação, em particular de sistemas de gestão orientados a ficheiros e sistemas de gestão orientados a objectos. Apresenta ainda a arquitectura genérica de um SGBD, de acordo com as definições ANSI-SPARC, e segundo o conceito cliente-servidor. Esta secção termina com a apresentação do conceitos de *vista* e do conceito de *transacção*, conceitos fundamentais no projecto de qualquer base de dados.

Modelos Relacional, Orientado por Objectos e Objecto-Relacional

Os sistemas de gestão de dados, e em particular os SGBD, podem ser classificados de acordo com duas propriedades muito importantes dos problemas de gestão de informação que se destinam a apoiar [Stonebraker & Moore 1998]:

- Complexidade de dados que se destinam a gerir.
- Necessidade de funcionalidades de interrogação flexível.

Os SGBD baseados no modelo relacional são muito limitados no tipo de dados que permitem gerir de uma forma bem suportada (ver secção 4.4), mas para os tipos de dados que suportam oferecem facilidades de consulta muito flexíveis, em particular através da norma SQL.

Os SGBD resultantes da evolução das linguagens de programação orientadas por objectos, permitem gerir tipos muito ricos de dados, mas como não existem normas aceites universalmente do tipo da liguagem SQL, tornam-se difíceis de interrogar [Stonebraker & Moore 1998]. Esta limitação torna também complicada a sua participação em sistemas distribuídos, devido à inexistência de um modelo de dados simultaneamente simples e

normalizado. Existe uma proposta de linguagem OQL ("Object Query Language") para este tipo de modelos, mas não parece ter ainda aceitação industrial suficientemente forte.

Os SGBD mistos, resultantes da adição de características de orientação por objectos ao modelo relacional, tentam estabelecer um compromisso entre a manutenção de uma visão simples e uniforme da informação baseada em tabelas, a manutenção de uma linguagem de interrogação normalizada, e o suporte integrado a tipos de dados mais complexos requeridos por aplicações de CAD ("Computer Aided Design"), GIS ("Geographical Information System") ou SAD ("Sistemas de Apoio à Decisão"[3]). Este compromisso passa actualmente pelo estabelecimento da norma SQL-3, promovida pela Oracle e IBM.

Desta forma obtém-se a matriz da Figura 4.5, onde se indicam exemplos de aplicações, ferramentas e mercados relativos esperados para os quatro tipos de sistemas que melhor se adequam aos problemas com as características referidas. Esta matriz será justificada mais adiante e em pormenor na secção 4.7.

Quadrante / *Modelo* / *Exemplos* / *Mercado relativo SGBD 2005*	*Dados Simples*	*Dados Complexos*
Sem Interrogação:	1 Sistemas de Ficheiros UNIX, Word, vi, Emacs, Excel, Powerpoint *n/a*	3 **SGBDoo** CAD, ECAD, O2, Versant **1**
Com Interrogação:	2 **SGBDr** SQL-2, ORACLE, DB2, Access, SQL Server **100**	4 **SGBDor** SQL-3, Odaptor, UniSQL, Ilustra **150**

Figura 4.3 Modelos de SGBD: Matriz de Classificação para Arquivos de Dados cf. [Stonebraker & Moore 1998].

Seja qual for o modelo em que se baseia, um SGBD será em princípio tanto melhor quanto melhores forem as suas características técnicas. No entanto, a adopção de um SGBD por uma empresa não depende apenas das suas propriedades técnicas, mas sobretudo da avaliação da relação global custo versus benefício. Factores como o preço do produto, a estabilidade do fabricante ou a quota de mercado para os seus produtos, e a adequação dos recursos humanos existentes na própria empresa, têm normalmente um peso importante na decisão.

Do ponto de vista do conjunto de utilizadores ou de uma empresa podem-se apresentar as seguintes solicitações técnicas a um SGBD e às respectivas Bases de Dados:

[3] Este é o termo mais correcto em Portugal: SAD, "Sistema de Apoio à Decisão", tradução de DSS, "Decision Support System". No Brasil o termo mais utilizado é SSD, "Sistema de Suporte à Decisão".

- Deve poder prestar um serviço permanente, 24 horas por dia, 7 dias por semana, assegurando a persistência da informação e dos dados[4], garantindo níveis de segurança face a acessos indevidos e facilidades para recuperação face a erros de utilização.
- Deve permitir a integração de informação de várias origens, possibilitando a sua partilha por vários utilizadores, permitindo a execução de operações de consulta e actualização em paralelo, mas garantindo a consistência e integridade.
- Deve poder interligar-se ao nível dos dados com qualquer outro sistema, por exemplo através de um portal (gateway), e ao nível do desempenho deve ter interfaces para qualquer monitor de transacções.
- Deve permitir a execução em qualquer plataforma de equipamento ou de sistema operativo, incluindo multiprocessadores de memória partilhada, garantindo assim a independência de dados.
- Deve funcionar bem com bases de dados muito grandes (VLDB), suportando espaços de endereçamento superiores aos disponíveis directamente pela arquitectura particular dos processadores utilizados.
- Deve permitir a distribuição da base de dados por vários locais interligados em rede, assegurando de uma forma transparente e eficiente a realização de processos de replicação de dados, caso seja necessário.

Os fabricantes de SGBD têm não só de assegurar estas propriedades, como ainda garantir processos de evolução, manutenção e apoio excelentes, visto que os mercados para estes produtos são actualmente muito competitivos.

Os principais produtos comerciais actualmente existentes no mercado baseiam-se no modelo relacional[5]. Entre os mais conhecido referem-se os sistemas Oracle (da Oracle), DB2 da IBM, Access, SQL Server e Sybase, todos da Microsoft, Ingres da Computer Associates, Progress e 4D. Este último sistema é muito utilizado sobretudo em ambientes Macintosh. Todos estes produtos oferecem funcionalidades semelhantes, e todos eles se aproximam do modelo relacional, procurando seguir normas aceites internacionalmente para tal efeito. Todos eles

---

[4] Os dados dizem-se persistentes se, independentemente do suporte físico, se mantiverem inalterados enquanto não forem modificados na sequência de uma acção independente ou voluntária. O conceito oposto, de dados temporários, está por exemplo associado às estruturas criadas por um programa e mantidas apenas durante a sua execução. Alguns autores defendem que nunca se deveriam eliminar dados de um sistema, apenas actualizar a sua veracidade. Existem certas disposições legais relativamente a alguns tipos de dados que não permitem a sua eliminação ou modificação, como é o caso dos assentos de nascimento, das escrituras públicas, ou dos dados recebidos do espaço a partir de satélites.

[5] [Stonebraker & Moore 1996]: “Despite the crowded playing field, the relational DBMS vendors are generally very healthy companies. Relational DBMS customers have seemingly insatiable need for additional licenses, add-on products, and consulting services. Collectively, the relational DBMS market is approximately $8 billion per year and growing at more than 25% per year. The only storm cloud on the horizon is the eminent arrival of Microsoft into this market. They have acquired the Sybase code line for Microsoft Windows NT and have a substantial group in Redmond improving it. Many would argue that Microsoft SQL server is now a superior product on Microsoft Windows NT. Besides having a good product, Microsoft is committed to “PC pricing” for database systems. They are undercutting the pricing structure of the traditional vendors and generally causing relational DBMS license process to fall. Look for continued erosion of pricing in this market”.

têm características de desempenho diferentes em situações particulares. Por exemplo o Access e o 4D estão vocacionados para um processo de prototipificação evolutiva rápido e para poucos utilizadores simultâneos, enquanto que o SQL Server, Oracle e DB2 podem suportar centenas ou milhares de transacções simultâneas, mas pressupõem um processo de desenvolvimento mais estruturado à partida e uma evolução de requisitos mais lenta.

Dos Sistemas Documentais às Bases de Dados e aos SGBD

O Exemplo 4.1 é talvez demasiado simples para se perceberem claramente as vantagens de se utilizar uma Base de Dados ou um SGBD, em vez de um arquivo documental tradicional. Para uma adega pequena com poucos movimentos seria possível manter toda a informação em papel, ou poder-se-ia recorrer a um ficheiro de texto ou a uma simples folha de cálculo, suportados em aplicações simples de escritório electrónico, evitando assim por exemplo o recurso a um SGBD e à necessidade de dominar o SQL.

Tem normalmente de haver boas razões para se passar de uma solução documental para uma solução informática, e também para passar de uma solução informática simples com ficheiros isolados para uma solução que recorra a uma base de dados e a um SGBD.

No entanto se se imaginar uma adega relativamente grande, com muitos milhares de caixas de vinho e com muitas centenas ou milhares de clientes e fornecedores, numa situação em que existem muitos utilizadores simultâneos do sistema, é fácil perceber as vantagens de recorrer a um sistema mais elaborado de apoio à gestão de dados, ou seja a um sistema informatizado. Um tal sistema deveria garantir, relativamente a uma solução baseada em fichas ou documentos em papel, o seguinte:

1. *Compactação dos suportes*: O espaço físico necessário deveria ser menor em suportes magnéticos do que em papel.
2. *Velocidade*: Os tempos de actualização e consulta deveriam ser menores.
3. *Actualidade*: Toda a informação deve estar sempre disponível para qualquer utilizador e deve estar sempre actualizada.
4. *Satisfação*: Os utilizadores deveriam sentir-se melhor ao evitarem tarefas repetitivas e os erros desse tipo de processos manuais deveriam ser eliminados.

Embora estas propriedades sejam importantes, elas podem aplicar-se também a um sistema de informação baseado no armazenamento de dados num conjunto de ficheiros, suportados por aplicações simples de escritório electrónico. Um sistema de gestão de dados pode e deve garantir algo mais. Relativamente a uma solução baseada em ficheiros, o seguinte é essencial:

1. *Unilocalização de Dados*: Sendo o contrário de redundância de dados, refere-se à propriedade de uma colecção de dados, em que todo e qualquer dado particular se pode e deve encontrar exclusivamente num único local. Por exemplo, uma instituição com dois ficheiros, um de clientes e outro de funcionários, provavelmente irá ter redundância de dados, por exemplo ao nível dos nomes, se permitir que os funcionários também sejam clientes.
2. *Consistência de dados*: Sendo o contrário de inconsistência, corresponde a uma propriedade lógica que significa que não pode haver simultaneamente na base de dados um facto e a sua negação, ou que ambos não podem ser logicamente

deduzidos simultaneamente. Por exemplo, uma base de dados sobre produtos farmacêuticos não pode manter a informação de que o lote 345 da vacina HB tem a data de fabrico de 1998.06.18 e simultaneamente que o mesmo lote tem a data de fabrico de 2002.06.18 (que logicamente significa que o lote 345 da vacina HB não tem a data de fabrico de 1998.06.18). A unilocalização de dados facilita a manutenção da sua consistência.

3. *Integridade de dados*: Corresponde à satisfação pela base de dados de um conjunto de regras para além das regras lógicas, e que normalmente dependem da aplicação ou situação para a qual se está a utilizar a base de dados. Por exemplo, uma mesma base de dados sobre produtos farmacêuticos não pode manter a informação de que o lote 345 da vacina HB tem a data de fabrico de 1998.06.18 e a informação de que o mesmo lote foi vendido em 1997.05.01, se existir uma regra que indique que não se pode vender algo antes do seu fabrico[6].

4. *Independência de dados*: Num contexto informático, corresponde à propriedade que, se for satisfeita, permite a aplicações externas à base de dados acederem aos seus dados sempre da mesma forma, seja qual for a tecnologia, sistema operativo, arquitectura, ou linguagem de programação que é utilizada para a suportar na realidade. A satisfação desta propriedade depende em grande medida na existência de um modelo abstracto dos dados, suportado por uma linguagem normalizada de interface, suportada pelos fabricantes de SGBD nas múltiplas plataformas. As bases de dados ou sistemas persistentes anteriores ao modelo relacional, bem como algumas bases de dados baseadas em modelos novos não satisfazem esta propriedade. Normalmente devido à falta de um modelo abstracto, suficientemente simples, divulgado e suportado pelos fabricantes e utilizadores. No caso dos sistemas de bases de dados pré-relacionais, a falta de tal modelo era agravada pela utilização de estruturas de armazenamento físico optimizadas para determinados fins, devido às limitações de memória e capacidade de processamento existentes, e que tinham necessariamente de ser conhecidas por qualquer programa de acesso.

Um SGBD deverá satisfazer tecnicamente todas as anteriores características. Para além disso, só deverá ser utilizado no caso de haver vantagens claras com a sua utilização, quer do ponto de vista técnico, quer do ponto de vista empresarial. Os SGBD de modelo relacional são os que hoje em dia melhor cumprem todas as propriedades anteriormente apresentadas, bem como simultaneamente satisfazem os critérios empresarias na relação custo-benefício.

No caso do exemplo seguinte ou no caso apresentado no Anexo A, apenas um sistema baseado num SGBD poderia responder às necessidades da instituição.

[6] À primeira vista poderia parecer que, devido a uma regra de lógica temporal, estes dois factos geravam uma inconsistência. Deve observar-se que primeiro seria necessário relacionar o significado dos verbos *fabricar* e *vender*, bem como considerar a regra, não trivial ou lógica, de que *não se pode vender algo que ainda não se fabricou*. De facto esta regra, a existir, é um exemplo de uma *regra de negócio* ou *restrição de integridade sobre a informação* nas tabelas da base de dados.

Exemplo 4.2: O Sistema de Informação integrado para a empresa WiNet:

> "O grupo WiNet resultou de um processo de aquisições e fusões de um conjunto de empresas de armazenamento, revenda e retalho de vinhos, liderada pelos sócios da adega WiNet. O processo de reengenharia por que as várias empresas passaram, originou um conjunto de requisitos para o sistema de informação de apoio às operações e gestão da WiNet.
>
> A estratégia de posicionamento do grupo no mercado, as suas necessidades de informação e a oferta informática existente, contribuíram para se adoptar uma solução baseada numa aplicação parametrizável existente no mercado, assente em tecnologia SGBDr, a que seria necessário acrescentar um conjunto de módulos e funcionalidades específicas ao sector da comercialização de vinho, de acordo com os factores críticos de sucesso identificados pela WiNet.
>
> Assim, tornou-se por exemplo necessário integrar toda a informação relativa a clientes e fornecedores do grupo, aos funcionários das várias empresas e dos vários departamentos, mantendo permanentemente actualizada a estrutura da organização, os orçamentos, custos e receitas das várias unidades. Foi também necessário manter actualizadas as existências de vinhos e produtos similares, bem como a sua localização nos vários armazéns e lojas. Introduziu-se também um módulo para permitir aproximar os funcionários e colaboradores da WiNet com os clientes existentes e os clientes potenciais, melhorando o conhecimento disponível no sistema, por exemplo integrando dados sobre as especialidades dos funcionários, os interesses dos clientes e os produtos locais".

SGBD relacionais e Bases de Dados Relacionais

Do ponto de vista prático, e numa primeira abordagem, uma base de dados diz-se relacional se satisfizer simultaneamente as seguintes 3 propriedades:

1. *Tabulação bidimensional da informação*: Toda a informação é mantida em tabelas bidimensionais, com um número determinado, estável e normalmente muito pequeno de colunas e com um número à partida ilimitado, variável e normalmente muito grande de linhas.
2. *Valores de informação atómicos*: Toda a informação guardada em cada célula é atómica, não podendo haver em cada célula vários valores; isto é, na intersecção de uma linha e de uma coluna de qualquer tabela da base de dados, não pode haver listas, grupos ou conjuntos de diversos valores, valores estes com significado individual.
3. *Informação não repetida*: Cada linha de cada tabela é única, isto é, não há linhas com todos os valores repetidos nas tabelas de uma base de dados relacional.

Uma base de dados em que todas as tabelas respeitam estas características diz-se que está em **primeira forma normal** (ver secção 4.5.1). Por exemplo, a base de dados WiNet, apenas com a tabela da Figura 4.1, é uma base de dados relacional, de acordo com esta definição. Já a base de dados contendo a tabela da Figura 4.4 não é relacional, visto que as 3 propriedades anteriores não se verificam:

1. O número de colunas não está limitado à partida: a tabela tem um número indeterminado de colunas, com nomes *cliente1*, *cliente2*, …, *clienteN*.

2. Existem células com valores múltiplos; por exemplo, na 6ª linha, 1º campo são referidos num só campo nomes de 2 vinhos: "Dão, Duque de Viseu". Cada nome tem significado, identificando um vinho particular. Como se verá, um processo de pesquisa ou consulta eficiente não seria em geral possível se se permitissem campos com valores múltiplos[7].
3. A tabela tem as duas últimas linhas iguais, que não se podem distinguir apenas com a informação apresentada. Como os valores são precisamente iguais, não há forma de os distinguir (tal como um conjunto não pode conter dois elementos de igual valor, uma tabela também não pode conter duas linhas iguais, pois representam exactamente o mesmo objecto).

Vinhos {esta tabela não pertence a uma base de dados relacional}

nomesDosVinhos	*produtor*	*cliente1*	*cliente2*	…	*clienteN*	*tipo*	*local*
Reguengos	Coop. Ag. Reguengos	A. Santos	C. Matos		Z. Silva	Tinto	AAG
Quinta do Carmo, S. Martinho	S. Ag. Quinta do Carmo	J. J. Cunha	A. Silva		P. Matos	Tinto	ABS
Alvarinho 1995 Soalheiro	A. Esteves Ferreira	J. Lino	G. Santos		H. Nóvoa	Tinto	CVR
Quinta de Simaens	Borges	R. Campos	J. Cabral		J. Sousa	Tinto	TCC
Mateus Rosé	Sogrape	D. D. Vida	R. Silva			Tinto	DSD
Dão Duque de Viseu, Bairrada	Sogrape	S. Resende	A. Silva		A. Silva	Tinto	PQR
Esteva	Casa Ferreirinha	L. Matos	J. Moreira		T. Glavão	Tinto	PQR
Esteva	Casa Ferreirinha	L. Matos	J. Moreira		T. Glavão	Tinto	PQR

*Figura 4.4 Exemplo de tabela de uma base de dados **não relacional***

Um SGBD relacional é um conjunto de suportes lógicos que permitem construir e manter bases de dados relacionais de uma forma consistente, integrada e independente.

Arquitectura ANSI-SPARC

Uma definição mais elaborada para um SGBD pode ser dada recorrendo à arquitectura ANSI-SPARC (Figura 4.5) e à arquitectura cliente-servidor (Figura 4.6, Figura 4.8 e Figura 4.9).

Os níveis interno, conceptual e externo apresentados na Figura 4.5 correspondem respectivamente aos detalhes de implementação num dado equipamento, ao esquema completo mas abstracto de toda a informação mantida na base de dados, e aos vários esquemas conceptuais a que cada utilizador dos dados tem acesso.

O **nível externo** corresponde ao que o utilizador normal em consulta ou actualização de dados vê da base de dados, ou ao que o programador de uma determinada interface com o utilizador ou com outro sistema externo pode ver da base de dados. Sendo assim, as linguagens de interface do utilizador normal ao nível externo tanto podem ser os formulários e relatórios de

[7] De facto o modelo relacional não impõe esta condição, mas na prática todas as implementações actuais do modelo relacional a exigem, devido ao número muito limitado de tipos de dados suportados. A simplicidade dos tipos de dados é também uma das vantagens dos SGBD relacionais, tendo em atenção a matriz da Figura 4.3. No entanto, e segundo [Date 2000], pp. 112-113 (entre outras páginas), o modelo relacional permite atributos cujos valores são relações (e imagens ou sons), desde que existam operadores para gerir valores desses tipos. O próprio autor reconhece que afirmações anteriores, indicando que os domínios contêm somente valores atómicos, estão erradas. Esta abertura do modelo relacional a valores menos simples, abre a possibilidade de suporte à gestão de objectos de uma forma evolutiva, originando o designado modelo objecto-relacional.

uma interface previamente desenvolvida sobre as bases de dados, como podem ser pra aum programador de aplicações as linguagens C++, Java ou VisualBasic, incluindo comandos de uma sub-linguagem de dados como a linguagem SQL. Normalmente para facilitar a construção de aplicações externas, típicas os SGBD oferecem as designadas Linguagens de 4ª Geração, ou 4GL - *4th Generation Language*[8].

As 4GL são actualmente linguagens proprietárias, e incluem SQL nos SGBD de modelo relacional. Permitem a definição interactiva do esquema da base de dados, a construção de consultas interactivamente através de uma linguagem tipo QBE – *Query By Example* (ver por exemplo [Date 2000], pp. 205), e a definição gráfica de formulários e relatórios baseados em consultas ou tabelas, bem como de alguns tipos simples de procedimentos associados a estes objectos.

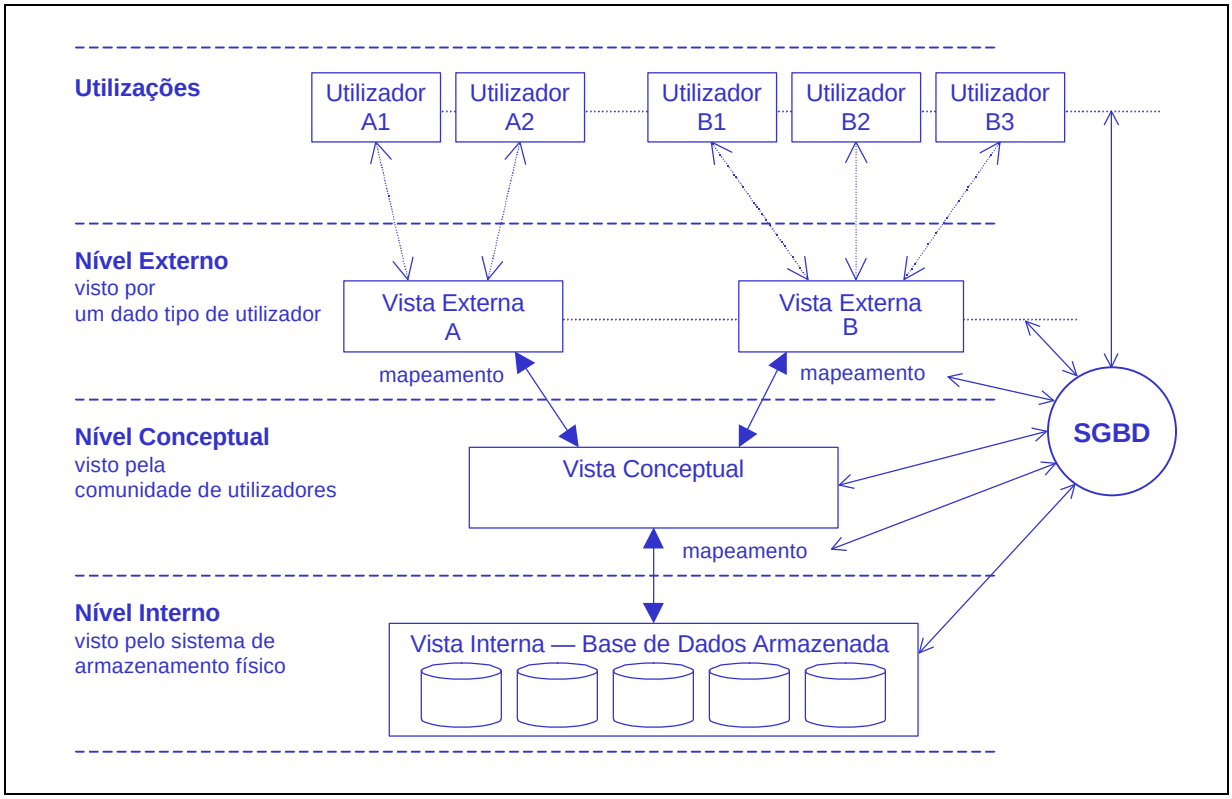


Figura 4.5 Arquitectura ANSI-SPARC para um SGBD relacional

O **nível conceptual** corresponde à visão global de toda a informação contida na base de dados, ou seja, à visão conjunta que permitirá obter toda e qualquer vista externa possível. Apesar de completa, esta vistão conceptual não pode nem deve incluir qualquer pormenor ou detalhe relativo à forma de implementação da base de dados ou da codificação da informação

[8] As linguagens 4GL permitem desenvolver aplicações de forma mais expedita do que as outras linguagens introduzidas anteriormente, em termos históricos. De acordo com esta classificação, em *gerações*, as linguagens de primeira geração seriam os códigos de programação máquina, as linguagens de segunda geração seriam as linguagens *assembly*, e as linguagens de terceira geração seriam as linguagens de mais alto nível, tais como Fortran, Cobol, Pascal, C, C++ ou Java, que embora totalmente flexíveis consomem mais recursos, nomeadamente tempo, para construir aplicações totalmente funcionais.

numa determinada plataforma informática, seja SGBD comercial, sistema operativo ou equipamento físico.

O **nível interno** tem a ver com as opções de implementação. As operações de optimização do desempenho da base de dados, e dos restantes requisitos não funcionais, requerem que um utilizador apropriado (gestor) saiba o suficiente sobre o nível interno para poder *afinar* o desempenho da base de dados, ajustando os parâmetros de configuração relacionados com a gestão do nível interno. A possibilidade de transportar uma base de dados para equipamentos de fabricantes diferentes, com sistemas operativos distintos, ou com arquitecturas diferentes, depende da independência entre o nível conceptual e o interno, e da forma como o SGBD gere e optimiza as traduções necessárias entre o nível conceptual e o nível interno da arquitectura específica de cada sistema operativo e equipamento.

Arquitecturas cliente-servidor

A partir do momento em que se difundiram as bases de dados e as aplicações que funcionando sobre essas bases de dados respondiam aos vários requisitos dos utilizadores, assistiu-se à separação física e lógica entre os programas que geriam o acesso aos dados e os programas que respondiam às necessidades dos utilizadores. Esta separação resultou de compromissos entre fiabilidade, segurança e desempenho, e acentuou a função da linguagem de interface SQL. As funções de gestão de acesso aos dados são asseguradas pelo **servidor** da base de dados e as funções de resposta aos utilizadores são asseguradas pelos **clientes**. O SGBD cumpre funções de gestão no servidor e nos clientes.

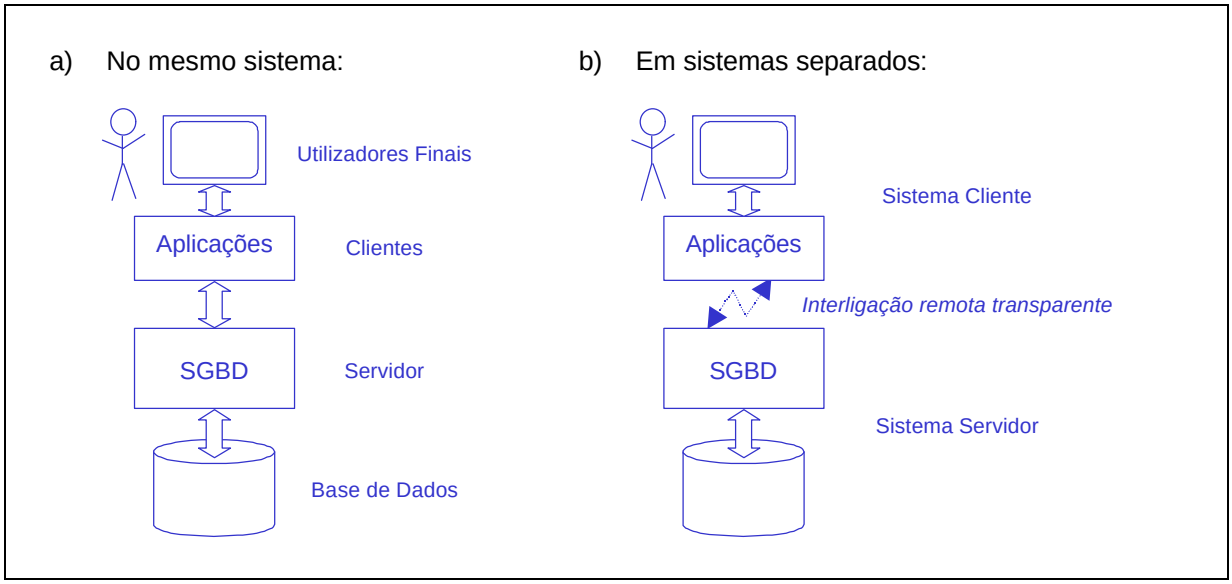


Figura 4.6 Arquitecturas cliente-servidor – o SGBD também gere uma parte da funcionalidade dos clientes, em particular garantindo uma interface cliente-servidor tipicamente em SQL

Do ponto de vista de segurança, os processos que asseguram a gestão do servidor da base de dados executam em espaços de endereçamento protegidos, são processos muito bem testados e com resistência a falhas extraordinariamente elevada. Por exemplo, considere-se o caso de uma base de dados com muitas aplicações clientes e com muitas transacções. Não seria aceitável que um problema de execução de um destes programas clientes comprometesse o funcionamento ou desempenho do servidor da base de dados. Normalmente será também

desejável que na medida do possível as aplicações clientes executem em processadores separados dos processadores do servidor, diminuindo a sobrecarga sobre este. Esta distribuição de processamento deve obedecer a uma análise cuidada das funcionalidades desejadas, e tem de considerar sempre requisitos não funcionais relacionados com os tempos de resposta aceitáveis e com as capacidades de investimento em sistemas informáticos (do lado dos servidores e dos clientes).

```
<%@LANGUAGE = "VBScript"%>
<%

'VARIÁVEIS GLOBAIS

'VARIÁVEIS LOCAIS
dim strDB_DSN     'Variável para conexão à base de dados
dim strSQL        'Resultado de instrução SQL
dim oConn

set oConn = server.createobject("ADODB.connection")
                  'inicialização de variável de conexão

strDSN = "DSN=BDados;UID='';PWD=''"
                  'BDados é a conexão ODBC definida

oConn.open strDSN
                  'temos a conexão ODBC à base de dados

strSQL="SELECT CodAluno,Nome,Cidade,NTelefone,AnoNascimento from Aluno"

set oRS = oConn.execute (strSQL)
                  'oRS passa a conter os valores devolvidos pelo SELECT
                  'efectuado, que se podem aceder da forma oRS("Nome")
oConn.close
%>
```

Figura 4.7 Formato geral de um programa VB para criação de uma conexão cliente-servidor tradicional a uma base de dados

Note-se que mesmo com clientes e servidor executando fisicamente no mesmo equipamento e partilhando o mesmo processador e memória, a separação lógica deve existir por forma a que, por exemplo, um erro numa aplicação cliente programada em C++ não possa “parar” a base de dados. Quando os programas clientes são computacionalmente exigentes, por exemplo em cálculo, a separação do cliente num processador diferente pode ter vantagens claras de desempenho. Nos casos em que o programa cliente não efectua processamento, a vantagem da sua separação não pode ser atribuída ao desempenho, uma vez que é sobre o servidor que recai o esforço de interpretar e processar o acesso aos dados solicitado pelo cliente.

A separação entre cliente e servidor na arquitectura de um SGBD é essencialmente justificada por questões de segurança, e afecta desde logo negativamente o desempenho dos programas

cliente. De facto, esta separação é na prática conseguida pela introdução de uma interface lógica através da qual passam normalmente instruções que têm de ser interpretadas e optimizadas. Em geral estas instruções são em SQL, passando através de uma ligação ODBC (ver Figura 4.7) e a interacção é relativamenta lenta, pois requerem por parte do servidor a interpretação e eventual optimização, ou um mapeamento de estruturas de dados. Um cliente pode ainda colocar um pedido de execução de procedimentos armazenados na base de dados, tratando-se neste caso de uma interacção muito mais rápida, mas que requer à partida um conhecimento quase total do tipo de informação desejada. Em qualquer dos casos, a resposta do servidor é dada normalmente através de uma tabela ou de uma estrutura de dados equivalente, ou através de uma mensagem de que se iria verificar um erro, sem comprometer o funcionamento posterior do servidor da base de dados.

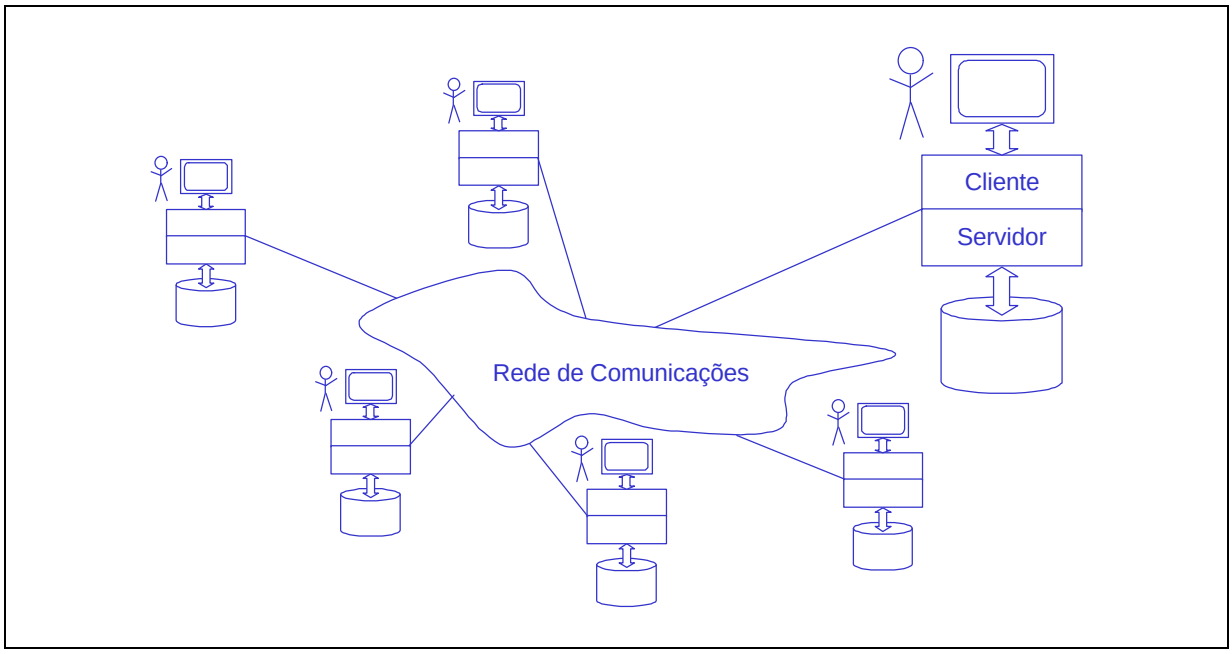


Figura 4.8 Arquitectura cliente-servidor em rede – por exemplo Internet: cada sistema é simultaneamente cliente e servidor

Os SGBD de modelo orientado por objectos não apresentam normalmente esta separação entre processos cliente e servidor. Como vantagem, as respostas às solicitações não requerem o tempo de interpretação e optimização subjacente à interface, ou o mapeamento de estruturas de dados, mas apresentam o risco de que uma consulta menos testada comprometa a execução de futuras respostas. Uma análise detalhada deste problema e das soluções propostas pelos vários fabricantes de SGBD pode ser vista em [Stonebraker & Moore 1996].

Arquitecturas cliente-servidor através da Internet

A maior parte das instituições com necessidade de disponibilizar os seus sistemas de informação em vários locais utilizou no passado redes informáticas privadas, assentes em sistemas operativos e protocolos que permitiam **ligações orientadas à conexão**. Os processos de gestão das ligações eram da responsabilidade do protocolo de comunicação que assegurava a manutenção do canal. Exemplos destas redes são ainda extremamente comuns, incluindo não só as redes privadas das instituições mas igualmente redes de utilização pública como as que suportam a operação das ATM (multibanco).

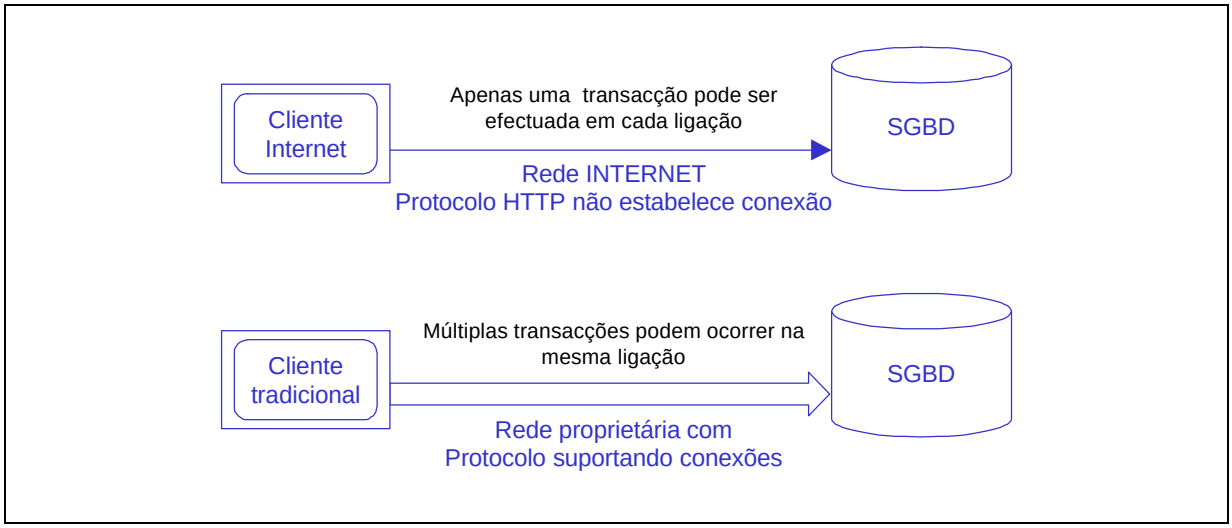


Figura 4.9 Arquitectura cliente-servidor tradicional e através da Internet

Uma ligação orientada à conexão significa que uma vez estabelecida a ligação entre uma aplicação remota e um servidor, as propriedades dessa ligação são mantidas enquanto a ligação não for interrompida por uma das partes. Um exemplo de uma ligação deste tipo é a ligação telefónica tradicional nas redes fixas entre dois assinantes, onde uma chamada depois de estabelecida se mantém activa enquanto pelo menos uma das partes não desliga o seu telefone (mesmo que não haja qualquer conversação). A comunicação na Internet para solicitar uma página remota, pelo contrário, é um exemplo de uma ligação que não é orientada à conexão, pois cada pedido de uma página inicia uma ligação que termina assim que a página é recebida (do lado do sistema remoto esta ligação inicia-se e termina ainda mais depressa). Assim, no caso da Internet cada ligação tem de incluir o identificador do remetente e do destinatário, e pode nem sequer ter memória.

A difusão da Internet, inicialmente utilizada apenas para o acesso a páginas de texto formatadas em HTML (WWW), num único sentido, rapidamente evoluiu para permitir o acesso a informação dinâmica armazenada em bases de dados e para interacção bidireccional (recorrendo ao protocolo ou norma CGI). A difusão extremamente rápida e acessível economicamente da Internet com os seus protocolos e serviços, colocou a questão nas instituições de recorrerem a esta infraestrutura quase ominpresente para distribuirem os seus sistemas de informação, substituindo as tradicionais soluções e redes proprietárias de custo elevado.

No entanto, uma vez que os protocolos base da Internet não são orientados à conexão, a sua utilização directa por sistemas cliente servidor não é possível. Há normalmente duas alternativas para resolver o problema: adicionar uma camada protocolar que garanta o protocolo exigido pelo sistema de informação existente na organização, ou alterar o sistema de informação para este passar a gerir a falta de orientação à conexão na Internet. A primeira solução resolve o problema num ambiente empresarial controlado, em que todos os locais passíveis de utilizar o sistema de informação podem ser dotados da camada protocolar necessária. Este é o caso por exemplo de uma empresa que controla centralmente todos os locais de acesso ao seu sistema, quer estes se encontrem na empresa ou nas empresas que com ela tem um relacionamento preferencial (clientes e fornecedores). Esta solução tem como inconveniente alguma degradação do desempenho, uma vez que a camada protocolar vai

utilizar recursos de banda disponíveis. No entanto, mais grave ainda, esta solução não é viável numa empresa que quer abrir o seu sistema de informação e bases de dados ao público em geral, por exemplo para actividades de comércio electrónico de produtos ou serviços. Assim sendo, o sistema de informação de uma livraria ou de um banco, que deve ser visto e utilizado pelos clientes ou fornecedores em geral, deve ser construído ou adaptado aos protocolos da Internet.

Esta adaptação requer a gestão do estado nas ligações, uma vez que cada ligação nova requer saber o estado da conexão. Podem utilizar-se várias técnicas para memorizar o estado, desde variáveis permitidas pela norma HTML até instalação de objectos memorizadores de ligações nos computadores dos clientes (os designados “cookies”). Uma discussão detalhada deste tópico é abordada por exemplo em [Ju 1997].

Vistas sobre uma Bases de Dados

Um dos mecanismos mais interessantes que devem ser suportados por um SGBD e em particular pela linguagem de interface, ou seja pelo SQL, é a possibilidade de definição e gestão de vistas sobre a Base de Dados. Uma vista corresponde a uma operação de selecção de informação à qual é associado um nome, e que o SGBD deve manter como uma relação virtual.

Entre as vantagens de utilizar vistas salientam-se as seguintes [Date 1997]:

- As vistas fornecem um recurso de abreviação adequado a colocar questões complexas.
- As vistas fornecem segurança automática para dados que se deseja manter ocultos a alguns utilizadores.
- As vistas permitem que a mesma informação seja partilhada por vários utilizadores de formas diferentes ao mesmo tempo.
- As vistas permitem fornecer independência lógica de dados, isto é, as vistas podem manter-se mesmo quando a estrutura lógica da base de dados é modificada, por exemplo devido a re-estruturação lógica por questões de desempenho ou de crescimento (ver secção 4.5.4).

Esta útima vantagem é possivelmente a mais importante e permite manter as interfaces externas estáveis, mesmo com a evolução natural dos sistemas. As vistas permitem definir uma interface com um sistema externo sem revelar a estrutura da base de dados. Com a proliferação de sistemas interligados só é possível a evolução através da manutenção de vistas relativamente estáveis.

Uma vista pode ser definida como resultado de qualquer consulta em SQL. A figura seguinte ilustra a definição de uma vista.

```
CREATE VIEW CaixasDeVinhos (nomeVinho, #Caixa, ano)
AS  SELECT  nomeVinho, #Caixa, ano
    FROM    Adega ;
```

Figura 4.10 Definição em SQL de uma vista muito simples sobre uma base de dados

Transacções

Uma transacção sobre uma base de dados deve ser uma operação atómica, no sentido de que ou se realiza na sua totalidade ou não se realiza de todo. Qualquer SGBD, não só relacional, que seja utilizado em situações complexas, em particular envolvendo actualizações, deve suportar o conceito transaccional e os mecanismos para os implementar. Um exemplo de uma transacção é o pagamento de um serviço por uma entidade A a uma entidade B, envolvendo um levantamento e um depósito nas respectivas contas bancárias. O sistema deve garantir que se realizam as duas operações, ou, no caso de haver qualquer falha entre o início e a conclusão, deve garantir que o estado inicial é reposto. Em SQL este requisito deveria ser expresso da seguinte forma:

```
   BEGIN TRANSACTION ;       /* Efectuar pagamento $ de A a B        */
        UPDATE Conta A ;     /* Levantamento de $ em A               */
        UPDATE Conta B ;     /* Depósito de $ em B                   */
   IF não houve qualquer problema
        THEN COMMIT ;        /* Operação concluída sem falhas        */
        ELSE ROLLBACK ;      /* Operação realizada com falha         */
   END IF ;
```

Figura 4.11 Definição em SQL de uma transacção

No entanto a norma SQL não requer a inicialização expressa das transacções com o comando `BEGIN TRANSACTION` e cada implementação comercial recorre a diversas formas de implementar o controlo de transacções.

Qualquer SGBD, mesmo sem ser de modelo relacional deverá possibilitar a gestão de transacções.

Transacções num ambiente distribuído com concorrência

O termo concorrência refere-se à possibilidade de um SGBD permitir gerir um conjunto de transacções simultâneas sobre os mesmos dados. Para tal ser feito adequadamente, o sistema tem de ter um bom mecanismo de gestão e controlo de concorrência de operações.

O principal problema que tal mecansimo tem de gerir é o facto de algumas transacções terem por objectivo alterar dados, em particular, alterar os mesmos dados. Para evitar o aparecimento de informação inconsistente é essencial que exista um mecanismo que permita a uma transacção reservar para alteração, ou *bloquear*, registos da base de dados. Este bloqueio pode ser feito ao nível do registo, da tabela ou da própria base de dados. Em princípio, e

mesmo supondo que a tecnologia utilizada permitia múltiplos acessos simultâneos à mesma informação, apenas um desses acessos deveria poder alterar ou eliminar informação num dado momento e da cada vez. Considerem-se por exemplo transacções que efectuam transferências entre contas bancárias, por exemplo para pagamento de serviços, num caso em que uma dada conta está a ser creditada com muitas transacções de actualização. Tipicamente cada uma destas transacções efectuará duas operações de actualização sobre cada uma das contas. Cada uma destas operações (`UPDATE` em SQL) envolve a leitura de um registo na base de dados (para saber o saldo actual da conta), uma operação de cálculo (subtracção ou adição para determinar o valor final) e uma operação de escrita num registo (para actualizar o saldo). Como é óbvio neste exemplo, antes de uma destas transacções terminar a fase de escrita num dado registo, não se pode iniciar a fase de leitura desse registo por uma outra transacção.

4.2.3 Organização Genérica de um SGBDr Comercial

De seguida apresentam-se alguns exemplos do Microsoft Access 97 e do ORACLE.

Exemplo de Módulos do Microsoft Access 97

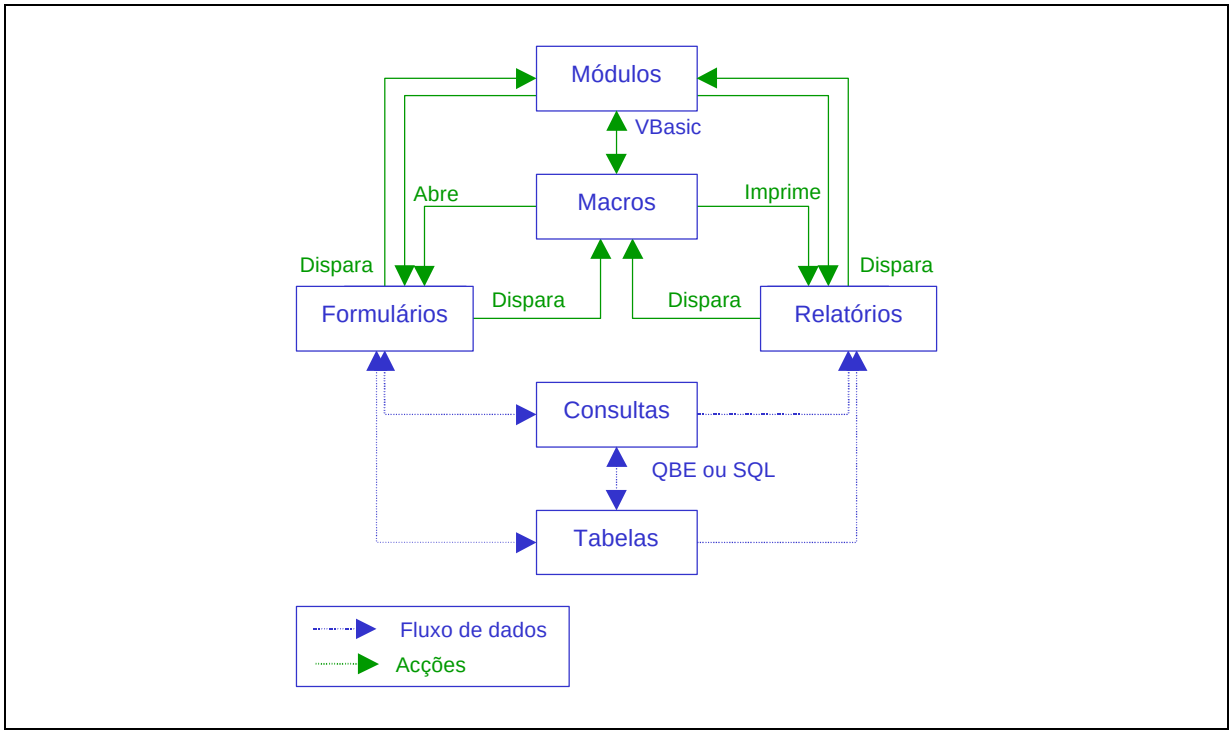


Figura 4.12 Módulos do SGBD Access 97 - ©1997 Microsoft

Uma Base de Dados Access pode ser colocada num servidor em rede e as aplicações clientes que acedem a essa base de dados podem ser colocadas em qualquer parte da rede. Embora sem grandes sofisticações, o Access oferece assim uma arquitectura cliente-servidor. Uma transacção da aplicação cliente, por exemplo baseada em módulos e formulários, ao efectuar uma consulta à base de dados solicita ao servidor o envio das tabelas que essa transacção envolve. A transacção é depois executada localmente, isto é no cliente. No caso de sistemas com uma separação clara entre cliente e servidor, cada transacção é enviada ao servidor e aí executada, regressando apenas a resposta ao cliente. No caso de o SGBD suportar transacções

pré-compiladas, como é o caso dos procedimentos armazenados (“stored procedure”) no ORACLE e no SQL Server, a execução é igualmente no servidor mas será em geral significativamente mais rapida.

PL/SQL e Oracle

A linguagem PL/SQL do Oracle é uma extensão procedimental da linguagem SQL que inclui ciclos, constantes, variáveis, cursores e permite o tratamento de excepções. Um programa em PL/SQL pode ser um **bloco de código anónimo**, ou pode ser um **procedimento armazenado** que pode ser executado invocando o seu nome. A linguagem PL/SQL permite definir procedimentos, funções e pacotes (“packages”). Um cursor permite manipular o resultado de uma consulta relacional em PL/SQL ou numa outra linguagem de programação com acesso a bases de dados (`strSQL` na Figura 4.7 é um exemplo de um cursor). É possível a partir de PL/SQL re-utilizar funções ou procedimentos escritos em C e guardados numa biblioteca partilhada.Tais rotinas executam em espaços de endereçamento distintos dos do servidor Oracle.

Um bloco de código anónimo, ou um procedimento armazenado podem incluir em Oracle expressões de **SQL Dinâmico**. Uma instrução de SQL Dinâmico permite por exemplo trabalhar com uma tabela cujo nome só é conhecido em tempo de execução. Para tal as instruções de SQL Dinâmico não são colocadas no código compilado, mas sim em variáveis de texto que são lidas ou construídas em tempo de execução.

Exemplo de Módulos do ORACLE – Designer 2000

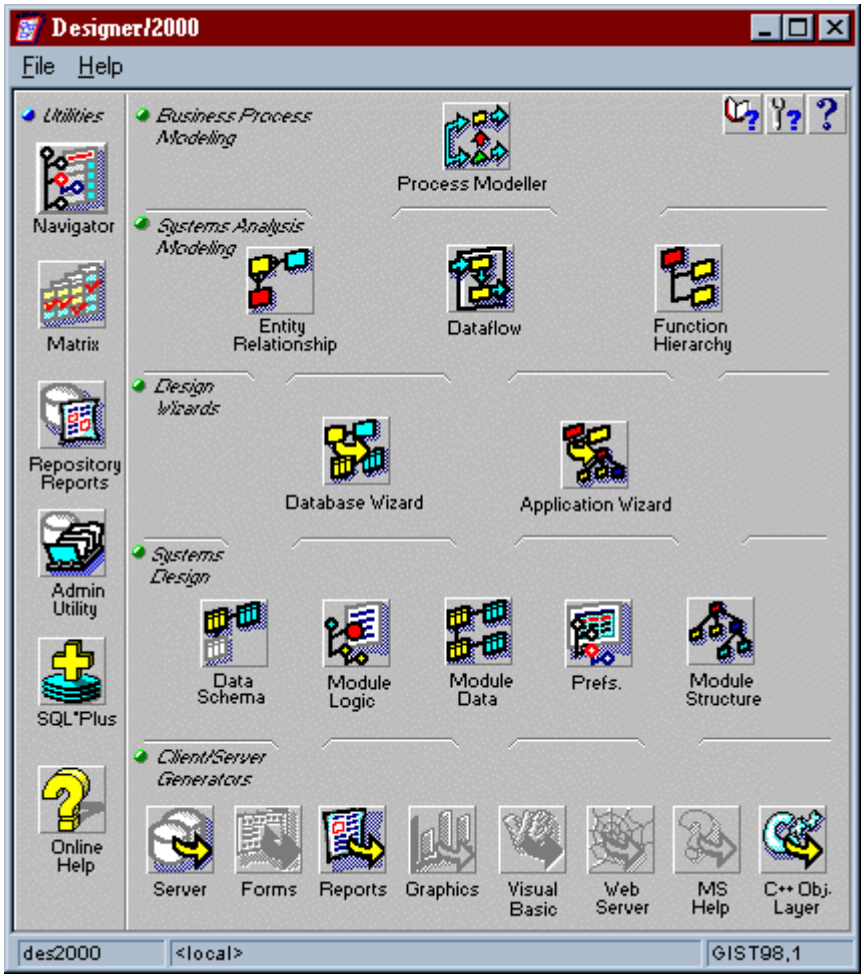


Figura 4.13 Janela Principal do Designer/2000 - ©2000 Oracle.

4.3 Álgebra e Cálculo Relacional

No final da década de 1960 os modelos de bases de dados hierárquicos e em rede, desenvolvidos de forma empírica, tinham revelado um conjunto de problemas suficientemente grandes para permitirem a aceitação de um novo modelo mais geral e flexível, fundamentado em sólidas bases teóricas. O trabalho desenvolvido e liderado por E. F. Codd na IBM culmina na publicação das bases teóricas do modelo relacional, nas suas três componentes de estrutura, integridade e manipulação [Codd 1970]. A componente de manipulação baseia-se na designada álgebra relacional, isto é num conjunto de objectos matemáticos operando sobre relações e retornando igualmente relações como resultado. Estes operadores foram adaptados por Codd para as relações que devem servir de modelo abstracto para a gestão de informação. A componente de manipulação do modelo relacional foi igualmente apresentada com base no designado cálculo relacional. Codd demonstrou que as duas abordagens eram perfeitamente equivalentes, através de um algoritmo que permitia a re-escrita de qualquer consulta descrita no cálculo relacional num conjunto de operações de consulta na álgebra relacional. O cálculo relacional é essencialmente declarativo, permitindo apenas indicar o resultado que se deseja obter. A álgebra relacional pelo contrário é operacional, uma vez que são indicadas as operações a realizar para obter o resultado.

A álgebra relacional baseia-se em operações sobre conjuntos, tais como união, intersecção e produto cartesiano, enquanto que o cálculo relacional se baseia na lógica de predicados, envolvendo a utilização de variáveis, predicados e quantificadores. A possibilidade de recorrer a estas duas bases teóricas interligadas forneceu ao modelo relacional uma fundamentação muito sólida, e permitiu um enriquecimento significativo dos processos de engenharia dos sistemas de bases de dados. O conhecimento destas bases teóricas é essencial para toda a actividade de engenharia de bases de dados, ao nível da construção dos sistemas (SGBD) e das aplicações.

4.3.1 Noções básicas

O modelo relacional está fundamentado em noções matemáticas à partida muito simples em particular *conjuntos*, *relações* e *funções*. Baseando-se nestas noções abstractas e completamente gerais, E. F. Codd desenvolveu uma álgebra e um cálculo adaptada à gestão de informação com significado nas organizações. De seguida apresentam-se as noções elementares *par ordenado, produto cartesiano AxB, relação binária* e *função de A para B*. Estas noções são depois generalizadas para o caso de dimensões superiores à binária, e adaptadas para servirem de base à álgebra relacional e ao cálculo relacional. Em particular a noção de *ordem* que existe no *par ordenado* e no *produto cartesiano* será parcialmente abandonada, uma vez que a ordem dos atributos numa tabela deve ser irrelevante[9].

[9] De facto é *quase* irrelevante. A chave primária de uma relação pode ser considerada uma ordenação que se mantém nos atributos. A chave primária, como se verá, impõe uma ordem ao nível da estrutura do conjunto de atributos em que a relação está definida: o sub-conjunto dos atributos chave e o sub-conjunto restante.

Par ordenado, Produto cartesiano, Relação e Função

Consideram-se conhecidas as formas de introdução de conjuntos em *extensão*, indicando todos os seus elementos, e em *compreensão*, indicando uma *propriedade* de que gozem todos os seus elementos e só eles.

Definição 4.1 ***Par ordenado***

Dados dois conjuntos *A* e *B* e seleccionando um elemento *a* de *A* ($a \hat{I} A$) e um elemento *b* de *B* ($b \hat{I} B$) podemos criar um *par ordenado (a, b)*. O elemento *a* é o 1º *termo* do par *(a, b)*, e *b* é o 2º *termo* do par *(a, b)*.

Por exemplo, sendo *A={1, 2, 3}* e *B={x: x é um número impar}*, temos por exemplo os pares ordenados (3, 3), (1, 3) ou (3, 1). Note-se que os pares (1, 3) e (3, 1) são diferentes (por isso se designam por pares *ordenados*), e que o par ordenado (1, 2) não pode ser definido nos conjuntos dados (o número 2 não é ímpar).

Definição 4.2 ***Produto Cartesiano AxB***

O *produto cartesiano* de *A* por *B* é o conjunto de todos os pares ordenados que é possível constituir de *A* para *B*, sendo representado esse conjunto por *AxB*.

Muitas vezes é possível ou útil representar um produto cartesiano como um conjunto de pontos de um plano. Se for possível representar o conjunto *A* e o conjunto *B* como valores em eixos ortogonais, os pares ordenados passam a corresponder aos pontos do plano definido pelos valores nesses eixos. O eixo com os valores do conjunto *A* é o designado *eixo das abcissas* e o eixo com os valores de *B* é o *eixo das ordenadas*.

Muitas vezes é possível igualmente representar um produto cartesiano como uma tabela de duas colunas: a primeira coluna tem os valores de A e a segunda coluna tem os valores de B (ver Figura 4.31).

Definição 4.3 ***Relação Binária de A para B***

Uma *relação binária* de *A* para *B* é um subconjunto dos pares ordenados de *AxB*.

Considere-se o conjunto *P*, com nomes de países, e o conjunto *C*, com nomes de cidades.

P={Portugal, Angola, França, Brasil}

C={Brasília, Recife, Lisboa, Londres, Porto, Paris, Luanda}.

Considere-se a relação R_1*={(p, c): c é a capital de p}* definida em compreensão. Na correspondência que se estabelece entre o conjunto *P* e o conjunto *C* cada país é transformado na cidade sua capital. A expressão «capital de» funciona como uma máquina onde se introduzem os elementos de *P* para serem transformados nos elementos de *C*.

R_1={(Portugal, Lisboa), (Angola, Luanda), (França, Paris), (Brasil, Brasília)}.

Definição 4.4 ***Função de A para B***

Sendo *A* e *B* dois conjuntos não vazios, chama-se *função* de *A* para *B*, ou *função* definida em *A* com valores em *B*, a toda a **correspondência unívoca** *f* de *A* para *B* (isto é uma

correspondência em que cada um dos elementos de *A* tem uma e uma só imagem em *B*). Por vezes o termo **aplicação** surge como sinónimo de função. Ao conjunto *A* chama-se **domínio** da função *f*, indicando-se como D_f. Ao conjunto *B* chama-se **conjunto de chegada** da função *f*, indicando-se como D'_f ou CD_f. Ao conjunto das imagens dos elementos do domínio da aplicação chama-se **conjunto imagem** ou **contradomínio** da função. O contradomínio pode não coincidir com o conjunto de chegada. Sendo *x* um elemento de *A*, representa-se por *f(x)* a correspondência de *x* ou imagem de *x* em *B*.

Uma função *f* diz-se **injectiva** se quaisquer dois elementos distintos do domínio têm imagens diferentes. Estando a função *f* representada graficamente, vê-se que essa função não é injectiva se é possível traçar pelo menos uma recta horizontal que intersecte o gráfico da função em mais do que um ponto.

Considere-se por exemplo a relação R_2=*{(p, c): c é cidade de p}*. A correspondência do conjunto *P* para o conjunto *C* não é unívoca. Há elementos do primeiro conjunto que têm mais de uma imagem no segundo (para facilitar a compreensão pode ser desenhado o diagrama sagital correspondente).

R_2={(Portugal, Lisboa), (Portugal, Porto), (Angola, Luanda), (França, Paris), (Brasil, Brasília), (Brasil, Recife)}.

Considere-se a relação R_3=*{(p, c): c é a maior cidade do continente onde se encontra p}*. A correspondência do conjunto *P* para o conjunto *C* será de novo unívoca[10]. Cada elemento do primeiro conjunto tem apenas uma imagem no segundo (embora a correspondência ou função inversa não seja unívoca).

Considere-se finalmente a relação R_4=*{(p, c): c é uma capital de p com porto de mar ou rio}*. A correspondência do conjunto *P* para o conjunto *C* será de novo unívoca, mas nem todos os elementos do primeiro conjunto têm imagem no segundo. Só são transformados os elementos de {Portugal, Angola, França}.

R_4={(Portugal, Lisboa), (Angola, Luanda), (França, Paris)}.

Relações e funções em espaços de outras dimensões

Todas as noções binárias introduzidas anteriormente, *par ordenado*, *produto cartesiano*, *relação* e *função*, podem ser generalizadas para *n* dimensões. Em geral, uma relação pode ser definida sobre um espaço de *n* dimensões, onde cada dimensão pode ser vista como um conjunto de valores. Em dimensões *n* superiores à binária a noção de **par ordenado** é substituida pela noção de **n-uplo ordenado**.

Uma certa relação[11] *R* é assim um conjunto de *n-uplos ordenados* de valores definidos em cada uma das dimensões de definição de *R*. Facilmente se pode entender a correspondência

[10] Considerando que não há dúvidas sobre qual a maior cidade de cada continente, e considerando que não há dúvidas sobre o continente de um dado país (o que na realidade nem sempre acontece ...).

[11] Esta relação, que foi nomeada R, deveria ser aqui designada pela letra minúscula *r* para evitar confusão com o espaço R, no qual a relação é um valor (como conjunto de pontos). Como a notação utilizada para as relações e para as variáveis de relação recorre a identificadores com letra inicial maiúscula, decidiu-se manter o identificador maiúsculo para ambos os casos.

entre n-uplos ordenados de um conjunto, pontos de um espaço com *n* dimensões, e linhas de valores em tabelas com *n* colunas.

Dados *n* conjuntos de valores, D_1, D_2, ..., D_n, pode definir-se um espaço designado *R* como sendo o produto cartesiano dos conjuntos com nomes D_1, D_2, ..., D_n: $R=D_1xD_2x,...,xD_n$. Cada relação é assim um subconjunto deste espaço *R*. O espaço *R*, visto também como conjunto, tem como elementos todas as n-uplas do tipo $(x_1, x_2, ..., x_n)$ com $x_i \hat{I} D_i$. Para efeitos de utilização em modelação de bases de dados relacionais, cada um destes conjuntos D_i tem de conter elementos com valores primitivos escalares[12]. Como se verá mais tarde, as relações que interessam para o modelo relacional são *relações concretas*, definidas sobre um conjunto de atributos *informativos*, atributos esses que tomam valores em conjuntos de valores abstractos D_i. Cada relação concreta **R** tem como valor um subconjunto de *R*, ou seja, é também um conjunto de n-uplos, elementos de *R*.

Por exemplo, seja $N=\{0, 1, 2, 3 ...\}$ (números inteiros), $D_1=D_2=N$, ou seja, $N^2=NxN$. Um espaço deste tipo permite definir obviamente muitas relações. Por exemplo, pode definir-se a relação **Fact**: N^2 (lê-se: relação **Fact** *do tipo* N^2) da seguinte forma: **Fact**$=\{(x, y): y=x!\}$. Esta relação podia ser apresentada, em extensão, como o conjunto infinito de pares ordenados: **Fact**$=\{(0, 1), (1, 1), (2, 2), (3, 6), (4, 24), ...\}$. A relação **Fact** é assim uma de entre muitas relações que se podem definir no domínio N^2.

O produto cartesiano que vai ser utilizado na álgebra relacional é um pouco diferente do anteriormente apresentado, sendo definido directamente sobre relações, e não sobre os conjuntos de definição dessas relações.

Definição 4.5 ***Produto cartesiano entre as relações*** **A** ***e*** **B*:*** **A** x **B**

Sejam **A** e **B** duas relações definidas respectivamente sobre atributos com valores nos conjuntos $A=A_1xA_2x,..., xA_n$ e $B=B_1xB_2x,..., xB_m$. Considere-se igualmente que não há repetições de nomes de atributos correspondentes aos conjuntos $A_1, A_2, ..., A_n, B_1, B_2, ...,$ e B_m (se tal suceder devemos previamente alterar os nomes dos atributos iguais). O produto cartesiano relacional entre as duas relações **A** e **B**, designado **A** x **B**, é a seguinte relação **P**:

$$\mathbf{P}=\{(a_1, a_2, ..., a_n, b_1, b_2, ..., b_m): (a_1, a_2, ..., a_n) \hat{I}\ \mathbf{A}\ e\ (b_1, b_2, ..., b_m) \hat{I}\ \mathbf{B}\}.$$

Deve notar-se que para fins da álgebra e do cálculo relacional apresentados de seguida, e de todo o modelo relacional, as relações e os atributos utilizados têm nomes cujos significados são relevantes, e que correspondem a tipos ou conjuntos de valores admissíveis. Os nomes dos conjuntos anteriores (A_1, ..., B_1, ...) serão igualmente referidos como atributos de uma relação, e também como nomes das colunas de uma tabela. De facto *nome do conjunto*, *nome do atributo* e *nome da coluna* são conceitos a níveis de abstracção diferentes[13].

[12] De facto esta condição não é essencial, mas os SGBD comerciais exigem valores escalares, com a possível excepção dos tipos de dados que gerem datas de calendário e tempos.

[13] Reservam-se as designações nome de **tabela**, elemento da **linha**, título de **coluna** e valor da **célula**, para o nível de abstracção da base de dados relacional; ao nível de abstracção do modelo matemático os conceitos utilizados são os seguintes: nome de **relação**, **n-upla**, nome do **atributo** e **termo** da n-upla. No nível fisico provavelmento seriam utilizazadas as designações **ficheiro** (“file”), **ficha** (“record”), **registo** (“field”) e **campo** (“place”).

Como se verá de seguida, a utilização de nomes de atributos com significado, e muito em particular, das chaves primárias, permite relaxar a necessidade de *ordem* nos termos de um *par ordenado* (e de ordem nos termos de um n-uplo). Como consequência desta alteração **o produto cartesiano entre relações é comutativo e associativo**. A ordem de apresentação dos atributos e respectivos valores numa relação é aqui totalmente arbitrária, embora seja normal apresentar em primeiro lugar os atributos da chave primária. Como se verá, os atributos que irão formar a chave primária de uma relação correspondem exactamente a uma escolha de *domínios* das funções que se podem identificar numa relação (*domínios* D_f, no sentido utilizado na Definição 4.4). Se todas as relações da base de dados estiverem em terceira forma normal, existirá apenas uma função em cada relação tabelada. Para tal deve ser considerada a chave primária como o nome do domínio de definição da função, e os atributos da parte dependente como o nome do domínio de chegada (sendo o conjunto de valores tabelados na parte dependente o contradomínio).

4.3.2 Álgebra Relacional

De seguida apresentam-se sumariamente os operadores de álgebra relacional em que se baseiam as operações de interrogação num sistema relacional. A definição formal, sintáctica e semântica é exemplificada de seguida para um destes operadores, o designado operador de *junção natural*. Como se poderá observar, cada operação da álgebra tem como resultado uma relação (diz-se que *retorna* uma relação). Desta forma é possível utilizar o resultado de uma operação como argumento de uma outra operação (*aninhando* operações). A linguagem SQL permite igualmente esta facilidade.

Operadores da Álgebra Relacional

1. **Restrição ou selecção**: dada uma relação **A** de um determinado tipo e uma condição *c* sobre os valores das n-uplas dessa relação, a operação de selecção retorna uma outra relação do mesmo tipo apenas com as n-uplas que satisfazem a condição. Indica-se da seguinte forma:

   R := [Restrição de **A** por *c*].

2. **Projecção**: dada uma relação **A** definida num conjunto de atributos Y, e sendo indicado um subconjunto X desses atributos Y, a operação de projecção retorna uma outra relação definida apenas para os atributos X. Indica-se da seguinte forma:

   R := [Projecção de **A** em X]

3. **Produto Cartesiano**: dadas duas relações **A** e **B**, o produto cartesiano retorna uma outra relação, conforme definido anteriormente. Se as relações de entrada tiverem n_1 e n_2 tuplas, a relação resultante terá $n_1 x n_2$ tuplas. Indica-se da seguinte forma:

   R := **A** x **B**

4. **União**: dadas duas relações do mesmo tipo (isto é definidas sobre atributos do mesmo tipo) a união retorna uma relação com as tuplas de ambas. Note-se que não haverá tuplas repetidas na relação resultante. Indica-se da seguinte forma:

   R := **A** + **B**

5. **Intersecção**: dadas duas relações do mesmo tipo (isto é definidas sobre atributos do mesmo tipo) a intersecção retorna uma relação com as tuplas da primeira que também aparecem na segunda. Indica-se da seguinte forma:

   R := **A** ∩ **B**

6. **Diferença**: dadas duas relações do mesmo tipo (isto é definidas sobre atributos do mesmo tipo) a diferença retorna uma relação com as tuplas da primeira que não aparecem na segunda. Indica-se da seguinte forma:

   R := **A** - **B**

7. **Junção natural** ou simplesmente **junção** ("join"): A junção natural é baseada no produto cartesiano, mas requer que as duas relações sobre as quais é aplicada a operação tenham um sub-conjunto de atributos comuns dos mesmos tipos (não precisam de ter os mesmos nomes nas duas relações, mas devem corresponder aos mesmos tipos de dados). A junção de **A** e **B** pelos atributos X será a relação **J** que se obtém da seguinte forma. Sejam **A**{A_a, X} e **B**{Y, B_b}, em que A_a, X, Y, B_b são listas de atributos, correspondendo X e Y aos sub-conjuntos de atributos comuns às relações **A** e **B**. Então a relação **J** pode ser obtida da seguinte forma (ver exemplo de uma junção na Figura 4.16):

   R := [Projecção de [Restrição de [**A** x **B**] por X=Y] em A_a, X, B_b]

   (ou **R** := [Projecção de [Restrição de [**A** x **B**] por X=Y] em A_a, Y, B_b]).

   Normalmente a operação de junção será indicada da seguinte forma, em que X e Y são os atributos comuns:

   R := Junção de [**A** e **B**] por X

   (ou **R** := Junção de [**A** e **B**] por Y)

8. **Divisão**: Fica como exercício a definição deste operador relacional. Indica-se da seguinte forma:

   R := **A** / **B**

As operações 3, 4, 5 e 6 são as tradicionais operações sobre conjuntos adaptadas agora às relações, de acordo com o referido na secção anterior.

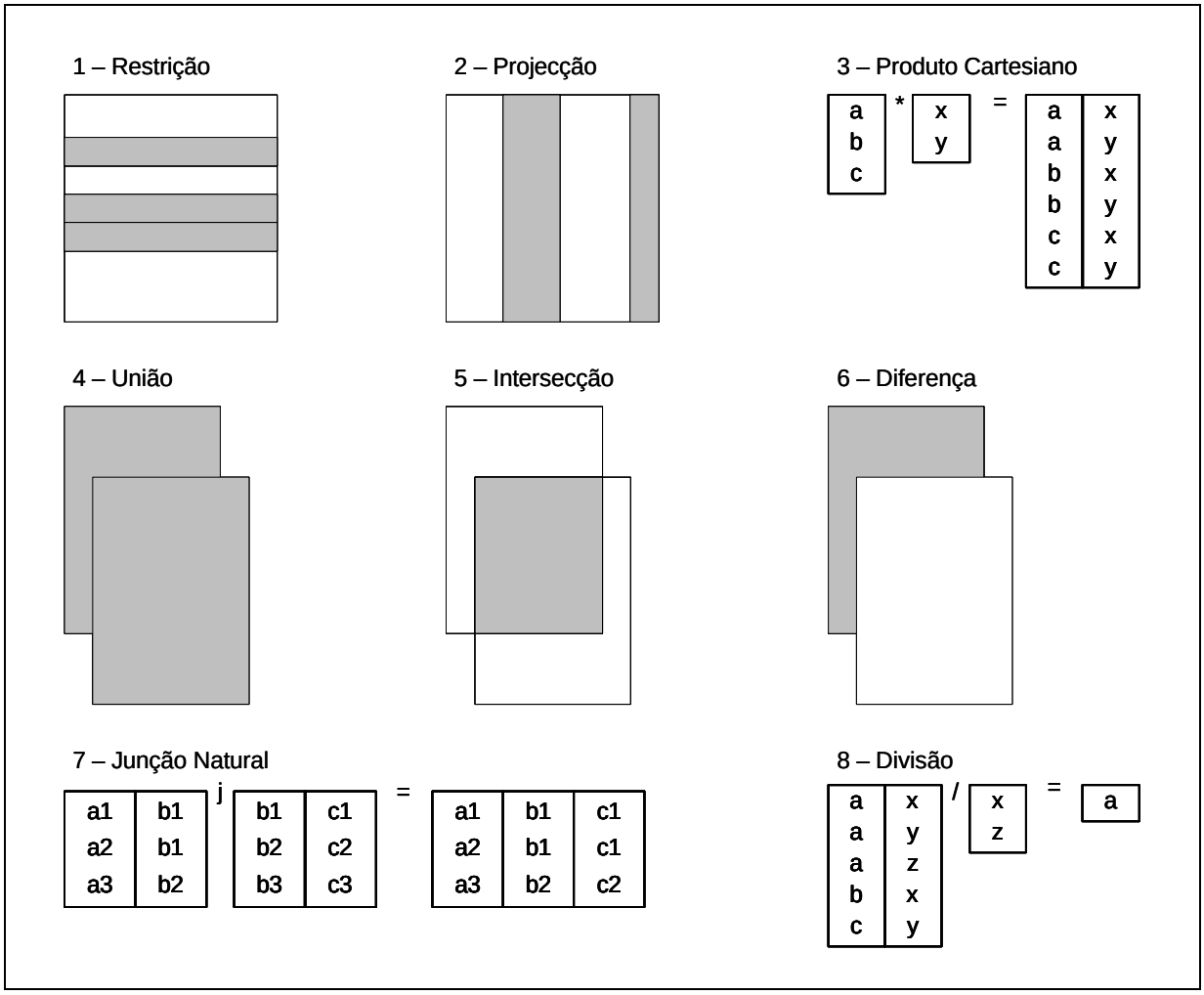
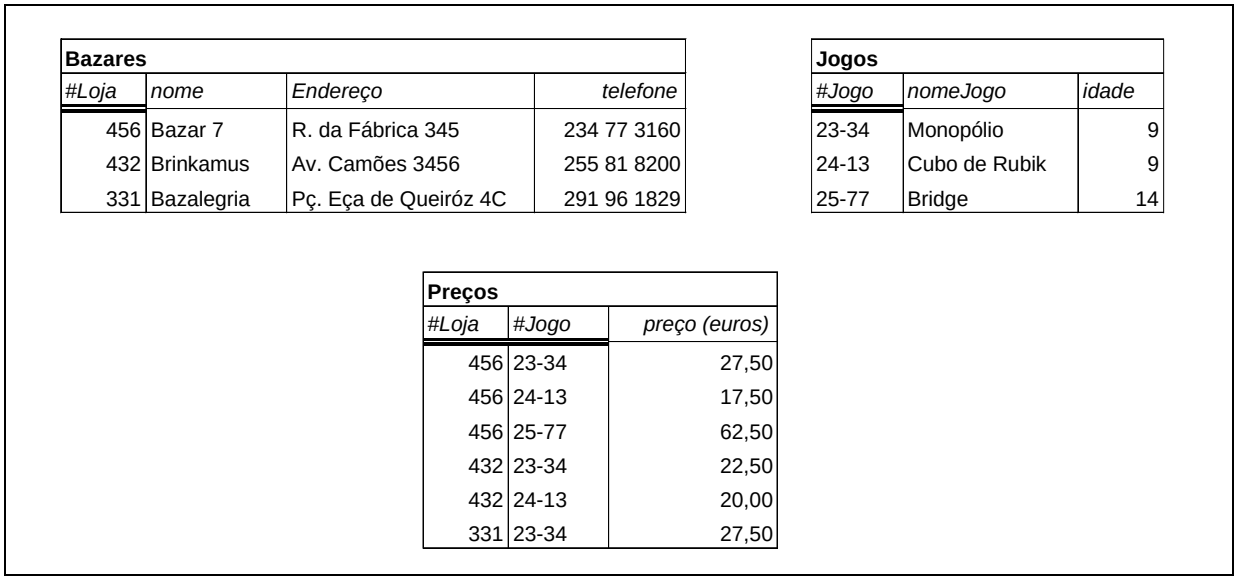


Figura 4.14 Os oito operadores originais da Álgebra Relacional [Date 1995]

4.3.3 A Base de Dados BJP – Bazares, Jogos e Preços

Na Figura 4.15 representam-se três relações possíveis de uma base de dados com informação sobre lojas de jogos e brinquedos. Esta base de dados muito simples ilustrará de seguida algumas operações de álgebra relacional.

A Figura 4.16 representa o resultado da operação de junção das relações **Preços** e **Jogos** pelo atributo *#Jogo*, e a Figura 4.17 exemplifica como esse mesmo resultado pode ser obtido através da aplicação sucessiva do produto cartesiano, restrição e projecção, de acordo com a definição da operação de junção.



Bazares

#Loja	*nome*	*Endereço*	*telefone*
456	Bazar 7	R. da Fábrica 345	234 77 3160
432	Brinkamus	Av. Camões 3456	255 81 8200
331	Bazalegria	Pç. Eça de Queiróz 4C	291 96 1829

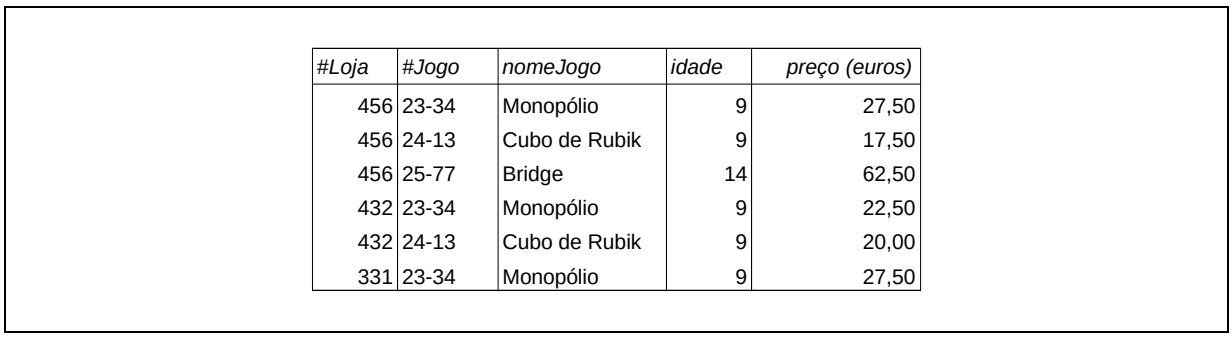
Jogos

#Jogo	*nomeJogo*	*idade*
23-34	Monopólio	9
24-13	Cubo de Rubik	9
25-77	Bridge	14

Preços

#Loja	*#Jogo*	*preço (euros)*
456	23-34	27,50
456	24-13	17,50
456	25-77	62,50
432	23-34	22,50
432	24-13	20,00
331	23-34	27,50

Figura 4.15 Tabelas da base de dados **Bazares**, **Jogos** *e* **Preços**.



#Loja	*#Jogo*	*nomeJogo*	*idade*	*preço (euros)*
456	23-34	Monopólio	9	27,50
456	24-13	Cubo de Rubik	9	17,50
456	25-77	Bridge	14	62,50
432	23-34	Monopólio	9	22,50
432	24-13	Cubo de Rubik	9	20,00
331	23-34	Monopólio	9	27,50

Figura 4.16 Junção de **Preços** *e* **Jogos** *por #Jogo (junção obtida directamente)*

Na Figura 4.17 são utilizadas relações **R1** e **R2** (representadas como tabelas) para mostrar os resultados intermédios. Numa operação de produto cartesiano sobre relações com nomes de atributos iguais, poderia surgir ambiguidade nos nomes dos atributos. Nestes caso, e tal como se pode observar na figura, costuma acrescentar-se o nome da relação de origem aos atributos. No exemplo seguinte, em vez de se obter uma relação com dois atributos *#Jogo*, indicam-se estes atributos como **Preços**.*#Jogo* e **Jogos**.*#Jogo* (por uma questão de espaço a figura utiliza apenas *P.#Jogo* e *J.#Jogo*). Outra possibilidade seria renomear os atributos antecipadamente, através de uma operação especial, de forma a evitar este problema com os nomes dos atributos.

Como referido, a operação de junção das tabelas **Jogos** e **Preços** pode ser obtida directamente, como na Figura 4.16, assim que o significado da sua definição seja entendido. A operação de junção pode em alternativa ser obtida através da definição, recorrendo a uma operação inicial de produto cartesiano entre relações.

Em geral, a operação de junção é definida da seguinte forma:

R := [Junção de [**A** e **B**] por X] := [Projecção de [Restrição de [**A** x **B**] por (X=Y)] em A_a, X, B_b].

As variáveis **A**, **B**, X, Y, A_a, X e B_b utilizadas são agora as seguintes:

A = **Preços**; X={*#Jogo*} (na relação **Preços**: **Preços**.#Jogo); A_a= {*#Loja, preço (euros)*}.

B = **Jogos**; Y={*#Jogo*} (na relação **Jogos**: **Jogos**.#Jogo); B_b= {*nomeJogo, idade*}.

Então iremos obter:

R = [Projecção de
[Restrição de [**Preços** x **Jogos**] por (**Preços**.#Jogo=**Jogos**.#Jogo)]
em {*#Loja, preço (euros)*, #Jogo, *nomeJogo, idade*}].

R1 := [**Preços x Jogos**]

#Loja	*P.#Jogo*	*preço (euros)*	*J.#Jogo*	*nomeJogo*	*idade*
456	23-34	27,50	23-34	Monopólio	9
456	23-34	27,50	24-13	Cubo de Rubik	9
456	23-34	27,50	25-77	Bridge	14
456	24-13	17,50	23-34	Monopólio	9
456	24-13	17,50	24-13	Cubo de Rubik	9
456	24-13	17,50	25-77	Bridge	14
456	25-77	62,50	23-34	Monopólio	9
456	25-77	62,50	24-13	Cubo de Rubik	9
456	25-77	62,50	25-77	Bridge	14
432	23-34	22,50	23-34	Monopólio	9
432	23-34	22,50	24-13	Cubo de Rubik	9
432	23-34	22,50	25-77	Bridge	14
432	24-13	20,00	23-34	Monopólio	9
432	24-13	20,00	24-13	Cubo de Rubik	9
432	24-13	20,00	25-77	Bridge	14
331	23-34	27,50	23-34	Monopólio	9
331	23-34	27,50	24-13	Cubo de Rubik	9
331	23-34	27,50	25-77	Bridge	14

R2 := [Restrição de [**Preços** x **Jogos**] por (**Preços**.#Jogo=**Jogos**.#Jogo)]

#Loja	*P.#Jogo*	*preço (euros)*	*J.#Jogo*	*nomeJogo*	*idade*
456	23-34	27,50	23-34	Monopólio	9
~~456~~	~~23-34~~	~~27,50~~	~~24-13~~	~~Cubo de Rubik~~	~~9~~
~~456~~	~~23-34~~	~~27,50~~	~~25-77~~	~~Bridge~~	~~14~~
~~456~~	~~24-13~~	~~17,50~~	~~23-34~~	~~Monopólio~~	~~9~~
456	24-13	17,50	24-13	Cubo de Rubik	9
~~456~~	~~24-13~~	~~17,50~~	~~25-77~~	~~Bridge~~	~~14~~
~~456~~	~~25-77~~	~~62,50~~	~~23-34~~	~~Monopólio~~	~~9~~
~~456~~	~~25-77~~	~~62,50~~	~~24-13~~	~~Cubo de Rubik~~	~~9~~
456	25-77	62,50	25-77	Bridge	14
432	23-34	22,50	23-34	Monopólio	9
~~432~~	~~23-34~~	~~22,50~~	~~24-13~~	~~Cubo de Rubik~~	~~9~~
~~432~~	~~23-34~~	~~22,50~~	~~25-77~~	~~Bridge~~	~~14~~
~~432~~	~~24-13~~	~~20,00~~	~~23-34~~	~~Monopólio~~	~~9~~
432	24-13	20,00	24-13	Cubo de Rubik	9
~~432~~	~~24-13~~	~~20,00~~	~~25-77~~	~~Bridge~~	~~14~~
331	23-34	27,50	23-34	Monopólio	9
~~331~~	~~23-34~~	~~27,50~~	~~24-13~~	~~Cubo de Rubik~~	~~9~~
~~331~~	~~23-34~~	~~27,50~~	~~25-77~~	~~Bridge~~	~~14~~

R2 :=

#Loja	*P.#Jogo*	*preço (euros)*	*J.#Jogo*	*nomeJogo*	*idade*
456	23-34	27,50	23-34	Monopólio	9
456	24-13	17,50	24-13	Cubo de Rubik	9
456	25-77	62,50	25-77	Bridge	14
432	23-34	22,50	23-34	Monopólio	9
432	24-13	20,00	24-13	Cubo de Rubik	9
331	23-34	27,50	23-34	Monopólio	9

R := [Projecção de **R2** em {*#Loja,* #Jogo, *nomeJogo, idade, preço (euros)*}]

#Loja	*P.#Jogo*	*preço (euros)*	~~*J.#Jogo*~~	*nomeJogo*	*idade*
456	23-34	27,50	~~23-34~~	Monopólio	9
456	24-13	17,50	~~24-13~~	Cubo de Rubik	9
456	25-77	62,50	~~25-77~~	Bridge	14
432	23-34	22,50	~~23-34~~	Monopólio	9
432	24-13	20,00	~~24-13~~	Cubo de Rubik	9
331	23-34	27,50	~~23-34~~	Monopólio	9

#Loja	*#Jogo*	*nomeJogo*	*idade*	*preço (euros)*
456	23-34	Monopólio	9	27,50
456	24-13	Cubo de Rubik	9	17,50
456	25-77	Bridge	14	62,50
432	23-34	Monopólio	9	22,50
432	24-13	Cubo de Rubik	9	20,00
331	23-34	Monopólio	9	27,50

Figura 4.17 Exemplo de como obter uma junção em termos de um produto cartesiano.

4.3.4 Cálculo Relacional

O cálculo relacional é equivalente em capacidade de consulta à álgebra relacional. No entanto como o seu estilo é diferente, sendo em alguns casos a sua utilização mais intuitiva do que a da álgebra relacional para formular consultas, a linguagem SQL adoptou algumas particularidades do cálculo. As referências [Date 1997] ou [Date 2000] abordam este tema em detalhe.

4.3.5 Álgebra relacional, cálculo relacional e SQL

Uma linguagem de consulta é dita **relacionalmente completa** se é pelo menos tão poderosa quanto o cálculo relacional, isto é, se toda a relação que pode ser definida no cálculo pode ser obtida pela linguagem. Uma linguagem de consulta é dita **computacionalmente completa** se qualquer processo de cálculo que é intuitivamente computável pode ser descrito nessa linguagem.

Uma linguagem relacional pode basear-se na álgebra relacional ou no cálculo relacional. A linguagem SQL baseia-se parcialmente em ambas, mas inclui igualmente características próprias. Na prática para além de existirem as normas internacionais, cada implementação comercial disponibiliza dialectos próprios, o que requer um estudo cuidadoso das particularidades.

A grande maioria das implementações comerciais de SQL não é relacionalmente completa nem computacionalmente completa. Por este último motivo certas consultas requerem a utilização de um linguagem de programação completa, isto é, com ciclos ou com recursão.

A construção de sistemas que se mantenham independentes do SGBD e cuja evolução seja possível requer conhecimentos da teoria base do SQL, das normas em vigor e em estudo, bem como das particularidades existentes nas implementações. Como é normal, a opção de utilização de particularidades existentes num dado momento é uma decisão de engenharia.

A secção seguinte introduz muito sucintamente a linguagem SQL.

4.4 Introdução à Linguagem SQL

A linguagem SQL[14] é a norma de interface para bases de dados relacionais. Qualquer sistema relacional deveria permitir a utilização de SQL, em particular da norma ISO/ANSI actualmente mais difundida designada por SQL/92, SQL-92 ou simplesmente SQL2, cujo nome oficial é **International Standard Database Language SQL (1992)** (conforme documentos ISO/IEC 9075:1992 ou ANSI X3.135-1992).

De facto cada um dos sistemas comerciais existentes suporta normalmente funcionalidades para além das que constam da norma, embora não suportem toda a norma. Está em preparação uma actualização da norma que visa incluir no SQL um conjunto muito acrescido de funcionalidades e modelos de interface, norma essa designada SQL3, mas que aguarda a qualquer momento ratificação nos organismos de normalização, e por esse motivo não está ainda a ser suportada pelas versões comerciais dos SGBD. A designação SQL utilizada aqui significará sempre SQL-92.

A SQL permite exprimir operações dos quatro tipos anteriormente referidos, ou seja permite *actualizar a estrutura* da base de dados, *actualizar* a informação, *consultar* informação e ainda, em geral, *gerir* a base de dados. A primeira operação necessária é claramente a nomeação de uma base de dados e a definição das suas tabelas.

4.4.1 A Base de Dados PVLF – Produtores, Vinhos, Lojas e Fornecimentos

O seguinte conjunto de tabelas constitui uma base de dados relacional normalizada que será utilizada nos exemplos de SQL.

[14] A sigla SQL teve origem na designação “Structured Query Language”, e pronuncia-se em inglês com “Sequel”. Apesar da origem do nome, a linguagem SQL é de utilização genérica, muito para além da capacidade de exprimir interrogações.

P (Produtores)				
#Produtor	*nomeCurtoProdutor*	*sedeProdutor*	*tipoProdutor*	*telefoneProdutor*
1	Caves Aliança	Sangalhos	100	351-234 74 3160
2	Quinta da Aveleda	Penafiel	80	351-255 71 8200
3	Ferreira	Vila Nova de Gaia	95	351-22 374 5292
4	Vinhos Borges	Vila Nova de Gaia	90	351-22 782 1896
5	Vinhos Poças Júnior	Vila Nova de Gaia	110	351-22 377 1070
6	Adega Coop. de Monção	Monção	50	351-251 65 2167
7	António Esteves Ferreira	Melgaço	50	351-251 41 6769
8	Casa Pinheiro	Monção	85	351-251 66 6053
9	Quinta da Gaivosa	St. Marta de Penaguião	30	351-259 37 2440
10	Carlos Alberto Codesso	Melgaço	30	351-251 40 4444
11	Niepoort Vinhos	Porto	54	351-22 338 9528
12	Quinta do Silval	Pinhão	23	351-22 510 7775
13	Cockburn's	Vila Nova de Gaia	76	351-22 377 6500
14	Quinta do Valado	Peso da Régua	76	351-254 33 6217
15	Maria Vitória Lencastre	Marco de Canavezes	98	351-255 53 5714
16	Sogrape	Avintes	110	351-22 783 8104
17	Quinta do Carmo	Estremoz	55	351-268 33 7320
18	Casa de Villar	Lousada	75	351-255 91 1380
19	Adega Coop. Torres Vedras	Lousada	50	351-261 33 5500
20	Tapada do Chaves	Portalegre	80	351-245 20 1973
21	Barbeito	Funchal	80	351-291 76 1829

Figura 4.18 Exemplo da Tabela **Produtores** *de uma base de dados relacional*

V (Vinhos)

#Vinho	*nomeVinho*	*cl*	*cor*	*região*	*#Produtor*
1	Barca Velha	75	Tinto	Douro	3
2	Foral Grande Escolha	75	Tinto	Douro	1
33	Morgadio da Torre	75	Branco	Vinho Verde	16
34	Redoma	75	Rosé	Douro	11
37	Reserva	75	Tinto	Douro	16
38	Reserva	75	Branco	Douro	16
44	Charamba	75	Branco	Douro	2
45	Quinta de Azevedo	75	Branco	Vinho Verde	16
55	Quinta da Leda	75	Tinto	Douro	3
56	Dorna Velha	75	Tinto	Douro	12
57	Vallado	75	Tinto	Douro	14
67	Borges	75	Tinto	Douro	4
76	Coroa Douro	75	Tinto	Douro	5
77	Tapada do Chaves	75	Tinto	Alentejo	20
88	Muralhas de Monção	75	Branco	Vinho Verde	6
99	Soalheiro	75	Branco	Vinho Verde	7
101	Casa de Vila Boa	75	Tinto	Vinho Verde	15
116	Vinho da Senhora	75	Branco	Vinho Verde	18
220	Quinta do Carmo	75	Tinto	Alentejo	17
221	Quinta do Carmo	75	Branco	Alentejo	17
301	Reserva	75	Tinto	Estremadura	19
345	Quinta de Alderiz	75	Branco	Vinho Verde	8
456	Dona Paterna	75	Branco	Vinho Verde	10
508	Vinha dos Freires	75	Tinto	Douro	14
566	Quinta das Caldas	75	Tinto	Douro	9
621	Dom Martinho	75	Tinto	Alentejo	17
677	Tuella	75	Branco	Douro	13
703	Vintage Quinta da Leda	75	Tinto	Vinho do Porto	3
789	Malvazia	75	Aloirado	Vinho da Madeira	21

FV (FornecimentoDeVinhos)

#Loja	*#Vinho*	*qtdd*
9	8	7000
9	9	600
9	10	5000
9	11	400
9	12	1400
9	13	10000
9	16	4000
34	8	4000
34	9	500
34	10	500
76	9	3000
76	16	400
98	6	750
98	7	2000
98	8	2000
98	14	750

L (Lojas)

#Loja	*nomeLoja*	*localLoja*	*ipLoja*
45	Vinho Biológico	Gaia	7
76	Vinho e Companhia	Porto	8
98	Venda de Vinho	Dili	9
9	Vinho de Portugal	Maputo	10
34	Vinho de Casa Antiga	Portalegre	45
33	Vinho de Uvas Novas	Matosinhos	12
43	Estiva e Vinha	Porto	13
12	Vinho da Videira	Recife	14
88	Garrafeira de Vinho	Luanda	15
91	Vinho Velho	Vila Real	16

Figura 4.19 Exemplo de Tabelas **Vinhos**, **Lojas** *e* **FornecimentoDeVinhos** *de uma base de dados relacional*

4.4.2 Criação de bases de dados e de tabelas em SQL

A figura seguinte ilustra a expressão de SQL[15] para definir a tabela **Adega**. Repare-se que os tipos de dados que podem ser utilizados são exclusivamente os que o sistema já tem pré-definidos. As declarações de tipos de dados mais gerais, tal como existentes nas

[15] A norma SQL não aceita o símbolo "#" nos nomes de colunas. No entanto devido à facilidade de leitura será aqui utilizado à semelhança de [Date 1997], e de acordo com as convenções em **Error! Reference source not found.**.

linguagens de programação, só serão suportadas pela norma SQL-3. Esta norma permitirá um suporte muito mais simples a aplicações orientadas por objectos, embora com modelos de interrogação mais elaborados.

```
CREATE TABLE adega
     (   #caixa                 CHAR(5)     NOT NULL,
         nomevinho              CHAR(30)    NOT NULL,
         nomecurtoprodutor      CHAR(30),
         ano                    NUMERIC(4),
         ngarrafas              NUMERIC(2),
         dlitros                NUMERIC(2),
         coretipo               CHAR(20),
         prazo                  NUMERIC(4),
         #estante               CHAR(5),
     PRIMARY KEY     ( #caixa ),
     UNIQUE          ( #estante )                 );
```

Figura 4.20 Definição SQL da Tabela **Adega** *apresentada na Figura 4.1*

Tipos de dados base em SQL

A linguagem SQL admite os tipos de dados indicados na figura seguinte, com algumas variantes ao nível de grafia, e com os significados intuitivos.

```
CHARACTER [VARYING](n)     ou   CHAR [VARYING](n)
BIT [VARYING](n)
NUMERIC(p,q)
DECIMAL(p,q)                    DATE
INTEGER                         TIME
SMALINT                         TIMESPAN
FLOAT(p)                        INTERVAL
```

Figura 4.21 Tipos de dados SQL

Gramática formal de SQL em BNF

A linguagem SQL obedece a um conjunto de regras precisas, quer ao nível da sintaxe quer da semântica. A norma de SQL pode ser vista como a definição da gramática formal de SQL, isto é:

- Um vocabulário de símbolos primitivos (tais como: `CREATE TABLE`, `CHAR`, `SELECT`, `CHARACTER`) e de regras de construção de identificadores que podem ser usados (tais como nomes válidos para relações ou atributos) – o **léxico** de SQL.

- Um conjunto de regras definindo a forma que uma expressão de SQL pode tomar - a **sintaxe** de SQL. O Anexo B apresenta a sintaxe formal da linguagem SQL-2 (ou SQL-92).
- O significado de cada regra de sintaxe, ou seja, o resultado de aplicar ou utilizar uma dada expressão de SQL - a **semântica** de SQL.

O significado de uma dada expressão de SQL é assim obtido a partir da semântica definida para a regra de SQL que é utilizada para construir essa expressão. Em termos abstractos, a álgebra relacional e o cálculo relacional permitem definir e depois determinar *automaticamente* este significado, independentemente de uma implementação particular de SQL.

As regras de sintaxe de SQL são apresentadas com definições num formalismo designado *Forma Normal de Backus-Naur* ou BNF (*Backus-Naur Form*). As definições seguintes para as operações de SQL podem parecer complicadas para quem nunca viu uma expressão de SQL ou de uma linguagem de programação apresentada como uma *gramática formal* em BNF. Aqui ficam algumas observações que podem ajudar a entender as definições:

- O símbolo "`::=`" pode ser lido "é definido como:".
- As palavras em letras maiúsculas devem ser escritas exactamente assim.
- As palavras ou frases nominais em minúsculas, escritas entre símbolos "`<`" e "`>`" estão definidas numa outra regra da gramática (ou já foram definidas, ou serão definidas mais adiante). Por exemplo, a primeira cláusula da expressão "`SELECT`" é uma *expressão de interrogação*; haverá uma definição na gramática indicando o que pode ser uma *expressão de interrogação*.
- As expressões em itálico devem ser substituídas por nomes de objectos utilizados na base dados, tais como nomes de tabelas ou nomes de colunas. Letras em itálico normalmente devem ser substituídas por números intereiros. Por exemplo `NUMERIC(P,Q)` pode ser substituído por `NUMERIC(7,2)`.
- Diferentes opções de definição para uma mesma expressão são indicadas pela barra vertical "`|`" (que se pode ler "ou").
- Expressões entre parênteses rectos "`[`" e "`]`" são opcionais.
- Expressões entre parênteses curvos "`{`" e "`}`" têm obrigatoriamente de ser utilizadas. Se há uma série de expressões entre parênteses curvos separadas por barras verticais, pelo menos uma delas tem de ser utilizada.
- A sequência especial "`[, ... n]`" indica que a cláusula anterior pode ser repetida mais do que uma vez, se tal for desejado.

Na prática, cada implementação de SQL obedece a uma gramática formal que difere da norma, mas é apresentada de uma forma semelhante utilizando BNF para a sintaxe e alguma forma de álgebra relacional ou de cálculo relacional para a semântica.

Na Figura 4.22 apresenta-se a definição do `SELECT` do SQL Server 2000 (de acordo com [Litwin *et al* 2000]). Chama-se a atenção para que as definições gramaticais aqui apresentadas apenas incluem a sintaxe. A semântica formal requer meios expressivos mais complexos, nomeadamente o recurso a técnicas de semântica denotacional, pelo que normalmente o

significado de um expressão é apresentado em linguagem natural, recorrendo a noções precisas de álgebra relacional (tal como foi feito anteriormente para a semântica da operação de junção natural, recorrendo ao produto cartesiano).

Por exemplo uma expressão `SELECT` em SQL pode significar em simultâneo operações de junção, projecção e restrição da álgebra relacional:

- Os atributos que se seguem imediatamente à palavra `SELECT` correspondem aos atributos de projecção.
- As relações referidas imediatamente a seguir à palavra `FROM` correspondem às relações objecto de operação. No caso de se tratar de uma só operação de projecção, será referida apenas uma relação.
- A condição que surge imediatamente após a palavra `WHERE` corresponde à condição de restrição.

Os exemplos de consultas SQL baseadas em `SELECT` das Figura 4.23 a Figura 4.26 mencionam as operações de álgebra relacional em que se baseiam.

```
<expressão de selecção> ::=
    <expressão de consulta>
    [ ORDER BY
         { expressão de ordenação | posição de coluna [ASC|DESC] }
            [,...n] ]
    [ COMPUTE
         { { AVG | COUNT | MAX | MIN | SUM } (expressão) } [,...n]
         [ BY expressão [,...n] ] ]
    [ FOR BROWSE ]
    [ OPTION ( <pista para consulta> [,...n] ) ]

<expressão de consulta> ::=
    { <especificação de consulta> | ( <expressão de consulta> ) }
    [ UNION [ALL] <especificação de consulta>
         |( <expressão de consulta> ) [...n] ]

<especificação de consulta> ::=
    SELECT [ ALL | DISTINCT ]
    [ {TOP inteiro | TOP inteiro PERCENT } [ WITH TIES ] ]
         <lista de selecção>
         [ INTO nova tabela]
         [ FROM {<tabela de origem> } [,...n] ]
         [ WHERE <condição de pesquisa> ]
         [ GROUP BY [ALL] expressão de agrupamento [,...n]
            [ WITH { CUBE | ROLLUP } ]          ]
         [ HAVING <condição de pesquisa> ]
```

```
<lista de selecção> ::=
    { *  | { nome de tabela | nome de vista | atalho de tabela }.*
         | { nome de coluna | expressão | IDENTITYCOL | ROWGUIDCOL }
         [ [AS] atalho de coluna ]
         | atalho de coluna = expressão } [,...n]

<tabela de origem> ::=
    nome de tabela [ [AS] atalho de tabela ]
    [ WITH ( <pista de tabela> [...n] ) ]
    | nome de vista [ [AS] atalho de tabela ]
    | função linhas [ [AS] atalho de tabela ]
    | tabela derivada [AS] atalho de tabela
         [ (atalho de coluna [,...n] ) ]
    | <tabela juntada>

<pista de tabela> ::=
    { INDEX (valor de indexação [,...n])
    | FASTFIRSTROW      | HOLDLOCK        | NOLOCK
    | PAGLOCK           | READCOMMITTED   | READPAST
    | READUNCOMMITTED   | REPEATABLEREAD  | ROWLOCK
    | SERIALIZABLE      | TABLOCK         | TABLOCKX
    | UPDLOCK  }

<tabela juntada> ::=
    <tabela de origem > <tipo de junção> <tabela de origem > ON
    <condição de pesquisa>
    | <tabela de origem > CROSS JOIN <tabela de origem >
    | <tabela juntada>

<tipo de junção> ::=
    [ INNER | { { LEFT | RIGHT | FULL } [OUTER] } ]
    [ <pista de junção> ]
    JOIN

<condição de pesquisa> ::=
    { [ NOT ] <predicado> | ( <condição de pesquisa> ) }
    [ {AND | OR} [ NOT ] { <predicado> | ( <condição de pesquisa> ) } ]
    } [,...n]
```

```
<predicado> ::=
    { expressão { = | <> | != | > | >= | !> | < | <= | !< } expressão
    | expressão de string [ NOT ] LIKE expressão de string
    [ ESCAPE 'caracter de escape']
    | expressão [ NOT ] BETWEEN expressão AND expressão
    | expressão IS [ NOT ] NULL
    | CONTAINS
    ( { coluna | * }, '<contém condição de pesquisa>' )
    | FREETEXT ( { coluna | * }, 'string de texto livre' )
    | expressão [ NOT ] IN ( sub-consulta | expressão [,...n] )
    | expressão { = | <> | != | > | >= | !> | < | <= | !< }
    { ALL | SOME | ANY } ( sub-consulta )
    | EXISTS ( sub-consulta ) }

<pista para consulta> ::=
    { { HASH   | ORDER } GROUP
    | { CONCAT | HASH | MERGE } UNION
    | { LOOP   | HASH | MERGE } JOIN
    | FAST número de linhas
    | FORCE ORDER
    | MAXDOP número
    | ROBUST PLAN
    | KEEP PLAN }
```

Figura 4.22 A sintaxe em BNF da expressão `SELECT` no SQL Server 2000

4.4.3 Consulta e actualização em SQL

Um vez definida a base de dados podem utilizar-se os operadores `SELECT`, `INSERT`, `UPDATE` e `DELETE` para implementar consultas e actualizações de informação. Nos exemplos seguintes será utilizada a base de dados apresentada na Figura 4.18 e Figura 4.19 para ilustrar algumas operações em SQL para consulta e actualização de base de dados. Obviamente que se espera que os exemplos permitam entender a semântica (significado) das operações SQL apresentadas.

Consultar informação

Consulta:

```
SELECT nomevinho, cor, região
FROM Vinhos
WHERE regiao='Alentejo' AND cor='tinto'
```

Resultado:

nomeVinho	*cor*	*região*
Tapada do Chaves	Tinto	Alentejo
Quinta do Carmo	Tinto	Alentejo
Dom Martinho	Tinto	Alentejo

Figura 4.23 Vinhos tintos do Alentejo (operações de projecção e restrição)

Consulta:

```
SELECT *          ou      SELECT Vinhos.*          ou      TABLE Vinhos
FROM Vinhos               FROM Vinhos
```

Nota: o símbolo "*" significa *todos* os atributos ou colunas.

Resultado:

#Vinho	*nomeVinho*	*cl*	*cor*	*região*	*#Produtor*
1	Barca Velha	75	Tinto	Douro	3
2	Foral Grande Escolha	75	Tinto	Douro	1
33	Morgadio da Torre	75	Branco	Vinho Verde	16
...	...	...	...	...	...
789	Malvazia	75	Aloirado	Vinho da Madeira	21

Figura 4.24 Toda a informação sobre Vinhos (operação de projecção)

Consulta:

```
SELECT #loja, iploja
FROM lojas
```

Resultado:

#Loja	*ipLoja*
45	7
76	8
...	...
91	16

Figura 4.25 Endereços ip das lojas (operação de projecção)

Consulta:

```
SELECT nomevinho, cor, nomeCurtoProdutor
FROM Vinhos, Produtores
WHERE Vinhos.#produtor=Produtores.#produtor
```

Resultado:

nomeVinho	*cor*	*nomeCurtoProdutor*
Barca Velha	Tinto	Ferreira
Foral Grande Escolha	Tinto	Caves Aliança
Morgadio da Torre	Branco	Sogrape
Redoma	Rosé	Niepoort Vinhos
Reserva	Tinto	Sogrape
Reserva	Branco	Sogrape
Charamba	Branco	Quinta da Aveleda
Quinta de Azevedo	Branco	Sogrape
Quinta da Leda	Tinto	Ferreira
Dorna Velha	Tinto	Quinta do Silval
Vallado	Tinto	Quinta do Valado
...	...	...
Tuella	Branco	Cockburn's
Vintage Quinta da Leda	Tinto	Ferreira
Malvazia	Aloirado	Barbeito

Figura 4.26 Produtores dos Vinhos com indicação da cor (operações de junção e projecção)

Actualizar informação

Introdução:

```
INSERT  INTO VinhosTintos (#Vinho, nomevinho)
        SELECT #Vinho, nomevinho
        FROM vinhos
        WHERE cor='tinto';
```

Figura 4.27 Colocação dos códigos e nomes de vinhos tintos numa tabela **VinhosTintos**{#Vinho, nomeVInho}, *previamente criada*

Alteração:

```
UPDATE  produtores
SET  tipoprodutor = tipoprodutor + 10
WHERE sedeprodutor = 'funchal';
```

Figura 4.28 Actualização dos produtores com sede no Funchal

Eliminação:

```
DELETE FROM   produtores
WHERE tipoprodutor < 25;
```

Figura 4.29 Eliminação dos produtores cujo tipo é inferior a 25

4.4.4 Limitação do SQL para obter o fecho transitivo

A linguagem SQL permite a realização de um número considerável de consultas sobre bases de dados, baseando-se numa filosofia de interacção pergunta-resposta, onde qualquer resposta é entendida como uma relação (ou tabela), sendo no caso limite uma relação com apenas um atributo e um valor (tabela com uma coluna e uma linha). Como este modo de funcionar da linguagem era visto como essencial para a sua utilização por pessoas sem grnade experência informática, não se permitiu desde o início a inclusão do conceito de iteração ou de recursão. Apenas o conceito de substituição está presente, pois em qualquer consulta uma relação pode ser substituída por uma outra consulta (em termos da construção de consultas em SQL diz-se que as operações podem ser *aninhadas* umas dentro das outras).

Esta decisão de concepção de SQL não permite determinar, no caso geral, o que se designa por *fecho transitivo de uma relação*. A linguagem SQL que normalmente está disponível, em particular a que está normalizada com base na álgebra ou no cálculo relacional, por definição relacionalmente completa, não permite obter o fecho transitivo de uma relação ou de um conjunto de relações que definam uma estrutura em árvore potencialmente com um número ilimitado de níveis.

Considere-se por exemplo o caso de uma consulta destinada a obter todos os componentes e sub-componentes que um determinado produto necessita para a sua produção, até ao nível atómico, isto é, componentes que não têm sub-componentes. Essa informação pode estar caracterizada numa única tabela **PC (Produtos-Componentes)**, da forma **PC**{*#componente, #sub-componente, no-sub-componentes*}. Nesta tabela a chave primária é dupla, e o atributo *#sub-componente* é ainda uma chave alheia sobre o atributo *#componente* da própria tabela.

Exercício: Defina valores para esta tabela, por exemplo com 3 produtos de códigos 987, 765, 087. Cada produto tem vários componentes, cada um dos quais também tem um código. Toda a estrutura de composição desses componentes e sub-componentes está assim definida nesta tabela. Pode haver duas situações: existe um nível a partir do qual se sabe à partida que não há mais componentes, ou pode haver necessidade de suportar qualquer estrutura e nível de composição de produtos. Qual destas situações seria possível tratar em SQL? Ilustre as duas situações.

4.4.5 Introdução à optimização de consultas a Bases de Dados

A eficiência à resposta a operações descritas em SQL depende em grande medida da qualidade do optimizador de SQL. Sobre este tópico ver por exemplo [Date 1997]. Há uma série de estratégias óbvias que o optimizador de SQL deve seguir. Por exemplo, devem ser aplicadas em primeiro lugar as restrições e só depois as junções.

Praticamente todas as estratégias e técnicas de optimização clássica e não clássica são aplicáveis a este problema, incluindo os algoritmos de caminho mais curto, heurísticas, algoritmos genéticos e outras meta-heurísticas.

4.5 NFD – Normalização Funcional de Dados para Concepção de BDr

Um problema de gestão de informação implica normalmente a definição de um arquivo de dados e a implementação de dois **tipos de operações** ou processos: os que actualizam os dados e os que não actualizam os dados. Os processos que modificam dados incluem introdução, alteração e eliminação de informação. Os processos que não modificam dados incluem operações de consulta e pesquisa. A tipificação prévia das operações ou funções a realizar sobre a base de dados[16], bem como a indicação prévia da respectiva eficiência e tempo máximo de resposta permitido pelo utilizador, condicionam significativamente a forma de organizar as tabelas da base de dados. Em sistemas reais qualquer organização de tabelas resulta de compromissos de engenharia envolvendo decisões sobre custos de equipamentos e sistemas, tempos de desenvolvimento e tempos de resposta. No entanto, erros graves de organização de tabelas da base de dados podem tornar a resposta do sistema demasiado lenta por mais recursos que se disponham para a plataforma física.

Esta secção introduz os conceitos necessários para discutir as alternativas de organização que as tabelas de uma base de dados relacional podem ter. Em particular, introduzem-se os conceitos de **primeira forma normal** (**1FN**) de uma relação, **granulosidade** da operação sobre uma base de dados, **dependência funcional** entre conjuntos de atributos de uma relação, **chaves candidatas** de uma relação, **chave primária** de uma relação, **chaves alheias** de uma relação, **segunda forma normal** (**2FN**) de uma relação, **terceira forma normal** (**3FN**) de uma relação, ou em geral, **formas normais** de relações. Introduz-se para tal o processo de normalização funcional de relações, ou simplesmente, **normalização**.

4.5.1 Primeira forma normal

A possibilidade de actualização e pesquisa eficiente sobre grandes volumes de informação de alguns tipos variados, requer a sua organização prévia em tabelas segundo o modelo relacional. Isto significa na prática que a informação se deve distribuir por tabelas bidimensionais, onde cada tabela tem um número pequeno e à partida determinado de colunas, e em cada célula ou campo existe um dado atómico que pode ser processado por uma só operação.

Diz-se que uma relação ou um conjunto de relações está em **primeira forma normal** no caso de a condição anterior se verificar. Só desta forma é possível de uma forma sistemática e eficiente relacionar ou "cruzar" dados de duas tabelas que partilhem um atributo do mesmo tipo. De facto para um base de dados se designar relacional todas as suas tabelas devem estar em primeira forma normal (conforme Secção 4.4).

[16] Designa-se **requisito funcional** qualquer requisito que tenha sido colocado sobre o sistema relativamente à validade e consistência dos resultados a obter. Por exemplo, um sistema que permita calcular o valor dos impostos a pagar por uma pessoa singular, tem como requisitos funcionais a legislação em vigor. Muitos sistemas podem cumprir os mesmos requisitos funcionais recorrendo a equipamentos ou linguagens de programação distintas e oferecendo tempos de resposta diferentes. Designa-se **requisito não funcional** qualquer outro tipo de requisito, tal como a plataforma, custo ou tempo de desenvolvimento, a norma de documentação a respeitar ou o tempo de resposta que cada operação pode ter.

Por exemplo não é possível, de uma forma suportada automaticamente por um SGBD relacional, cruzar com base nos atributos *nomeDoVinho* e *nomesDosVinhos* a informação das tabelas da Figura 4.1 e da Figura 4.4. Nesta última tabela cada campo do atributo *nomesDosVinhos* contém vários nomes de vinhos, não estando por isso a tabela normalizada em primeira forma normal.

4.5.2 Granularidade de uma operação

Como se viu, uma base de dados deve suportar diversos tipos de operações, em particular operações de consulta ou pesquisa sobre uma ou mais tabelas e operações de actualização de dados como sejam inserção, alteração ou eliminação.

Estas operações podem ainda envolver a manipulação de partes significativas da base de dados, ou apenas ter um impacto muito localizado. Considere-se, por exemplo, a operação global para determinar uma listagem com os valores médios mensais de depósitos efectuados pelos clientes de um banco nos últimos 5 anos, listagem essa com valores médios para cada agência bancária e para cada região geográfica (freguesia) de endereço de residência dos seus clientes[17]. Esta operação pode envolver consultas a milhões de registos da base de dados. Alternativamente, uma operação pode envolver apenas a manipulação de um único campo ou registo. Por exemplo, a alteração do número de telefone de um cliente, ou a consulta do saldo de uma conta pessoal.

Diz-se que a **granulosidade** da operação sobre uma base de dados pode ser mais **fina** ou mais **grossa** dependendo da número de registos da base de dados que são envolvidos nessa operação.

4.5.3 Eficiência de uma operação e tempo de resposta

Uma boa organização prévia das tabelas da base de dados é um dos requisitos fundamentais e essenciais para obter um bom desempenho das operações. Um outro requisito, cuja discussão vai para além do objectivo desta exposição, é a criação e manutenção de índices de ordenação e pesquisa nos atributos ou conjuntos de atributos das tabelas que sejam envolvidos nas operações previstas, e que normalmente incluem pelo menos as chaves primárias e as chaves alheias de todas as tabelas.

Como é fácil de concluir, a organização de uma base de dados destinada exclusivamente a consulta de informação, onde nunca fosse necessário actualizar informação, deveria ser completamente diferente da organização de uma base de dados destinada a actualizar informação. Numa situação normal, em que a base de dados deve possibilitar os vários tipos de operações de uma forma eficiente, torna-se necessário estabelecer compromissos ao nível da organização prévia. Como se verá de seguida com mais detalhe, uma base de dados com boas propriedades para suportar operações de actualização eficientes deve por exemplo ter os dados a actualizar situados num único local, com baixa redundância. Pelo contrário, uma base de dados para permitir consultas eficientes de granulosidade grossa deve ter os dados repetidos, com alguma redundância, desde que obviamente esteja à partida assegurada a sua consistência. Por exemplo uma operação de consulta que envolva cruzar, no momento da operação, informação proveniente de duas tabelas, será claramente mais lenta do que a mesma

[17] Tente, como exercício, criar uma listagem exemplo da resposta desejada a esta consulta.

operação sobre uma só tabela onde previamente se tenha o resultado da junção das duas tabelas. Esta tabela, que seria pré-calculada, teria um nível alto de redundância, o que dificultaria operações de actualização respeitando a consistência.

O discussão dos compromissos e das alternativas de organização da base de dados relacional é simplificada com a introdução da noção de **dependência funcional** e do processo de **normalização funcional**, apresentados e discutidos nas secções seguintes.

4.5.4 Bases de Dados para Produção e Bases de Dados para Apoio à Decisão

Numa situação real, conforme se pode ver na Figura 4.30, o sistema de informação de uma organização incluirá normalmente duas bases de dados: a **base de dados de produção** ou operacional, destinada a operações de actualização e consulta de granulosidade muito fina, e a **base de dados de apoio à decisão** destinada a operações de consulta de granulosidade menos fina ou grossa. Esta última base de dados é destinada à realização de estudos, e tem de ser actualizada com frequência adequada a partir da base de dados de produção operacional, podendo conter ainda informação adicional (por exemplo, data da última actualização).

A base de dados de apoio à decisão tem uma organização diferente da de produção, e destina-se também a constituir um **armazém de dados** (“data warehouse”) históricos relevantes da empresa, podendo integrar na prática informação de várias origens, para além da que consta da base de dados operacional.

Um motivo adicional para a criação e manutenção dos armazéns de dados é o factor de carga que a realização de operações de análise e estudo sobre a base de dados operacional colocaria, podendo afectar irremediavelmente o desempenho das operações simples. Tipicamente uma transacção de consulta de saldo ou uma transacção de transferência de fundos entre duas contas sobre a base de dados operacional (efectuada em simultâneo com milhares de outras transacções do mesmo tipo) não deverá demorar mais do que quatro ou cinco segundos (tal como o utilizador se apercebe). A realização de operações de granulosidade grossa iria na prática afectar irremediavelmente o desempenho das outras operações, podendo inclusivamente exigir o bloqueio para actualização de partes importantes da base de dados.

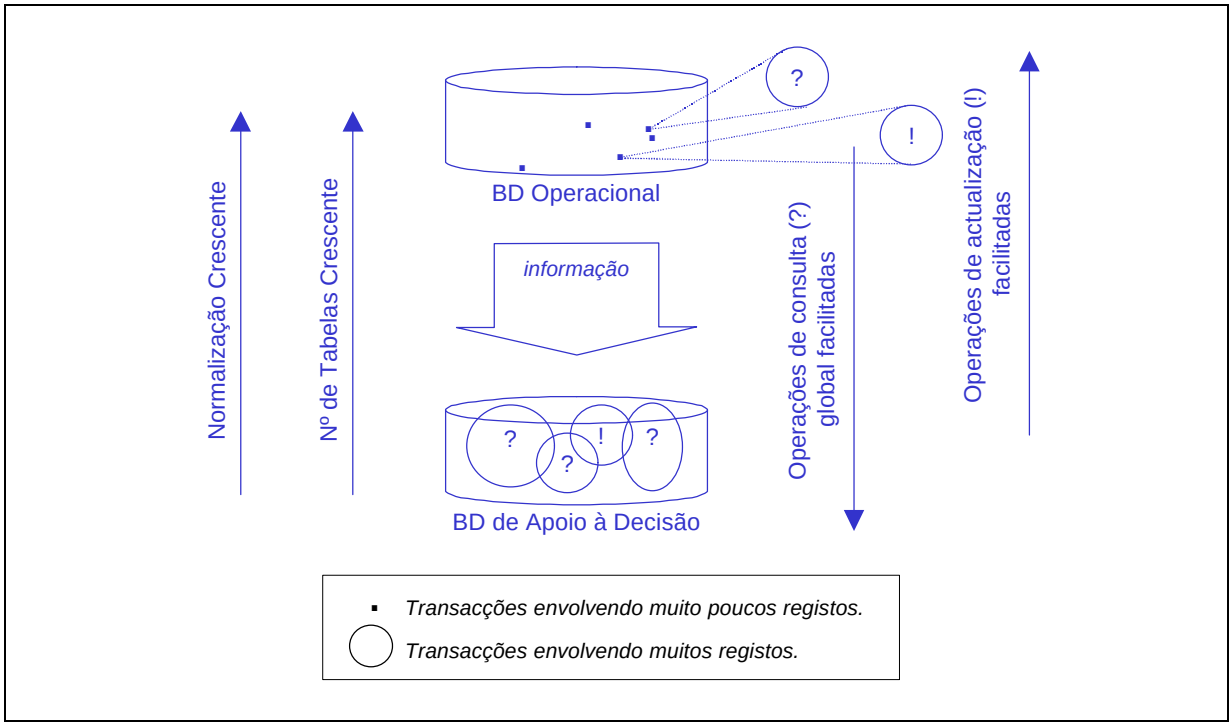


Figura 4.30 Normalização de Relações em Sistemas Operacionais e em Sistemas de Apoio à Decisão – Armazéns de Dados

A estrutura de organização da informação nas tabelas de base de dados relacional de produção resulta normalmente de um processo de normalização que coloca todas as tabelas em terceira forma normal. Pelo contrário, a estrutura de organização da informação num armazém de dados corresponde a um conjunto pequeno de tabelas que respeitam as condições mínimas do modelo relacional, ou seja as tabelas estão apenas em primeira forma normal.

4.5.5 Dependência Funcional, Chave Primária e Chave Alheia

O conceito de **dependência funcional** tem uma importância crucial na discussão das questões de organização e projecto de bases de dados, em particular na teoria da normalização de informação. A sua introdução é simples, mas para a discussão plena do seu impacto é necessário compreender bem as seguintes noções: relação, função, chave candidata, chave primária, chave alheia ou chave estrangeira (traduções de “foreign key”).

O processo de normalização de relações ou tabelas surgiu como forma de responder formalmente ao problema da repetição de informação numa base de dados. A existência da mesma informação em mais do que um registo ou em mais do que um campo da base de dados pode criar problemas graves para as operações de actualização. Se a mesma informação existir em vários locais, é fundamental que os processos de actualização dessa informação atinjam sempre todos esses locais, sob pena de a base de dados ficar inconsistente e de as operações de consulta retornarem eventualmente informação errada. Este problema é particularmente grave em relação a informação externa à base de dados, tal como nomes de pessoas, endereços, telefones, preços de produtos, ou códigos de bilhete de identidade, informação esta sujeita a erros de introdução e necessidade de permanente actualização ao longo do tempo. Numa base de dados completamente normalizada qualquer dado externo deveria estar guardado em apenas um único local. Como limite, até os valores das cores

("azul" ou "amarelo") ou os nomes das cidades ("Porto", "Bissau" ou "Londres"), sendo susceptíveis de tradução, deviam surgir em apenas um único registo de uma única tabela da base de dados.

Embora pareça intuitivamente simples, a identificação do que é *informação repetida* não resulta de um processo trivial, dependendo fundamentalmente da semântica ou significado dos nomes dos vários objectos informativos. Nas situações reais as pessoas estão habituadas a utilizar as palavras, dominam os seus significados e regras de utilização, bem como as estruturas por vezes complexas de excepções. Esta semântica deve tornar-se objectiva, para ser discutida sem ambiguidades, e como se verá pode ser capturada em grande parte com recurso a dependências funcionais, e a outras restrições de integridade sobre os atributos de informação (como se verá de seguida). Para tal é relevante analisar o significado dos vários atributos de informação, as convenções de codificação que o problema e os sistemas reais utilizam, o âmbito de utilização desses sistemas, e a longevidade que se espera que venham a ter, com a natural evolução de significados.

A análise objectiva e sem ambiguidades da informação deve ser feita utilizando noções formais, por exemplo recorrendo às noções de *valor*, *conjunto*, *predicado*, *atributo*, *relação* e *função*.

Do ponto de vista da teoria dos SGBD relacionais, uma tabela corresponde à tabulação de uma relação definida num espaço de domínios de valores atómicos ou escalares que podem ser representados e manipulados num sistema informático. Nos SGBDr comerciais existentes estes domínios de valores incluem os seguintes tipos de dados: INTEGER, CHAR(N), DATE, NUMERIC(N), DECIMAL(N) (ver também secção 4.4.1, sobre SQL).

Note-se, como se viu, que as relações e as tabelas que são utilizadas no modelo relacional não têm uma ordem estabelecida para os atributos em que estão definidas.

Exemplo 4.3: O Arquivo de Identificação – caso simples

> "O Arquivo de Identificação gere os números de Bilhete de Identidade e os Nomes de todos os cidadãos".

Numa visão simples do problema, o Arquivo de Identificação irá manter actualizada a relação com todos os nomes e números de Bilhete de Identidade. Essa relação pode ser vista de várias formas. Alguns dos elementos estão representados na Figura 4.31, e tabelados na Figura 4.33. Este problema simples obedece a um conjunto de *regras de integridade*. Por exemplo, um mesmo número de Bilhete de Identidade não pode num dado momento relacionar-se com mais do que um nome de pessoa. No entanto é possível e mesmo necessário que duas pessoas com nomes iguais tenham números de Bilhete de Identidade diferentes.

Pode colocar-se a questão de saber como é que um SGBD garantirá de uma forma simples e automática que estas regras sejam cumpridas. Por exemplo, uma tentativa de introduzir um número de BI já existente com um novo nome não deveria ser permitido. Como se verá, esta funcionalidade poderia ser suportada, através da designação do número de BI como *chave primária* da relação.

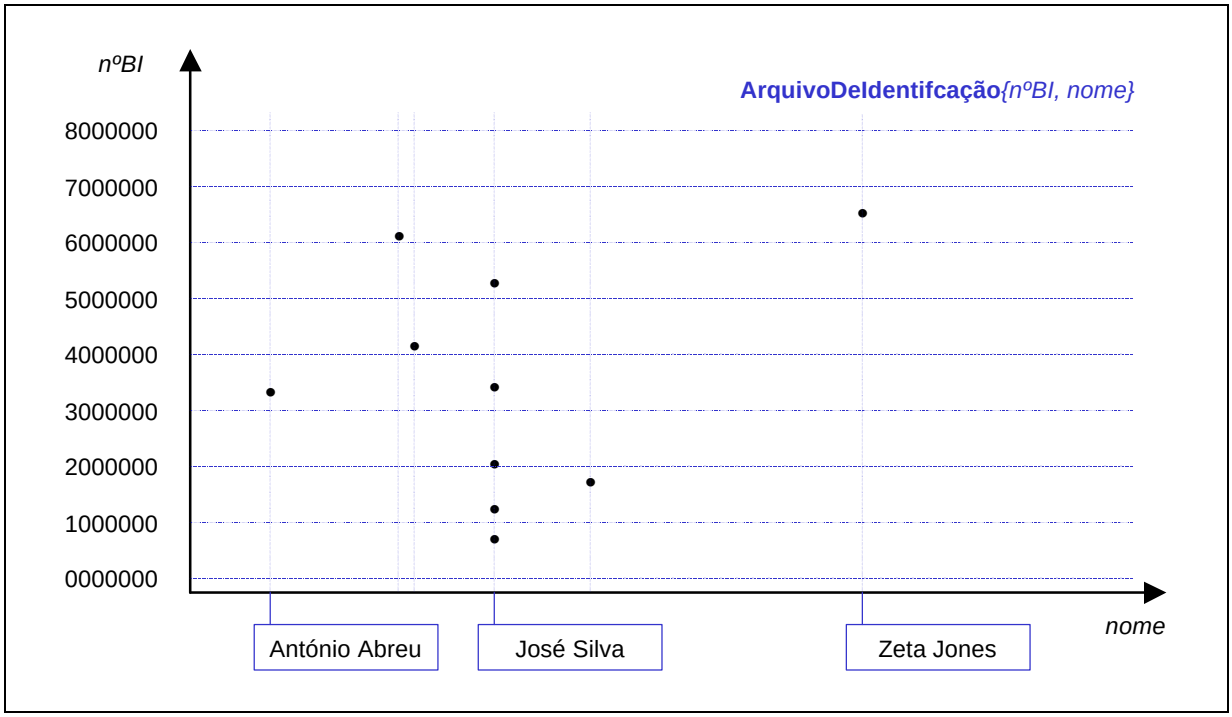


Figura 4.31 Representação num plano cartesiano da relação entre nomes e números de Bilhete de Identidade

No caso simples da relação apresentada na Figura 4.31, definida entre os domínios dos atributos *nºBI* e *nome*, é fácil verificar que existe uma função de *nºBI* para *nome*. No entanto não existe uma função de *nome* para *nºBI*. Como se verá de seguida, as funções existentes numa dada relação são identificadas ou definidas como **dependências funcionais** entre atributos dessa relação. Para o caso simples anterior, esta função de *nºBI* para *nome* seria representada como se indica na Figura 4.32.

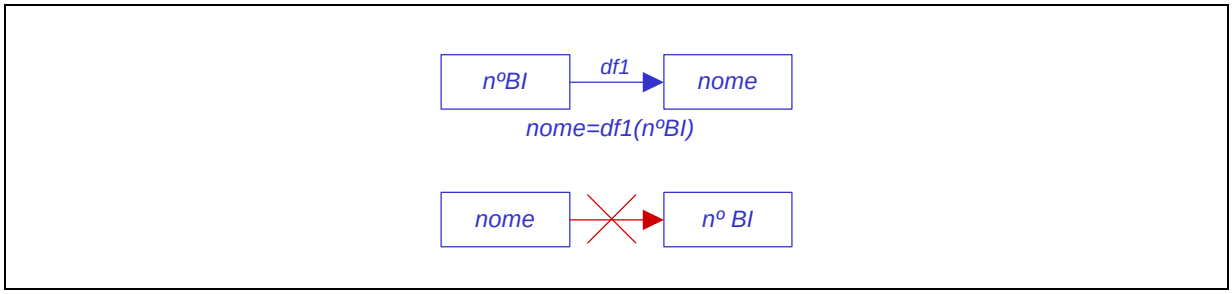
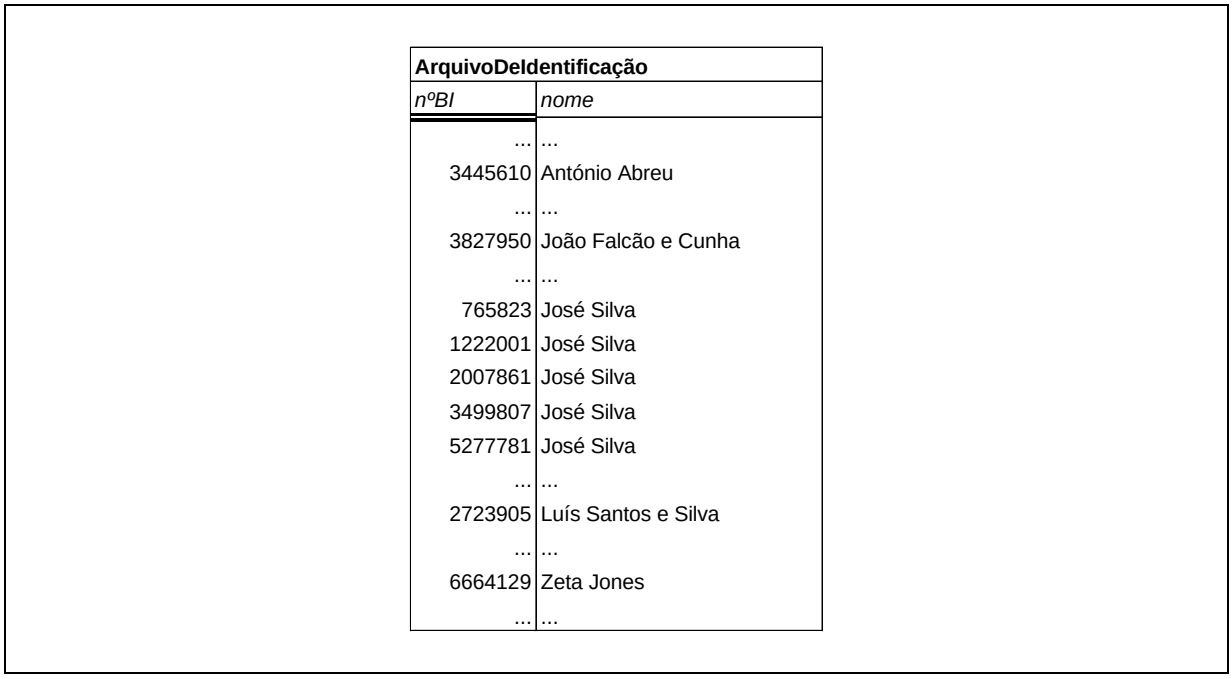


Figura 4.32 Função df1 *ou dependência funcional* df1 *que se pode identificar na relação* **ArquivoDeIdentificação**



ArquivoDeIdentificação

nºBI	*nome*
...	...
3445610	António Abreu
...	...
3827950	João Falcão e Cunha
...	...
765823	José Silva
1222001	José Silva
2007861	José Silva
3499807	José Silva
5277781	José Silva
...	...
2723905	Luís Santos e Silva
...	...
6664129	Zeta Jones
...	...

Figura 4.33 Representação numa tabela da relação entre nomes e números de Bilhete de Identidade

O caso simples anterior esconde alguma complexidade real. O Arquivo vai ter de considerar a possibilidade de uma pessoa mudar de nome, sem alterar o seu número de BI, por exemplo na sequência de um casamento. A manutenção da informação ao longo do tempo requer assim que o Arquivo de Identificação mantenha na sua base de dados uma tabela mais complexa do que a apresentada.

Define-se de seguida formalmente a noção de dependência funcional. A representação gráfica das dependências funcionais presentes num dado problema, ou num dado conjunto de relações, é feita de acordo com as normas introduzidas de seguida. A representação gráfica das dependências funcionais é designada Diagrama de Dependências Funcionais ou DDF.

Definição 4.6 ***Dependência Funcional (DF)***

Seja R uma variável de relação[18] e sejam X e Y dois subconjuntos do conjunto de atributos de R: $X=\{x_1, ..x_n\}$, $Y=\{y_1, ..y_m\}$. Diz-se que Y é funcionalmente dependente de X, ou que X determina funcionalmente Y, escrevendo-se $X \rightarrow Y$ (X seta Y), se e só se, para todo e qualquer valor válido de R, a cada valor distinto dos atributos de X estiver associado precisamente um só valor para cada atributo de Y. Nos casos de $n=1$ ou $m=1$ pode escrever-se apenas $x_1 \rightarrow Y$, $X \rightarrow y_1$ ou $x_1 \rightarrow y_1$.

[18] Para clarificar a noção de "variável de relação" observe-se o seguinte exemplo. Na Figura 4.35 há duas tabelas, ou seja dois valores possíveis para a relação **FV**. O identificador "FV" tem assim duas leituras: pode significar uma tabela concreta, num dado momento, ou pode significar uma variável que pode assumir vários valores, por exemplo os dois valores da figura referida. No primeiro caso utiliza-se **FV**, no último caso *FV*, como variável de relação.

O conjunto *X* é designado como **parte determinante** da DF (ou simplesmente **determinante** da DF) e *Y* como **parte dependente** da DF.

Uma DF diz-se **trivial** se e só se a sua parte direita (ou parte dependente) for um subconjunto da parte esquerda (ou parte determinante). Por exemplo, as DF *x, y→x* e *x, y→x,y* são triviais.

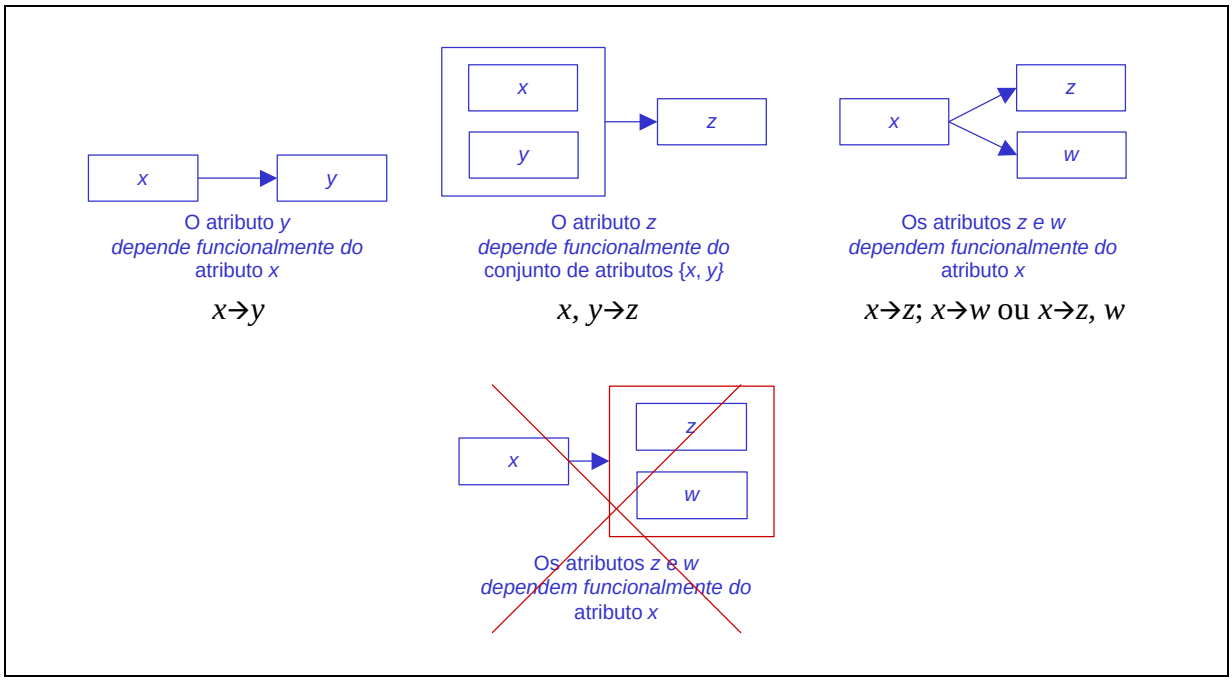


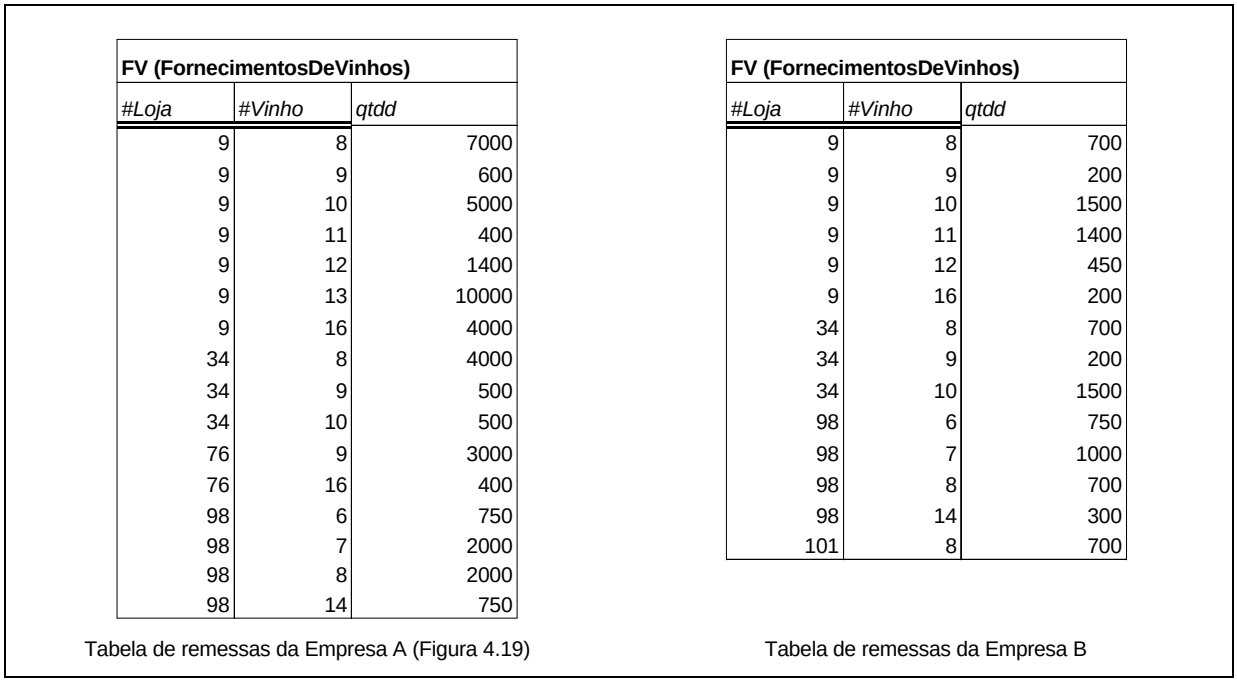
Figura 4.34 Representação correcta de dependências funcionais e exemplo de uma representação incorrecta

Basicamente uma dependência funcional, abreviada de seguida como DF, é um *relacionamento de muitos-para-um* **de** um conjunto de atributos **para** um outro atributo, no âmbito de uma dada relação. No exemplo da Figura 4.19, no caso da relação de informação sobre remessas, capturada pela tabela **FV**, existe uma dependência funcional do conjunto de atributos {*#Loja, #Vinho*} para o conjunto de atributos {*qtdd*}. Isto significa que sobre qualquer relação que seja um valor válido da variável de relação **FV** (ver Figura 4.35 para tabulação de dois valores válidos para a relação **FV**) se verifica a seguinte condição:

- Para qualquer valor *(l, v)* do par de atributos *#Loja* e *#Vinho* existe apenas um valor *q* do atributo *qtdd*: *#Loja*, *#Vinho*→ *qtdd*.

No exemplo, a identificação de tal dependência funcional significaria que a tabela **FV** capturava a informação relativa ao somatório das remessas ou fornecimentos de um dado vinho, que uma certa loja teria efectuado até ao momento.

A definição de dependência funcional apresentada permite obviamente que para valores distintos do par de atributos *#Loja* e *#Vinho* possa corresponder em geral um mesmo valor do atributo *qtdd*. Isto é, podem existir remessas de igual valor total para lojas e vinhos diferentes.



FV (FornecimentosDeVinhos)

#Loja	*#Vinho*	*qtdd*
9	8	7000
9	9	600
9	10	5000
9	11	400
9	12	1400
9	13	10000
9	16	4000
34	8	4000
34	9	500
34	10	500
76	9	3000
76	16	400
98	6	750
98	7	2000
98	8	2000
98	14	750

Tabela de remessas da Empresa A (Figura 4.19)

FV (FornecimentosDeVinhos)

#Loja	*#Vinho*	*qtdd*
9	8	700
9	9	200
9	10	1500
9	11	1400
9	12	450
9	16	200
34	8	700
34	9	200
34	10	1500
98	6	750
98	7	1000
98	8	700
98	14	300
101	8	700

Tabela de remessas da Empresa B

Figura 4.35 Exemplo de dois valores ou instanciações possíveis para a relação de remessas: estas duas tabelas poderiam existir em duas empresas que tivessem adquirido o mesmo sistema de informação.

Um dos passos mais importantes do projecto de concepção de uma base de dados relacional apresentado neste capítulo é exactamente a identificação das DF. O processo de organização da base de dados pode iniciar-se com a identificação dos atributos de informação relevantes, ou seja, pela selecção de identificadores, descrição sumária do seu significado, domínio de valores atómicos, e exemplo de valores possíveis. Esta identificação pode resultar na tabulação bidimensional de uma ou mais relações, cumprindo os 3 requisitos base do modelo relacional (relações em 1ª forma normal). As DF relevantes são exemplos de **restrições de integridade** que se desejam impor ou que existem na informação contida nas relações. A definição das DF é assim uma forma precisa de capturar o significado da informação que se pretende que a base de dados venha a gerir, permitindo ao SGBD limitar os valores a aceitar ou efectuar automaticamente a sua validação.

Por exemplo, a indicação de que qualquer relação **FV** deve respeitar a DF {*#Loja*, *#Vinho*}→{*Qtdd*}, ou simplesmente {*#Loja*, *#Vinho*}→*Qtdd*, não nos permite interpretar **FV** como o registo de todas as remessas[19]. Uma DF limita então os valores que uma relação pode ter em casos concretos.

Se cada linha da tabela **FV** representasse uma remessa individual, para cada par de valores *#Loja* e *#Vinho* poderia haver várias quantidades diferentes, ou seja, uma quantidade em cada remessa. Esta interpretação não é compatível com a DF indicada. A referida DF é por exemplo compatível com a interpretação de que *Qtdd* significa o somatório de todas as remessas enviadas ou recebidas até ao momento, de cada vinho e de cada loja (podendo haver outras interpretações compatíveis).

[19] Porquê?

A identificação das DF deve ser feita com cuidado pois deve ser resistente à evolução da informação e da base de dados ao longo do tempo. Também se deve ter cuidado ao inferir as DF a partir de exemplos concretos. Para um certo exemplo pode parecer que uma determinada DF se verifica, mas tal ser apenas circunstancial. No exemplo da Figura 4.35, a identificação das DF a partir apenas dos valores presentes na tabela da empresa B permitiria inferir erradamente que *#Vinho→Qtdd*.

É importante salientar que as regras de codificação para os valores dos atributos utilizados nos sistemas reais, em particular dos atributos códigos, influenciam de forma importante as DF que se podem ou devem identificar. Por exemplo, a inclusão no número do Bilhete de Identidade de uma pessoa de um ou mais digitos referente à actualização ou renovação do respectivo cartão iria alterar as DF adiante representadas.

Nas figuras seguintes apresentam-se várias exemplos de situações caracterizadas por diferentes estruturas de organização da informação. Cada exemplo é ilustrado com o respectivo DDF associado.

Exemplo 4.4: O Arquivo de Identificação – caso simples com códigos de contribuinte.

> "O Arquivo de Identificação gere os números de Bilhete de Identidade e os Nomes de todos os cidadãos. Além disso o Arquivo mantém ainda os códigos de contribuinte de todos os cidadãos".

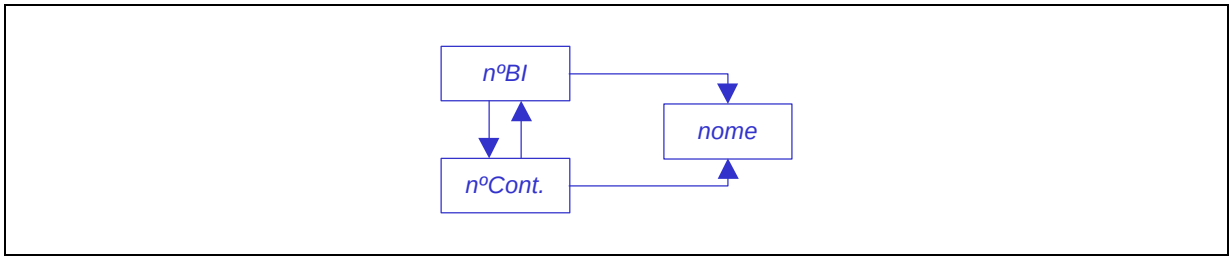


Figura 4.36 DDF do caso simples do Arquivo de Identificação com códigos de contribuinte

Exemplo 4.5: O Arquivo de Identificação – caso histórico.

> "O Arquivo de Identificação gere os números de Bilhete de Identidade e os Nomes de todos os cidadãos. O Arquivo de Identificação mantém um registo histórico com todas as alterações de nomes".

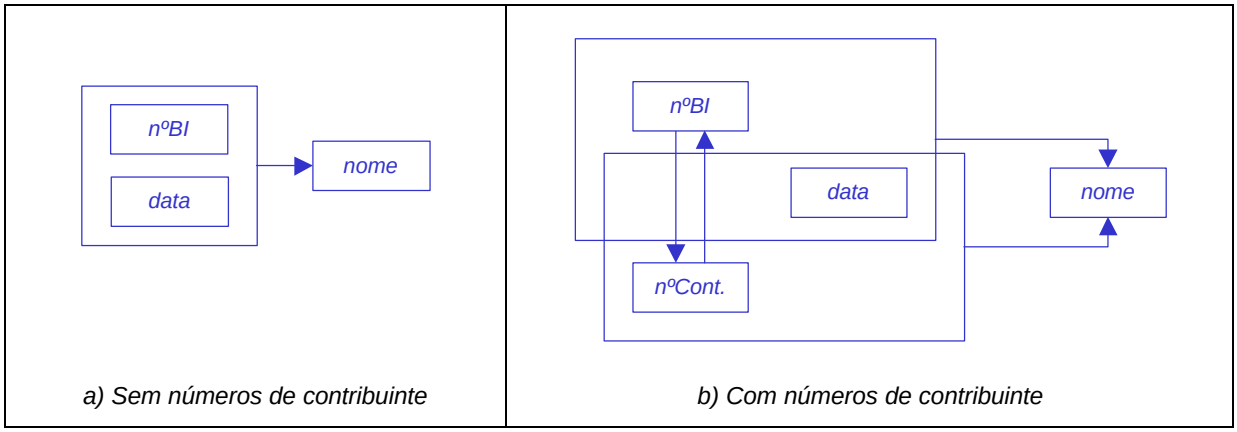


Figura 4.37 - DDF do caso histórico do Arquivo de Identificação

Nesta visão do problema, o Arquivo de Identificação irá manter uma a relação com os nomes e números de BI de todas as pessoas ao longo do tempo, registando a data de cada alteração de nomes. Sendo assim a Figura 4.31 pode ainda ser utilizada para representar a situação. No entanto já não é correcto afirmar que *nome=df1(nºBI)*. As restrições de integridade associadas a este novo problema podem ser representadas no diagrama da Figura 4.36.

Exemplo 4.6: O Arquivo de Identificação – caso histórico com código de actualização e data.

> "O Arquivo de Identificação gere os números de Bilhete de Identidade e os Nomes de todos os cidadãos. O Arquivo mantém um registo histórico com todas as alterações de nomes, e as respectivas datas de actualização. De cada vez que um BI sofre alteração do nome, é gerado um novo código de actualização sequencial para esse BI. Por exemplo: em 1963-03-15 a Senhora Maria Elisa Silva obteve o seu primeiro Bilhete de Identidade com o número 2309927 (*#Actual*=1); em 1975-09-03, e após contrair matrimónio, Maria Elisa Silva alterou o seu nome para Maria Elisa Silva e Cunha (*#Actual*=2); em 1996-09-23 Maria Elisa Silva e Cunha voltou a casar e retomou o seu nome de solteira (*#Actual*=4)".

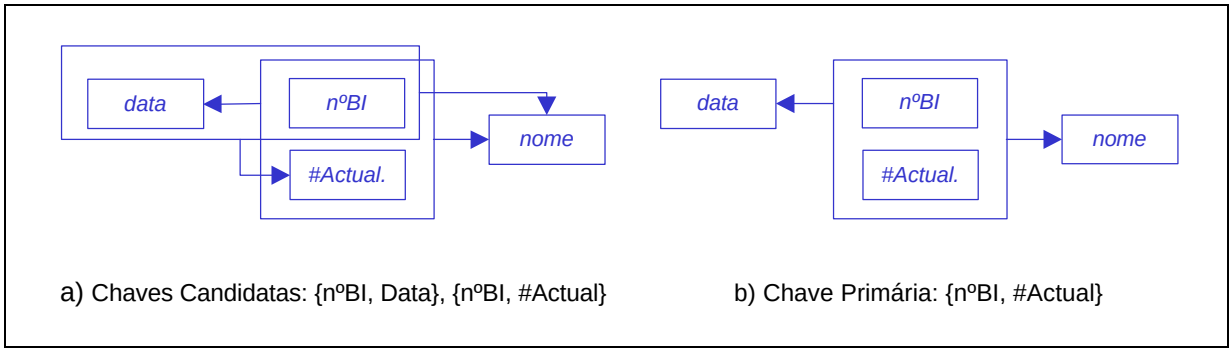


Figura 4.38 - Duas propostas de DDF do Arquivo de Identificação para o caso histórico com código de actualização e data

Exemplo 4.7: O Arquivo de Identificação – caso histórico com código de actualização único.

> "O Arquivo de Identificação gere os números de Bilhete de Identidade e os Nomes de todos os cidadãos. O Arquivo mantém um registo histórico com todas as alterações de nomes, e as

respectivas datas de actualização. De cada vez que um BI sofre alteração do nome, é gerado um novo código de actualização sequencial. Os códigos de actualização são únicos em todo o arquivo, permitindo identificar não só o número do Bilhete de Identidade como a alteração sofrida no nome".

Fica como exercício identificar as DF e desenhar o DDF para este exemplo.

Chaves candidatas e chave primária de uma relação ou tabela

Considere-se a tabela **Vinhos** ou a tabela **Produtores** dos exemplos anteriores. Nessas tabelas não devem aparecer valores repetidos respectivamente nas colunas *#Vinho* e *#Produtor*. Tal como o número de BI deve ser único para cada pessoa, um código de vinho identifica completamente um dado vinho. Cada linha de cada uma dessas tabelas tem um valor único para esses atributos. Qualquer relação ou tabela numa base de dados tem associadas **restrições de integridade**, ou seja, propriedades a que a informação armazenada deve obedecer. Por exemplo: *as capacidades de uma garrafa de vinho têm de estar compreendidas entre 0,2 e 5,0 litros*, ou *a taxa de execução de um projecto tem de ser inferior a 100%*. O requisito de não repetição dos valores de um dado atributo é igualmente uma restrição de integridade sobre a informação numa tabela de uma base de dados. Trata-se de facto de uma restrição muito importante, que irá estar na origem das noções de chave condidata e de chave primária. Diz-se então que *#Vinho* e *#Produtor* são **chaves candidatas** a serem **chaves primárias** das respectivas tabelas. As chaves candidatas são todos os identificadores únicos de informação de uma tabela. A chave primária da tabela é a chave candidata que é escolhida num determinado caso.

Definição 4.7 ***Chave Candidata***

Seja **R** uma relação definida num conjunto de atributos $X=\{x_1, ..., x_n\}$, sujeita a um conjunto C_{df} de dependências funcionais. Uma chave candidata para **R** é qualquer conjunto de atributos $C_i=\{c_1, ..., c_m\}$, $C_i \acute{I} X$, que satisfaça simultaneamente as duas propriedades seguintes.

1. ***Unicidade*** *(não repetição ou valores únicos)*: Dois n-uplos distintos de um dado **R** não podem ter os mesmos valores para os atributos do conjunto $C_i \acute{I} X$. Para um dado valor de **R** (mas qualquer que ele seja, como modelo da uma certa realidade sujeita a um dado C_{df}) os valores dos m-uplos de C_i nesse **R** são **únicos**.

2. ***Irredutibilidade***: Nenhum *subconjunto próprio*[20] A do conjunto C_i, $A \grave{I} C_i$, pode ter a propriedade anterior. Se existir um tal A então C_i não seria uma chave candidata para **R**. Designava-se então C_i como uma **superchave**.

[20] Um conjunto X designa-se como *subconjunto próprio* de Y, se X está contido em Y mas tem menos elementos do que Y. Por exemplo seja $Y=\{a, b, c\}$. O conjunto $\{a, b\}$ é um subconjunto próprio de Y, mas $X=\{a, b, c\}$ sendo um subconjunto de Y, $X \grave{I} Y$, não é um subconjunto próprio de Y.

Uma dada relação **R** pode ter em geral um conjunto $CC=\{C_1, C_2, ..., C_k\}$ com k chaves candidatas (ver exemplos abstractos na Figura 4.39). Uma chave candidata é assim o menor conjunto de atributos de uma relação cujos valores não se podem conjuntamente repetir.

As duas propriedades apresentadas são importantes, correspondendo de facto a regras de identidade associadas à informação. Uma das suposições básicas é a de que valores iguais não se podem distinguir e correspondem por isso exactamente à mesma entidade. Numa tabela relacional não podem existir duas linhas com todos os valores iguais. Se tal acontecesse estas duas linhas não eram distinguíveis (da mesma forma que um conjunto não pode conter dois elementos iguais). É através da identificação de chaves candidatas, com valores únicos numa dada relação, que se pode pedir ao SGBD para garantir automaticamente essa propriedade, no fundo a manutenção da integridade das entidades. Se a chave indicada não fosse irredutível, o sistema não poderia verificar adequadamente se os valores geridos eram únicos, e detectar por exemplo a tentativa de introduzir o mesmo elemento repetidas vezes. Considere-se como exemplo o caso do Arquivo e dos Bilhetes de Identidade.

Qualquer relação tem pelo menos uma chave candidata, uma vez que não é permitido no modelo relacional a existência de n-uplos com valores duplicados. No caso limite, em que qualquer valor é possível para um elemento (n-upla) da relação, a chave candidata dessa relação é o conjunto de todos os seus atributos.

Como se pode também facilmente concluir, a propriedade da irredutibilidade das chaves candidatas é necessária para assegurar que se identificou o menor subconjunto (ou os menores subconjuntos) de atributos da relação dos quais os restantes dependem funcionalmente.

A identificação das chaves candidatas é muito facilitada através da análise do DDF do problema, aplicando o seguinte procedimento que retira atributos do DDF. Imagine-se que ao retirar um atributo *a* de um DDF se retiravam por "arrastamento" todos os atributos que estavam ligados a esse atributo *a* por uma seta (DF), ou que estavam ligados por setas aos atributos que foram arrastados (o atributo *a* inicial pode ser igualmente um conjunto de atributos de onde parte uma seta). Um conjunto de atributos *C* que ao ser retirado de um DDF arraste consigo todos os restantes atributos presentes no DDF é uma chave candidata. Este conjunto *C* deve ser irredutível, isto é, se se partir de um subconjunto de C e ainda assim forem retirados todos os atributos do DDF, então o primeiro C identificado não era uma chave candidata. Obviamente que pode haver vários conjuntos irredutíveis a partir dos quais sejam eliminados todos os atributos do DDF, o que significa que pode haver várias chaves candidatas. No caso de haver um atributo de onde não parta nenhuma seta e onde não chegue nenhuma seta, então esse atributo deverá pertencer à chave candidata.

Suponha-se uma dada relação **R** definida sobre um conjunto de quaisquer 3 atributos, $X=\{a, b, c\}$, ou seja pode escrever-se **R**{a, b, c}. Por definição, a relação definida nestes 3 atributos não pode ter valores repetidos. Então, se não fosse necessário verificar a segunda propriedade de ***irredutibilidade***, esse conjunto de 3 atributos poderia ser imediatamente uma chave candidata. Uma situação em que a chave candidata seria mesmo o conjunto dos 3 atributos existiria quando, retirando um qualquer dos atributos, existissem sempre valores repetidos para os restantes atributos.

A Figura 4.39 apresenta quatro valores possíveis para uma tal relação **R** definida sobre um conjunto de atributos $\{a, b, c\}$, e as chaves candidatas CC_1, CC_2, CC_3 e CC_4 que seria possível identificar apenas por observação dos valores nas várias linhas de cada uma das tabelas. No

caso de **R1** poderia haver apenas uma chave candidata, enquanto que no caso **R4** poderia haver apenas quatro chaves candidatas.

Em cada situação concreta que esteja a ser analisada, para além de ser importante apresentar alguns exemplos de valores, é igualmente relevante associar a cada uma das relações um conjunto de DF consistente com as chaves candidatas. De facto, é mais fácil e até mais correcto definir as chaves candidatas após a identificação de todas as DF. No entanto identificar as DF e identificar as chaves candidatas são duas formas complementares de introduzir restrições de integridade, que, como já deve ser claro, se condicionam mutuamente. Como se viu, as DF estabelecem regras que permitem caracterizar todas as possibilidades de valores aceitáveis em cada linha de uma tabela.

Exercício: Proponha as DF, e desenhe os respectivos DDF, consistentes com cada uma das quatro relações apresentadas na Figura 4.39, respeitando os valores e as chaves candidatas apresentadas. Tente descrever situações reais que cada uma das relações assim caracterizadas possam representar.

R1

a	*b*	*c*
1	A	L1
1	A	L2
1	B	L1
1	B	L2
2	A	L1
2	A	L2
2	B	L1
2	B	L2

CC_1={{a, b, c}}

R2

a	*b*	*c*
1	A	L1
1	A	L2
1	A	L3
1	A	L4
2	B	L1
2	B	L2
2	B	L3
2	B	L4

CC_2={{a, c}, {b, c}}

R3

a	*b*	*c*
1	A	L1
1	A	L2
1	A	L3
1	A	L4
2	A	L1
2	A	L2
2	A	L3
2	A	L4

CC_3={{a, c}}

R4

a	*b*	*c*
1	A	L1
2	B	L2
3	C	L3
4	D	L5
5	E	L4
6	F	L8
7	G	L7
8	H	L9

CC_4={{a}, {b}, {c}}

Figura 4.39 Chaves candidatas para vários valores de R.

Informalmente, uma chave primária de uma relação ou tabela é um identificador único para a informação na referida relação. Por exemplo na tabela **Vinhos** não devem aparecer na coluna *#Vinho* valores repetidos. Cada linha da tabela tem um valor único de *#Vinho*. Trata-se evidentemente de uma restrição de integridade sobre a informação na base de dados, uma vez que não se aceitam vinhos diferentes com o mesmo código[21]. De facto uma chave primária é um caso particular de uma chave candidata.

[21] A noção de igualdade, se analisada com cuidado, é mais complicada do que parece. Objectos iguais numa situação podem não o ser numa outra situação. Por exemplo, imagine-se o caso das mesas ou cadeiras na Faculdade de Engenharia da Universidade do Porto, do ponto de vista da gestão do inventário: cada mesa e cada cadeira são objectos distintos com um código de inventário distinto. Possivelmente para a empresa que produz as mesas e as cadeiras, estas são apenas *quantidades* em produção. Na prática não têm existência individual na sua base de dados, sendo apenas elementos de um lote de produção de uma determinada peça de catálogo. Apenas cada lote de produção e cada peça de catálogo têm existência como entidade ou objecto.

Definição 4.8 ***Chave Primária***

> Seja **R** uma relação para a qual está identificado o conjunto $CC=\{C_1, C_2, ..., C_k, ...\}$ de chaves candidatas. Uma destas chaves candidatas é designada a **chave primária**, ou C_p, da relação. As restantes chaves candidatas são designadas **chaves alternativas**. A chave primária deverá ser escolhida como a chave candidata mais simples, mais estável ou mais apropriada em cada caso particular. No caso de não ser aparente ou não haver justificação de qual a chave candidata mais apropriada, pode ser feita uma escolha arbitrária. Eventualmente a escolha pode vir a ser alterada mais tarde.

Quando há apenas uma chave candidata a escolha da chave primária é óbvia e imediata. Nos casos em que há várias chaves candidatas a solução passa por escolher a chave *mais simples*, como se indicou. A justificação deste julgamento *chave candidata mais simples* deve ser feito com base em todo o conhecimento disponível sobre o problema concreto. Na Figura 4.38 estão identificadas duas chaves candidatas, mas parece mais *natural* escolher uma destas para chave primária. A chave primária indicada é mais simples: o atributo *data* tem associada mais informação do que o código sequencial *#Actual*. Este último código, tendo significado apenas no contexto da base de dados pode ser gerado automaticamente, estando assim menos sujeito a erros, e sendo assim melhor como chave alheia.

O termo "chave primária", apesar da relevância que tem, nunca deve ser abreviado apenas por "chave". Existem muitos outros tipos de chave, por exemplo: chave candidata, alheia, alternativa, de indexação, mestre, secundária, de pesquisa, de Hash, de encriptação, privada ou pública. A noção de chave primária é com certeza a noção de chave mais importante no projecto de sistemas de bases de dados, e seria a única a merecer ser abreviada como chave. No entanto devido à proliferação das noções de chave, é importante que se qualifique sempre qualquer uma das chaves, para evitar riscos de confusão.

Definição 4.9 ***Chave Alheia***

> Uma **chave alheia** C_a de uma relação **R2** é um atributo, ou conjunto de atributos, cujos valores sejam do mesmo tipo dos valores da chave primária C_p de uma outra relação **R1** distinta (ou da própria relação **R2**).
>
> Diz-se, quando tal acontece, que as relações se podem **cruzar** por essas chaves, e que C_a de **R2** **liga com** C_p de **R1** (ou da própria **R2**).

As chaves primárias são muito importantes no modelo relacional devido ao facto de serem a única forma de garantir um endereçamento que permita seleccionar uma só n-upla de uma relação. Se não se exigisse a existência de chaves primárias (ou candidatas), uma relação poderia ter n-uplas repetidas. Tal situação impedia que num sistema relacional, que esconde todos os detalhes de implementação, nomeadamente eventuais apontadores, fosse possível distinguir ou seleccionar uma única dessas n-uplas [Date 1995, p. 35].

Integridade de entidades e Integridade referencial

Como se viu, são necessárias chaves primárias nas relações como local para identificar e seleccionar informação, e chaves alheias por forma a garantir a possibilidade de cruzamento

de informação relacionada através dos valores dos atributos. Além disso, para que a informação constante na base de dados seja minimamente completa e clara, é importante estabelecer as seguintes regras de integridade ao nível dos conteúdos destes dois tipos de chave.

Definição 4.10 ***Integridade de entidades***

> Nenhum atributo que participe numa chave primária de uma relação pode aceitar valores nulos.

Esta regra significa por exemplo que não é aceitável ter armazenada no Arquivo de Identificação informação sobre uma dada pessoa mas não saber o seu número de BI.

Definição 4.11 ***Integridade referencial***

> Considere-se uma chave alheia C_a na relação **R2** que liga com a chave primária C_p de **R1** (ou que liga com a chave primária da própria **R2**). Então um dos dois casos se tem de verificar:
>
> a) Todo o valor de C_a na relação **R2** tem de existir em C_p em alguma n-upla da relação **R1** (ou da própria **R2**).
>
> b) Os valores de cada um dos atributos que compõe C_a na relação **R2** são nulos.

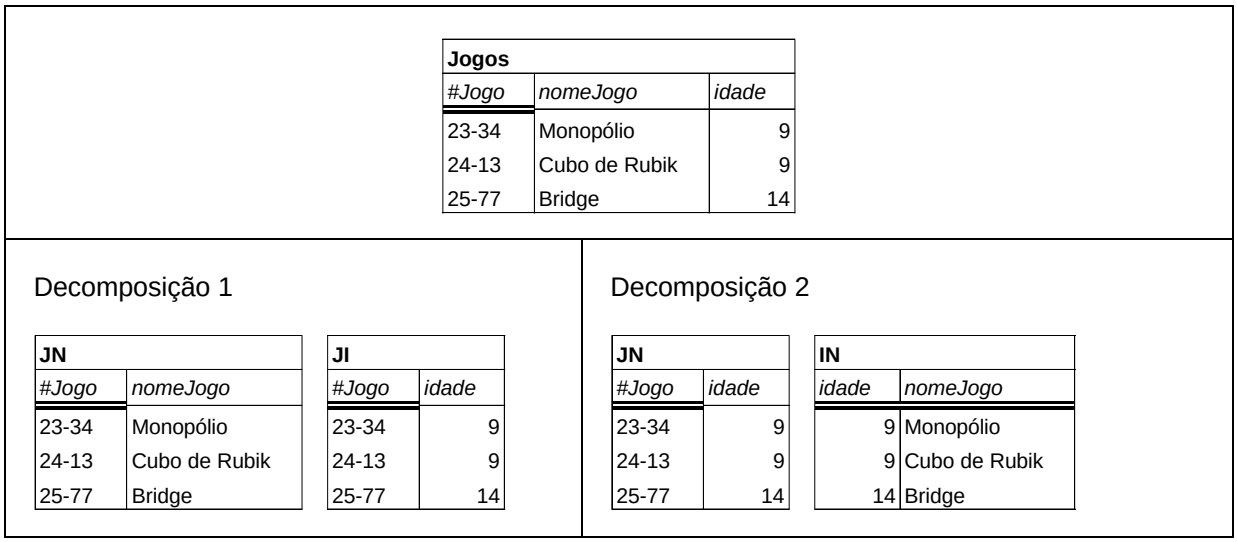
Informalmente esta regra de integridade, básica e essencial do modelo relacional, significa que se numa tabela existe uma referência com valor não nulo a informação adicional que está definida numa outra relação ou tabela (referência mantida através da ligação – igualdade de valores – entre a chave alheia e a chave primária), então nessa outra tabela tem de existir em alguma linha pelo menos uma vez esse valor.

As duas regras apresentadas aplicam-se a qualquer base de dados relacional. Como se verá, há inúmeras regras de integridade que poderiam ser adicionadas a uma base de dados, ao nível dos dados, das tabelas ou da própria base de dados globalmente.

De seguida serão apresentadas propriedades de tabelas e de bases de dados relacionadas com o nível de repetição que a informação armazenada apresenta. Como se referiu anteriormente, o nível de redundância permitido no projecto das tabelas afectará depois a eficiência dos processos de actualização e consulta.

Decomposição sem perda de informação

Numa base de dados respeitando o modelo relacional é por vezes possível representar a mesma informação numa só relação ou em várias relações. A distribuição de informação por várias relações deve ser feita sem perdas, e deve ser guiada por objectivos. O principal objectivo que orienta a decomposição é conseguir que qualquer informação externa à base de dados, e que possa ser alterada, esteja guardada num único local dessa base de dados, garantindo assim a *unilocalização de dados*, facilitando assim a manutenção da sua consistência (cf. secção 4.2.2).



Jogos

#Jogo	*nomeJogo*	*idade*
23-34	Monopólio	9
24-13	Cubo de Rubik	9
25-77	Bridge	14

Decomposição 1

JN

#Jogo	*nomeJogo*
23-34	Monopólio
24-13	Cubo de Rubik
25-77	Bridge

JI

#Jogo	*idade*
23-34	9
24-13	9
25-77	14

Decomposição 2

JN

#Jogo	*idade*
23-34	9
24-13	9
25-77	14

IN

idade	*nomeJogo*
9	Monopólio
9	Cubo de Rubik
14	Bridge

Figura 4.40 Decomposições da tabela **Jogos** *(Figura 4.15): a decomposição 1 em JN e JI preserva a informação, mas a decomposição 2 em JN e IN corresponde a perda de informação (porquê?).*

O processo de decomposição assume que seria possível colocar inicialmente toda a informação a gerir numa só tabela, em primeira forma normal. A informação nessa tabela sofre de vários problemas em relação ao espaço ocupado e ao processo de actualização de infromação. Torna-se então necessário decompor essa tabela sem perder informação, procurando a organização mais adequada. O processo de decomposição sem perdas de informação é designado por **normalização funcional** e procura, sempre que for possível, colocar informação separada em tabelas distintas (distintas, mas relacionadas através das ligações entre valores das chaves alheias e primárias). O objectivo da decomposição ou normalização funcional é ainda procurar que a actualização de uma certa informação atómica seja feita através da alteração de um únic registo.

Redundância

A existência de dados redundantes na base de dados tem como consequência o aumento das necessidades de memória, originando ainda vários problemas ao nível do processo de actualização de informação.

Os atributos que guardam valores numa base de dados podem ser de três tipos:

- Atributos apenas de identificação – atributos identificativos.
- Atributos apenas de informação – atributos informativos.
- Atributos que servem para identificação e informação – atributos informativos e identificativos.

Os atributos código são normalmente apenas de identificação, sobretudo quando os seus valores têm origem na própria base de dados. Estes atributos fazem parte normalmente das chaves das relações, e são usados para estabelecer relações entre objectos. Atributo apenas de informação são por exemplo nomes, endereços ou telefones. Os valores que estes atributos podem tomar são modelados a partir dos contextos reais, e não são nunca gerados pela própria base de dados. Os atributos mistos são por exemplo códigos com origem em sistemas de

classificação tradicionais mas que o sistema em causa tem de manter como informação (sujeita por exemplo a erro de introdução). Em geral esta classificação dos atributos é subjectiva, dependendo do sistema e do autor. Por exemplo, uma biblioteca tem de usar códigos ISBN. Estes códigos são simultaneamente informativos e identificativos. Para a entidade que cria e atribui os códigos ISBN, não havendo grande possibilidade de erro ou necessidade de proceder a alterações, o atributo é sobretudo identificativo.

A existência de valores repetidos dos atributos identificadores em vários locais ou tabelas de uma base de dados não é considerada informação redundante. A existência de valores repetidos dos atributos informativos é considerada informação redundante. No caso dos atributos mistos aplica-se a mesma regra: se a função é informativa, há redundância, se é identificativa, não há redundância.

A normalização funcional visa em eliminar redundância nos atributos informativos, e sempre que possível também nos atributos informativos e identificativos.

4.5.6 1ª, 2ª e 3ª Formas Normais

Nesta secção iremos considerar apenas relações com uma única chave candidata, que é precisamente a chave primária. No caso geral, em que existem várias chaves candidatas, as definições seguintes devem ser alteradas (ver por exemplo [Date 1997]). Na quase totalidade dos casos práticos a introdução de atributos identificadores adequados torna óbvia qual a chave primária a utilizar, sendo assim necessário considerar apenas essa chave no processo de normalização.

Informalmente, e indo directamente à terceira forma normal, uma relação **R** está em 3FN se e só se os **atributos não-chave** (os atributos que não participam na chave primária da relação):

a) São **mutuamente independentes** (dois ou mais atributos são mutuamente independentes se nenhum deles é funcionalmente dependente de qualquer combinação dos restantes).

b) São **dependentes irredutivelmente** da chave primária (isto é, não dependem funcionalmente de um subconjunto próprio de atributos da chave primária).

Começando pelo princípio, e de acordo com o que já foi indicado.

Definição 4.12 ***Primeira Forma Normal – 1FN***

> Uma relação **R** está em 1FN se e só se todos os domínios de definição dos atributos têm apenas valores atómicos ou escalares.

Definição 4.13 ***Segunda Forma Normal – 2FN***

> Uma relação **R** está em 2FN se e só se está em 1FN e todos os **atributos não-chave** (os atributos que não participam na chave primária da relação) são **dependentes irredutivelmente** da chave primária (isto é, não dependem funcionalmente de um subconjunto próprio de atributos da chave primária).

Definição 4.14 ***Terceira Forma Normal – 3FN***

> Uma relação **R** está em 3FN se e só se está em 2FN e todos os **atributos não-chave** são **dependentes não transitivamente** da chave primária (isto é, não dependem funcionalmente de outros atributos que não os da chave primária).

4.5.7 Forma Normal de Boyce-Codd – FNBC

No caso em que uma relação possa ter mais do que uma única chave candidata a definição de terceira forma normal em particular requer uma actualização, dando origem à designada Forma Normal de Boyce-Codd – FNBC. Esta forma normal, com o mesmo objectivo de obter decomposição sem perda de informação e boas propriedades de actualização e consulta pode ser estudada em detalhe em [Date 1997]). A definição de terceira forma normal não tratava adequadamente o caso em que uma relação tinha:

a) Duas ou mais chaves candidatas.

b) Pelo menos duas dessas chaves candidatas eram compostas por mais do que um atributo.

c) Havia pelo menos um atributo comum a essas duas chaves candidatas compostas.

Definição 4.15 ***Forma Normal de Boyce-Codd – FNBC***

> Uma relação **R** está em FNBC se e só se toda e qualquer FD não trivial e irredutível à esquerda, tem uma chave candidata como determinante (ver Definição 4.6).
>
> Por outras palavras, as únicas setas no DDF são setas a partir de chaves candidatas. Não havendo mais setas, não é possível *normalizar mais* a relação.

Repare-se que o processo de normalização é orientado pela semântica da informação a gerir, e não pelos valores que se encontram numa dada relação ou conjunto de relações num dado instante. A semântica de um problema depende da utilização dos dados no ambiente real e pode ser capturada em parte pelas dependências funcionais associadas aos atributos do problema.

4.5.8 Outras formas normais

O processo de normalização não fica ainda completo, embora no que diz respeito às dependências funcionais simples, tal como foram definidas fiquem resolvidos todos os problemas desses tipos (ou seja, todas as restrições de integridade que se possam exprimir por DF são tratados pelo processo de normalização até 3FN e FNBC).

Existem no entanto alguns tipos de restrições de integridade que requerem a definição de **dependências funcionais multivalor** e que dão origem a níveis de normalização mais especializados, as designadas quarta e quinta formas normais: 4FN e 5FN. Para este tema sugere-se por exemplo o Capítulo 11 de [Date 1997].

4.6 Processo de Concepção de uma BDr baseado em Normalização Funcional de Dados

Uma vez identificada e justificada a necessidade de construir um sistema que inclua uma base de dados relacional, há métodos bem conhecidos que podem e devem ser seguidos para o seu projecto. Estes métodos não podem ignorar toda a análise de sistemas sempre necessária, considerando em particular as necessidades dos utilizadores, formalizadas por exemplo com casos essenciais de uso, cenários, ou casos concretos de uso (ver Capítulo 3). No entanto, assumindo que se está envolvido num processo iterativo e evolutivo, e que a concepção de uma base de dados relacional visa definir um conjunto de tabelas relacionadas que possam depois ser consideradas para base de um projecto com um SGBD comercial, há que seguir as técnicas existentes para esta fase do projecto.

A Figura 4.41 resume a abordagem proposta. A **fase de análise e modelação** prévia do sistema produz um enunciado do problema, incluindo a análise ou síntese de requisitos, por exemplo com uma proposta de cenários ou casos de uso, de um modelo de classes e de processos de negócio. A **fase de concepção** inclui sete passos. Esta fase inicia-se com a adaptação ou interpretação do enunciado pela equipa de projecto, e termina com a produção de um esquema relacional conceptual validado e pronto para dar início à fase de programação da base de dados. Os requisitos não funcionais relevantes, nomeadamente os que influenciam o desempenho das consultas e actualizações de informação já devem ter sido considerados nesta fase do projecto (ver secção 4.5.3).

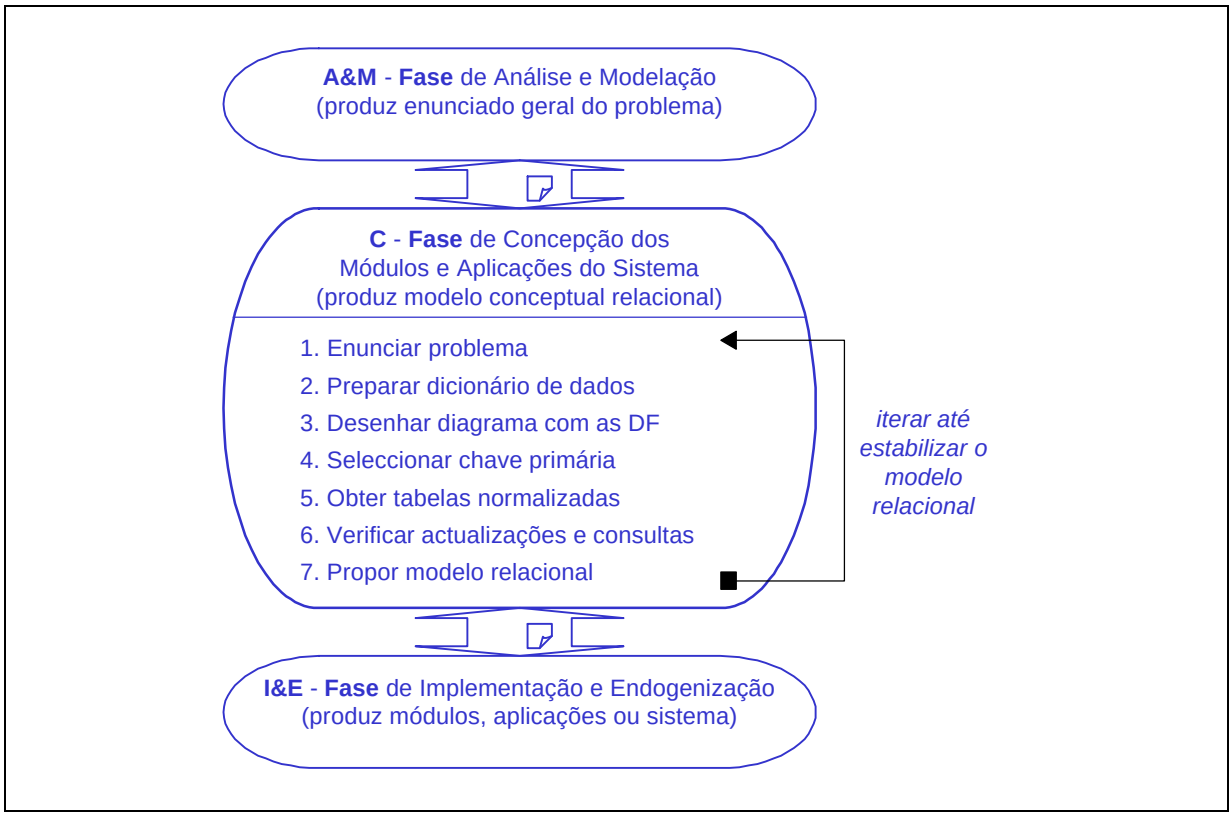


Figura 4.41 A fase de concepção dos módulos e aplicações de um sistema tal como deve ser aplicada à concepção de uma base de dados relacional – detalhe dos 7 passos recomendados

A **fase de implementação e endogenização**, posterior à concepção da base de dados, inclui a implementação das tabelas, selecção de tipos de dados concretos a utilizar, implementação dos procedimentos para construir a interface com o utilizador ou com outros sistemas (funções de actualização e consulta). Nesta fase serão considerados todos os requisitos não funcionais, tais como a memória ou capacidade de processamento disponível, os tempos de resposta requeridos e as funções disponíveis no SGBD comercial seleccionado. É nesta fase que são criadas as chaves ou índices de ordenação, simples ou abrangendo conjuntos de atributos, que podem tornar mais eficientes os processos de consulta e selecção de informação. Os vários requisitos não funcionais exigem normalmente que nesta fase se proceda a uma re-estruturação das tabelas definidas no processo de concepção, alterando o nível de normalização, a distribuição de informação particular por tabelas múltiplas ou pelo contrário a inclusão numa só tabela de informação de tipos diferentes. Deve haver no entanto o cuidado de manter documentação simples mas eficaz sobre as opções de transformação sobre o modelo conceptual de classes ou de relações, bem como manter as vistas externas estáveis (para interface com outros sistemas). Normalmente os modelos conceptuais de classes ou de relações são o único ponto de comunicação comum entre os representantes dos utilizadores e os representantes dos programadores, e é essencial que sejam claras as implicações sobre este modelo da evolução da base de dados.

A **processo de concepção da base de dados relacional** pode envolver as duas fases fronteiras indicadas anteriormente, sobretudo para a construção de um primeiro protótipo com funcionalidades para efeito de validação junto dos utilizadores. Neste caso toda a análise é feita informalmente mas com cuidado, e todos os requisitos não funcionais são anotados mas em geral ignorados na construção do primeiro protótipo (excepto se forem requisitos não funcionais muito importantes).

Supondo que se dispõe de um entendimento suficientemente completo do problema e do contexto de aplicação, a fase de concepção da base de dados relacional deveria contemplar os seguintes sete passos:

1. ***Enunciar problema***: descrever adequadamente o contexto e o problema em causa, as principais solicitações dos seus possíveis utilizadores (pessoas ou outros sistemas) e o objectivo do sistema proposto como solução, em termos do negócio da organização ou da unidade funcional em que se insere. Tipicamente este enunciado corresponde aos **requisitos dos utilizadores**.
2. ***Preparar dicionário de dados e tabela base***: identificar todos os atributos simples de informação, criando uma única tabela com todos os identificadores de atributos, uma descrição sucinta, os tipos de dados propostos e exemplos de valores. Estes atributos podem vir a ser os nomes das colunas das tabelas em primeira forma normal. Sendo assim, devem ser eliminados atributos cujos valores sejam listas, listas estas cujos elementos tenham significado individual (os sistemas relacionais existentes não suportam operações eficientes sobre este tipo de valores, em particular pesquisa com indexação). A produção de uma tabela inicial única não é claramente confortável para problemas com mais de 15 ou 20 atributos. Para problemas de maior dimensão deve partir-se logo de um conjunto de tabelas mais ou menso normalizadas, que podem ser identificadas através do processo de análise baseado na modelação de classes (Capítulo 3). Um projectista de base de dados experiente consegue normalmente identificar imediatamente um

conjunto de tabelas necessárias, iniciando a normalização num estado já avançado de decomposição do dicionário de dados (que mantém a sua importância).

3. ***Desenhar diagrama de dependências funcionais***: caracterizar as dependências funcionais, representando-as graficamente através de um DDF. Num processo de análise, as DF capturam algumas das regras de negócio; num processo de síntese as DF permitem impor regras que podem ser respeitadas automaticamente pelo SGBD. Grande parte dos valores e significados associados à informação na base de dados é assim condicionado objectivamente.

4. ***Seleccionar chave primária***: identificar conjuntos de atributos candidatos a chave primária, isto é identificar as chaves candidatas. Escolher de entre estas a chave primária, se possível de uma forma justificada. Em particular as chaves primárias só devem conter atributos identificadores, raramente devem conter atributos informativos. Os atributos informativos serão utilizados para manter relações entre informação, podendo os seus valores ser repetidos em chaves alheias. Sendo assim, qualquer actualização do valor de um atributo informativo pode exigir alterações em vários locais da base de dados, por forma a garantir consistência e adequação à realidade.

5. ***Obter tabelas normalizadas e identificar chaves alheias***: distribuir ou decompor a informação de todos os atributos, identificada anteriormente no dicionário de dados ou na tabela base, por um conjunto de tabelas normalizadas sem perdas de informação. Normalmente este passo conduz a um conjunto de tabelas inter-relacionadas em terceira forma normal (ou tabelas em FNBC). Neste passo é necessário identificar todas as chaves alheias existentes em todas as tabelas, pois são estas chaves que estabelecem as inter-relações entre tabelas, e permitem "cruzar" informação, permitindo executar as consultas.

6. ***Verificar possibilidade de actualização e consulta***: verificar se a estrutura da base de dados permite satisfazer as questões a formular pelos utilizadores, em termos de requisitos funcionais. Deve normalmente existir uma expressão em SQL que permita responder a cada uma e qualquer solicitação dos utilizadores ou sistemas externos. Neste passo descobrem-se muitas vezes falhas nas soluções encontradas nos passos anteriores. Convém lembrar que alguns requisitos funcionais podem não ter resposta apenas com consultas SQL, exigindo a utilização de uma linguagem procedimental com ciclos ou recursão. Alterações à estrutura da base de dados, ou a inclusão de atributos adicionais, por exemplo de códigos novos, podem facilitar significativamente as consultas, ao nível da estrutura de concepção da consulta. A eficiência futura das consultas, a testar na fase da implementação da base de dados, também pode ser obviamente afectada. No final desta fase devem ser considerados os requisitos não funcionais, e a forma como se prevê que o esquema relacional que foi obtido responda em condições de utilização real, nomeadamente quando o volume de dados for grande ou muito grande.

7. ***Propor modelo relacional***: propor um conjunto de tabelas inter-relacionadas, com restrições de integridade e lista de operações para interface com utilizadores e sistemas externos. Este modelo será utilizado na fase de implementação e endogenização.

Estes passos são obviamente iterativos e evolutivos, podendo afectar a formulação do problema. Os utilizadores podem ser confrontados com uma situação em que têm de alterar a forma de colocar as questões para se poder construir o sistema dentro dos orçamentos e prazos admissíveis.

De seguida exemplifica-se a derivação de um modelo relacional para uma empresa de vinhos, com base nos sete passos apresentados.

4.6.1 Enunciar problema

Nesta primeira fase deve ser claro para a equipa de projecto qual o problema a resolver, sendo para tal necessário obter um enunciado simples e objectivo desse problema. Uma descrição pode corresponder a uma situação existente, ou uma situação nova para a qual se deseja construir um sistema de gestão de informação. No primeiro caso está-se perante uma actividade de **análise de requisitos**, no segundo caso perante uma actividade de **síntese de requisitos**. Em geral o termo análise de requisitos é utilizado em ambos os casos. Naturalmente a síntese de requisitos envolve um conjunto de riscos associados à inovação, e requer muito mais experimentação e validação junto de potenciais utilizadores.

O enunciado do problema pode incluir um texto resumido, exemplos de formulários ou listagens em utilização ou propostos, relatórios ou estudos existentes, bem como decisões ou regras de negócio estabelecidas pelos vários departamentos da organização. Este enunciado deveria ser organizado em cenários ou casos de uso (ver por exemplo descrição do caso VML, em Anexo, e Capítulo 3 sobre Modelação Conceptual de Classes).

Exemplo 4.8: O Sistema de Informação integrado para a empresa WiNet:

> "O grupo WiNet resultou de um processo de aquisições e fusões de um conjunto de empresas de armazenamento, revenda e retalho de vinhos, liderada pelos sócios da adega WiNet. O processo de reengenharia por que as várias empresas passaram, originou um conjunto de requisitos para o sistema de informação de apoio às operações e gestão da WiNet.
>
> A estratégia de posicionamento do grupo no mercado, as suas necessidades de informação e a oferta informática existente, contribuíram para se adoptar uma solução baseada numa aplicação parametrizável existente no mercado, assente em tecnologia SGBDr, a que seria necessário acrescentar um conjunto de módulos e funcionalidades específicas ao sector da comercialização de vinho, de acordo com os factores críticos de sucesso identificados pela WiNet.
>
> Assim, tornou-se por exemplo necessário integrar toda a informação relativa a clientes e fornecedores do grupo, aos funcionários das várias empresas e dos vários departamentos, mantendo permanentemente actualizada a estrutura da organização, os orçamentos, custos e receitas das várias unidades. Foi também necessário manter actualizadas as existências de vinhos e produtos similares, bem como a sua localização nos vários armazéns e lojas. Introduziu-se também um módulo para permitir aproximar os funcionários e colaboradores da WiNet com os clientes existentes e os clientes potenciais, melhorando o conhecimento disponível no sistema, por exemplo integrando dados sobre as especialidades dos funcionários, os interesses dos clientes e os produtos locais".

4.6.2 Preparar dicionário de dados

Uma vez compreendido o problema, torna-se necessário extrair todos os atributos de informação que se julgam relevantes. Para tal deve ser criado um dicionário de dados.

Definição 4.16 ***Dicionário de Dados (DD)***

Um dicionário de dados é uma tabela22 que inclui pelo menos as seguintes quatro informações: nome ou identificador de cada atributo, descrição sucinta do seu significado, tipo de dados proposto para os valores do atributo, e exemplos de valores possíveis.

O dicionário de dados assim estruturado corresponde também a uma simplificação da ficha de objecto ou classe, técnica de identificação de requisitos muito utilizada em análise orientada por objectos[23].

[22] A palavra "tabela" é aqui usada no sentido normal, não no sentido de tabela de um sistema relacional. No entanto quer um SGBD quer cada uma das bases de dados que um SGBD gere em cada momento têm associados conceitos idênticos mas formais de dicionário de dados ou de catálogo de dados baseados no conceito informal aqui introduzido. Por exemplo, existem tabelas do SGBD acessíveis apenas aos utilizadores especiais que guardam para cada base de dados activa os designados **metadados**: por exemplo, nomes dos atributos e das respectivas tabelas em utilização, número de linhas em cada tabela, ou o número de tabelas em cada base de dados (ver por exemplo [Date 2000], p. 59).

[23] Ver [Harmon & Watson 1998], [Gomes 1998] ou [Dayton *et al* 1998] ("Task Object Cards", p. 42). Neste último caso, uma ficha está identificada pelo nome da classe e apresenta a sua descrição sucinta, uma lista de atributos, uma lista de acções que o utilizador lhe pode aplicar, e um conjunto de relações de pertença para outros objectos ("estou contido em/pertenço a", "contenho"). Para a ficha de um quarto de um hotel seria por exemplo: **Nome**: Quarto; **Descrição**: Local de um hotel que um cliente pode marcar ou ocupar; **Atributos**: #Quarto, paraFumador, #Cliente, planta; **Acções**: ver, editar, quardar, imprimir; **Relações**: *pertenço a* Hotel; *contenho* Móveis.

Dicionário de Dados para o Sistema Winet

Atributo	Tipo de dados	Descrição e observações	Exemplos
#Produtor	Número entre 0 e 10000 (inclusivé).	Código do produtor; atributo identificador único de um produtor com capacidade para comercilaizar uma ou mais marcas de vinho; não inclui apenas armazenistas ou distribuidores.	45567, 22322, 33
nomeCurtoProdutor	20 caracteres alfanuméricos.	Nome pelo qual o produtor é normalmente conhecido pelo público.	António Esteves Ferreira, Ferreira
nomeCompletoProdutor	80 caracteres alfanuméricos.	Nome formal do produtor, normalmente de uma empresa ou organização com estatuto fiscal. Pode ter um valor igual ao nomeCurtoProdutor	António Esteves Ferreira, A. A. Ferreira, S.A.
tipoProdutor	0..200	Reflecte um julgamento sobre estatuto e tipo de contrato comercial para aquisição de vinhos	0, 43, 100, 200
telefoneProdutor	30 caracteres numéricos, +, -, *espaço*	Ver exemplo	351-251 41 6769
faxProdutor	*telefoneProdutor*	Ver exemplo	351-251 41 6770
sedeProdutor	20 caracteres alfanuméricos	Local (Cidade, Vila ou lugar) onde o produtor indica ter a sua sede	Vila Real, Amarante, Porto, Setúbal
#Vinho	Número entre 0 e 1000000	Código do vinho; atributo identificador único de um tipo ou marca de vinho	221, 33, 999999
nomeVinho	30 caracteres alfanuméricos	Ver exemplos	Tapada do Chaves, Muralhas de Monção
qtdd	Número inteiro	Quantidade (medida em garrafas) de um vinho fornecida a cada loja.	0, 3000
cl.	Número inteiro	Capacidade medida em cl. da garrafa de vinho.	50, 75, 100, 150
região	Lista de Valores	Região vitivinícola; não inclui sub-regiões.	Douro, Vinho do Porto, Alentejo
cor	15 caracteres alfanuméricos	Cor do Vinho	Rosé, Tinto, Aloirado
#Loja	Número entre 0 e 100	Código de loja de venda de vinhos; atributo identificador único de cada loja da rede	91, 2, 99
NomeLoja	30 caracteres alfanuméricos	Nome único para cada loja da rede	Bebo Sempre em Casa
localLoja	30 caracteres alfanuméricos	Nome da localidade onde se situa a loja. No caso de uma cidade de média ou grande dimensão pode ser indicada igualmente a freguesia.	Porto - Paranhos, Luanda, Dili, Lisboa
IpLoja	Número entre 0 e 254	Número da rede interna (tipo “Internet Protocol”). Cada loja tem um identificador único para o ip.	23, 254

Nesta fase de definição dos atributos surgim normalmente muitas dúvidas e questões que originam alterações ao enunciado do problema. No exemplo anterior podem por exemplo ser

colocados os seguintes problemas, que levam à necessidade de definir com grande cuidado as palvaras ou conceitos utilizados.

- Um produtor pode também ser armazenista ou distribuidor?
- Um produtor está associado a uma única região vitivinícola?
- Qual a definição de produtor? Um produtor é alguém, pessoa individual ou colectiva que produz uvas, ou é alguém que tem autorização para comercializar uma determinada marca de vinho?

Só após uma definição cuidada dos atributos utilizados se deve dar início ao processo de normalização, embora como este processo seja evolutivo se possa e deva sempre voltar à fase inicial, se tal for aconselhável. No entanto um pequeno esforço inicial de definição cuidadosa evita um número exagerado de ciclos evolutivos.

A partir do dicionário de dados pode ser construída uma tabela relacional com tantas colunas quantos os atributos. Para testar a compreensão do problema devem ser introduzidas nesta tabela linhas com exemplo de informação real. A Figura 4.42 apresenta um exemplo dos dados que se podem colocar na referida tabela (note-se que, por uma questão de espaço, a primeira coluna da figura apresenta os atributos; ao contrário da norma utilizada neste capítulo, as linhas de dados são aqui apresentadas em colunas). A introdução de dados de teste deve ser feita com cuidado, uma vez que a informação de cada linha diz respeito aos mesmos objectos ou entidades, devendo por isso obedecer às regras em vigor, ou às regras que se quer vri a respeitar. Durante este exercício devem ir sendo anotadas as regras ou suposições que vão sendo feitas sobre a infromação. Algumas destas regras podem depois ser utilizadas na definição das chaves, pode ser possível a sua introdução na definição em SQL da estrutura da base de dados (como restrições de integridade), ou pode ser necessário recorrer a programação para as validar, verificar ou impor.

WiNet {Tabela transposta em primeira forma normal - sem indicação de chave}						
#Produtor	3	3	7	11	18	21
nomeCurtoProdutor	Ferreira	Ferreira	António Esteves Ferreira	Niepoort Vinhos	Casa de Villar	Barbeito
nomeCompletoProdutor	A. A. Ferreira, S.A.	A. A. Ferreira, S.A.	António Esteves Ferreira	Niepoort Vinhos, S.A.	Casa de Villar, Lda.	Vinhos Barbeito, S.A.
tipoProdutor	95	95	50	54	75	80
telefoneProdutor	351-22 374 5292	351-22 374 5292	351-251 41 6769	351-22 338 9528	351-255 91 1380	351-291 76 1829
faxProdutor	351-22 374 5293	351-22 374 5293	351-251 41 6770	351-22 338 9529	351-255 91 1381	351-291 76 1830
sedeProdutor	Vila Nova de Gaia	Vila Nova de Gaia	Melgaço	Porto	Lousada	Funchal
#Vinho	1	1	99	34	116	789
nomeVinho	Barca Velha	Barca Velha	Soalheiro	Redoma	Vinho da Senhora	Malvazia
qtdd	2000	2000	1500	75	3500	5
cl.	75	75	75	75	75	75
região	Douro	Douro	Vinho Verde	Douro	Vinho Verde	Vinho da Madeira
cor	Tinto	Tinto	Branco	Rosé	Branco	Aloirado
#Loja	48	49	49	49	49	49
NomeLoja	Só Uvas e Bagos	Uvas Atenas Portas	Uvas Atenas Portas	Uvas Atenas Portas	Uvas Atenas Portas	Uvas Atenas Portas
localLoja	Faro	Idanha-a-Velha	Idanha-a-Velha	Idanha-a-Velha	Idanha-a-Velha	Idanha-a-Velha
lpLoja	32	77	77	77	77	77

Nota: por questões de espaço, a tabela é apresentada de forma transposta relativamente à norma.

Figura 4.42 Tabela WiNet em primeira forma normal – 1FN - contendo informação de todos os tipos referenciados no dicionário de dados anterior: cada linha vertical corresponde a valores para os vários atributos.

4.6.3 Identificar Dependências Funcionais e representar o respectivo DDF

Considerando os significados anteriores, e ainda que apenas os códigos de produtor, vinho e loja são atributos identificadores, identificam-se as dependências funcionais seguintes, graficamente representandas no Diagrama de Dependências Funcionais (DDF) da Figura 4.43.

Supõe-se que cada vinho tem apenas um produtor (*#Vinho → #Produtor*), e que cada loja tem uma dada quantidade de garrafas de cada vinho (*#Loja, #Vinho → qtdd*). Também se indica que um produtor pode em geral ter mais do que um vinho (visto não se verificar a DF *#Produtor → #Vinho*). As restantes dependências funcionais indicam que cada produtor, vinho e loja estão caracterizados.

Nada é referido, por exemplo, sobre a possibilidade de haver produtores que partilhem o mesmo número de telefone ou local de sede (o que significa que tal pode acontecer). O facto de não se verificarem as DF *#Vinho → #Loja* e *#Loja → #Vinho* significa que não existe nenhuma correspondência funcional entre estes identificadores.

Em resumo as DF que se deseja impor no problema são de momento[24] as seguintes:

#Produtor → nomeCurtoProdutor, nomeCompletoProdutor, tipoProdutor, telefoneProdutor, faxProdutor, sedeProdutor.

#Vinho → #Produtor.

#Vinho → nomeVinho, cl, região, cor.

#Loja → nomeLoja, localLoja, ipLoja.

#Loja, #Vinho → qtdd.

[24] Uma vez que este processo é evolutivo pode acontecer que, por exemplo, após se testar a solução encontrada seja necessário alterar este DDF.

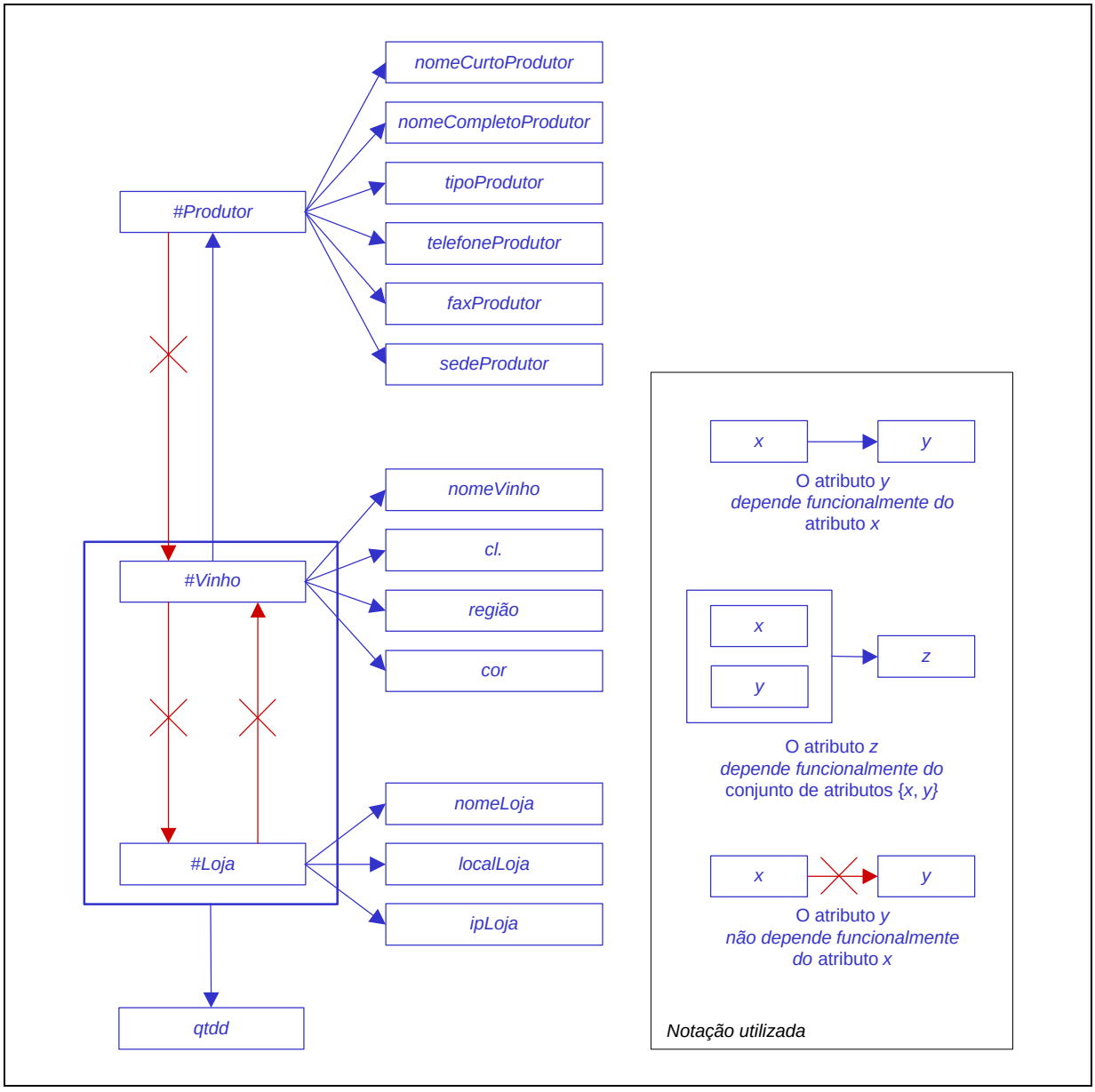


Figura 4.43 Diagrama de dependências funcionais associado à informação identificada no dicionário de dados anterior

4.6.4 Seleccionar a Chave Primária

A chave primária de uma relação, de acordo com a Definição 4.8, é seleccionada a partir do conjunto das chaves candidatas. As chaves candidatas são conjuntos de atributos que cumprem a Definição 4.7. É normalmente relativamente fácil identificar as chaves candidatas a partir de um DDF bem desenhado: uma chave candidata é um conjunto de atributos *irredutível* (ver exemplo seguinte) dos quais todos os outros dependam funcionalmente, directa ou indirectamente. O seguinte algoritmo "operacional" permite identificar as chaves candidatas.

Imagine-se que posso retirar (apagando) do DDF um atributo *a*; imagine-se que ao retirar um atributo retiro também os atributos que dependem funcionalmente desse atributo, e assim sucessivamente. Por exemplo se no DDF existir apenas a DF $a \rightarrow b$, quando retiro o atributo *a*

retiro "por arrastamento" igualmente o atributo *b*. Note-se que se se começar por retirar o atributo *b*, o atributo *a* mantém-se no DDF (pois não tenho a DF $b \rightarrow a$). No caso de DF do tipo $a, b \rightarrow c$, o atributo *c* só é retirado se anteriormente forem retirados os atributos *a* e *b*. Um conjunto de atributos é uma chave candidata se for um conjunto irredutível. Isto é, ao serem retirados todos os seus elementos como atributos do DDF, são retirados todos os restantes atributos do DDF "por arrastamento".

A chave primária deverá ser escolhida como a chave candidata mais simples, mais estável ou mais apropriada em cada caso particular.

No caso do DDF representado na Figura 4.43 a única **chave candidata será o conjunto {*#Loja, #Vinho*}**. Este conjunto "retira" todos os atributos do DDF. O conjunto {*#Loja, #Vinho, #Produtor*} também "retira" todos os atributos, mas não é irredutível, uma vez que se se eliminar o atributo *#Produtor* deste conjunto, o conjunto resultante {*#Loja, #Vinho*} continua a ter a mesma propriedade de retirar todos os atributos (é isto que significa *irredutível*). No entanto {*#Loja, #Vinho*} é irredutível, pois nem {*#Loja*} nem {*#Vinho*} retiram todos os atributos do DDF. **Neste caso como a chave candidata é única será também a chava primária**.

De seguida exemplifica-se a aplicação deste algoritmo a dois casos anteriores, apresentados na secção 4.5.5 sobre gestão de informação no arquivo de identificação.

Na Figura 4.37 - DDF do caso histórico do Arquivo de Identificação, as chaves candidatas são no caso a) apenas o conjunto {*nºBI, data*}. Mas no caso b) já há duas chaves candidatas: {*nºBI, data*} e {*nºCont, data*}. Sendo o Aruivo de Identificação responsável pela atribuição apenas do Bilhete de Identidade, teria todo o sentido escolher como chave primária em qualquer dos casos o conjnuto {*nºBI, data*}.

Na Figura 4.38 - Duas propostas de DDF do Arquivo de Identificação para o caso histórico com código de actualização e data, as chaves candidatas poderiam ser os seguintes conjuntos: {*nºBI, #Actual*} e {*nºBI, data*} (porquê?).

4.6.5 Obter tabelas normalizadas

Primeira Forma Normal – 1FN

A tabela da Figura 4.42 está em primeira forma normal. Como já se identificou o seu DDF e a chave primário pode agora com facilidade ser colocada a informação em segunda forma normal.

Segunda Forma Normal – 2FN

Em segunda forma normal, de acordo com a Definição 4.13, todos os atributos dependem funcionalmente de toda a chave primária.

Neste caso a tabela da Figura 4.42 daria origem a três tabelas, por forma a que, em cada uma destas tabelas todos os atributos dependam funcionalmente de toda a chave primária:

V&P{*#Vinho, nomeVinho, cl, região, cor, #Produtor, nomeCurtoProdutor, nomeCompletoProdutor, tipoProdutor, telefoneProdutor, faxProdutor, sedeProdutor***}**

Lojas{*#Loja, nomeLoja, localLoja, ipLoja* **}**

Quantidades{*#Loja, #Vinho, qtdd* **}**

V&P (Vinhos e Produtores)

#Vinho	*nomeVinho*	*cl*	*região*	*cor*	*#Produtor*	*nomeCurtoProdutor*	*nomeCompletoProdutor*	*tipoProdutor*	*telefoneProdutor*	*faxProdutor*	*sedeProdutor*
1	Barca Velha	75	Douro	Tinto	3	Ferreira	A. A. Ferreira, S.A.	95	351-22 374 5292	351-22 374 5293	Vila Nova de Gaia
99	Soalheiro	75	Vinho Verde	Branco	7	António Esteves Ferreira	António Esteves Ferreira	50	351-251 41 6769	351-251 41 6770	Melgaço
34	Redoma	75	Douro	Rosé	11	Niepoort Vinhos	Niepoort Vinhos, S.A.	54	351-22 338 9528	351-22 338 9529	Porto
116	Vinho da Senhora	75	Vinho Verde	Branco	18	Casa de Villar	Casa de Villar, Lda.	75	351-255 91 1380	351-255 91 1381	Lousada
789	Malvazia	75	Vinho da Madeira	Aloirado	21	Barbeito	Vinhos Barbeito, S.A.	80	351-291 76 1829	351-291 76 1830	Funchal

Lojas

#Loja	*NomeLoja*	*localLoja*	*IpLoja*
48	Só Uvas e Bagos	Faro	32
49	Uvas Atenas Portas	Idanha-a-Velha	77

Quantidades

#Vinho	*#Loja*	*qtdd*
1	48	2000
1	49	2000
99	49	1500
34	49	75
116	49	3500
789	49	5

Figura 4.44 Tabelas para a base de dados WiNet em segunda forma normal – 2FN

Terceira Forma Normal – 3FN

Em terceira forma normal, de acordo com a Definição 4.14, todos os atributos dependem funcionalmente de toda a chave primária, e dependem funcionalmente directamente da chave primária. No caso anterior, em segunda forma normal, os atributos que dependiam funcionalmente do código do produtor dependiam funcionalmente de toda a chave primária da tabela (código do vinho), mas indirectamente.

Neste caso a tabela **V&P** da Figura 4.44 daria origem a duas tabelas, por forma a que, em cada uma destas duas tabelas todos os atributos dependam funcionalmente e directamente de toda a chave primária. Assim sendo, em terceira forma normal haveria necessidade de quatro tabelas:

Vinhos{*#Vinho, nomeVinho, cl, região, cor, #Produtor***}**

Produtores{*#Produtor, nomeCurtoProdutor, nomeCompletoProdutor, tipoProdutor, telefoneProdutor, faxProdutor, sedeProdutor***}**

Lojas{*#Loja, nomeLoja, localLoja, ipLoja* **}**

Quantidades{*#Loja, #Vinho, qtdd* **}**

Vinhos

#Vinho	*nomeVinho*	*cl*	*região*	*cor*	*#Produtor*
1	Barca Velha	75	Douro	Tinto	3
99	Soalheiro	75	Vinho Verde	Branco	7
34	Redoma	75	Douro	Rosé	11
116	Vinho da Senhora	75	Vinho Verde	Branco	18
789	Malvazia	75	Vinho da Madeira	Aloirado	21

Produtores

#Produtor	*nomeCurtoProdutor*	*nomeCompletoProdutor*	*tipoProdutor*	*telefoneProdutor*	*faxProdutor*	*sedeProdutor*
3	Ferreira	A. A. Ferreira, S.A.	95	351-22 374 5292	351-22 374 5293	Vila Nova de Gaia
7	António Esteves Ferreira	António Esteves Ferreira	50	351-251 41 6769	351-251 41 6770	Melgaço
11	Niepoort Vinhos	Niepoort Vinhos, S.A.	54	351-22 338 9528	351-22 338 9529	Porto
18	Casa de Villar	Casa de Villar, Lda.	75	351-255 91 1380	351-255 91 1381	Lousada
21	Barbeito	Vinhos Barbeito, S.A.	80	351-291 76 1829	351-291 76 1830	Funchal

Lojas

#Loja	*NomeLoja*	*localLoja*	*IpLoja*
48	Só Uvas e Bagos	Faro	32
49	Uvas Atenas Portas	Idanha-a-Velha	77

Quantidades

#Vinho	*#Loja*	*qtdd*
1	48	2000
1	49	2000
99	49	1500
34	49	75
116	49	3500
789	49	5

Figura 4.45 Tabelas para a base de dados WiNet em terceira forma normal – 3FN

4.6.6 Verificar actualizações e consultas

Nesta fase deveria ser verificado se a base de dados da Figura 4.45 permite responder às questões que podem ser colocadas. Esta verificação deve ser feita do ponto de vista funcional e também pode ser feita do ponto de vista não funcional. Neste último caso poderia ser necessário regressar a uma base de dados menos normalizada para permitir consultas eficientes.

De momento vamos aceitar a base de dados como se encontra.

4.6.7 Propor modelo relacional

Se não houver nenhuma alteração, então a estrutura conceptual identificada seria agora a base para iniciar o processo de implementação, recorrendo a uma ferramenta comercial.

4.7 Conclusões

As bases de dados organizadas segundo modelos genéricos tais como o modelo relacional servem para um número muito elevado de aplicações e de sistemas de informação. No entanto há e haverá sempre muitas situações em que a construção ou utilização de uma base de dados de um modelo existente não é a solução mais apropriada.

Cada problema de gestão de informação deve ser analisado com cuidado. Uma dimensão pequena ou baixa complexidade pode permitir a utilização de outras tecnologias. Aplicações de gestão de dados com base em folhas de cálculo (por exemplo Excel ou Lotus) ou com base em sistemas de documentos estruturados (por exemplo Lotus Notes), podem ser boas alternativas.

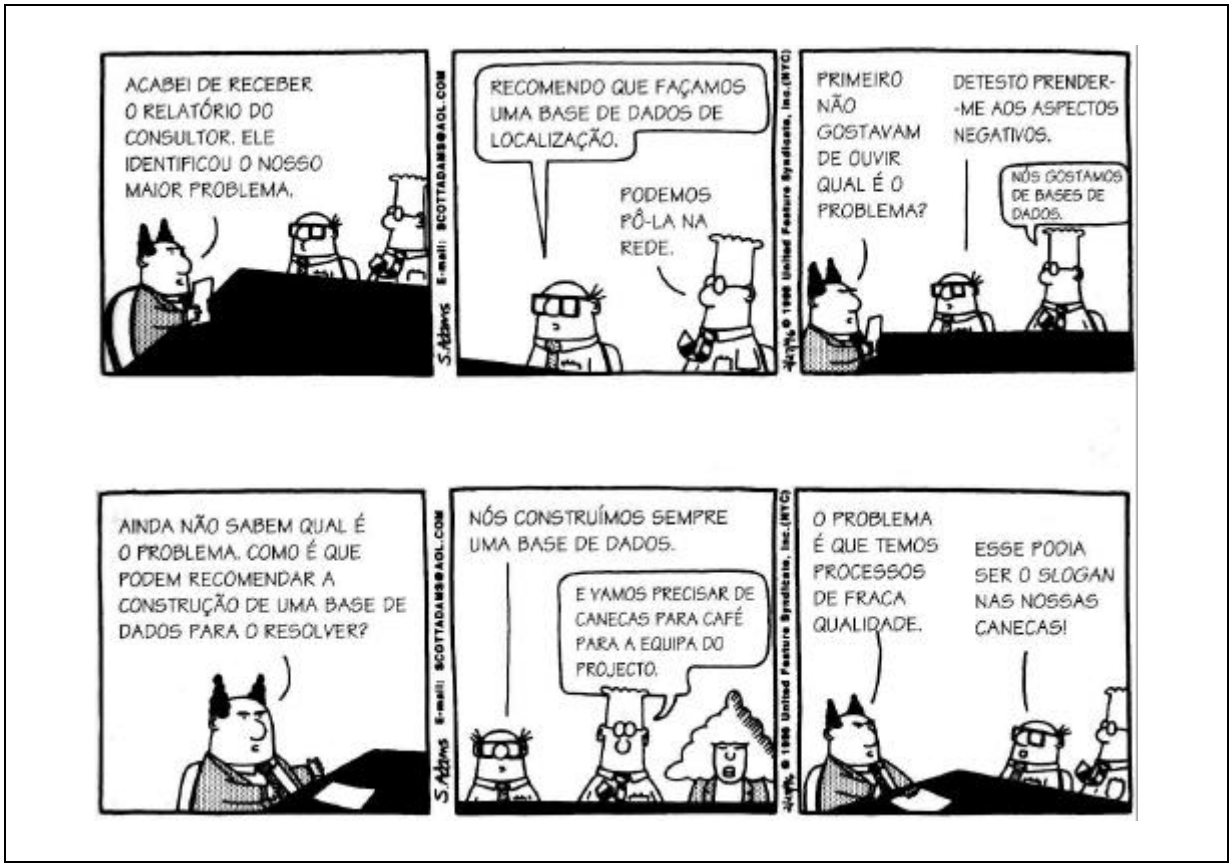


Figura 4.46 "Nós construímos sempre uma base de dados" ([Adams 1997], p. 67)

Da mesma forma sitauções de dimensão demasiado elevada ou de enorme complexidade podem inviabilizar as bases de dados relacionais. Como exemplo referem-se os sistemas para manutenção de dados estatísticos em bruto ou mesmo com baixos níveis de agregação, onde se tem de utilizar ficheiros clássicos, arquivados em bandas ou fitas magnéticas. É também impossível efectuar análise de qualidade de dados directamente sobre um SGBD relacional de produção. Os tempos de execução das consultas não só iriam perturbar o normal funcionamento dos sistemas, degradando os tempos de resposta dos utilziadores normais, como de facto poderiam tomar mais tempo do que o disponível para efectuar o diagnóstico

desejado. Seria normalmente mais eficiente analisar exuastivamente a qualidade dos dados de um SGBD trnasferindo toda a informação para um ficheiro de texto estruturado adequadamente e sobre esse ficheiro executar testes adequados programados numa linguagem apropriada (normalmente compilável).

Qeustões de compatibilidade com o passado, por exemplo nos sistemas de reserva de passagens em aviões ou em sistemas herdados, podem ainda impedir a utilização de bases de dados.

Neste capítulo abordou-se o modelo relacional de bases de dados. No entanto, e como já referido, há outros modelos para organização de dados em utilização e em investigação. Esta secção termina com uma breve referência aos modelos alternativos, incluindo os antepassados do modelo relacional.

Outros Sistemas Baseados em outros Modelos

Os primeiros SGBD surgiram como colecções estruturadas de ficheiros. Estes ficheiros tinham boas propriedades para permitirem acesso rápido aos dados, por exemplo directamente a uma ficha individual, e para garantir igualmente a manutenção de diversas ordens para pesquisa. A necessidade de estabelecer hierarquias e relacionamentos entre tipos de informação levou à colocação nos ficheiros de apontadores. No início estes apontadores tinham uma estruturação hierárquica em árvores, em que era simples navegar do ficheiro original para os seguintes. Estas bases de dados seguiam o designado **modelo hierárquico** (por exemplo o sistema IMS da IBM). As limitações na eficiência de navegação levaram à adição de apontadores cruzados e associações, permitindo a navegação eficiente em qualquer direcção previamente desejada. Estas bases de dados passaram a ter a designação de **modelo em rede**, e baseavam-se nas especificações CODASYL. Em ambos os casos o modelo conceptual ficava ligado ao modelo de implementação, e era necessário conhecer os detalhes de implementação dos apontadores para definir as consultas à base de dados, sob risco de as consultas serem extremamente ineficientes. O modelo em rede viria a permitir a introdução de uma camada que permitia separar os aspectos conceptuais dos de implementação, permitindo uma visão independente e conceptual da informação. Sistemas de tecnologia baseada neste modelo passaram a oferecer interfaces de acesso em dialectos de SQL. Embora ainda estejam em utilização bases de dados organizadas destas formas, a tendência é a desaparecerem, excepto em algumas situações particulares. [Wiederhold 1983] apresenta em detalhe a organização dos sistemas de gestão de bases de dados baseados neste modelos, a partir da perspectiva de implementação com ficheiros, e discute em pormenor as medidas de desempenho.

Uma das grandes vantagens da linguagem SQL é a possibilidade de ser interactiva. De facto uma das regras a que a sua concepção obedeceu era a de ser acessível a utilizadores sem experiência de programação. Em particular, a linguagem SQL não deveria necessitar de programação com ciclos ou de recursão. Esta visão de simplicidade assim oferecida pelo modelo relacional acabou por não se concretizar e na prática raros são os utilizadores que dominam SQL. Um dos problemas é que a organização da informação em tabelas ou relações com chaves primárias e chaves alheias que estabelecem as ligação entre objectos obriga a alguns artificialismos, e ao conhecimento explícito dos atributos de associação, não sendo o nível de abstracção ainda o adequado. Um SGBD que permitisse gerir informação estruturada de acordo com noções de objectos e de classes seria mais abstracto, ou seja mais próximo das

intuições comuns[25], e por isso melhor para os utilizadores "normais", pois estaria mais afastado do modelo computacional. Diversas propostas surgiram para criar SGBD orientados por objectos, alguns com estratégias comerciais ambiciosas (por exemplo o sistema O2). No entanto e aparentemente o modelo que tem sido de facto utilizado é híbrido, mantendo uma forte componente relacional. Os produtos novos aparecem com a designação de **modelo objecto-relacional** e a nova norma de SQL, a SQL-3, virá estabelecer um conjunto de regras nesta área.

A investigação em Inteligência Artificial que teve um forte crescimento a nível mundial no início da década de 1980 com o programa japonês da Quinta Geração e os vários programas de investigação que se lhe seguiram na Europa e EUA. O programa Japonês era baseado no paradigma da programação em lógica. Os esforços feitos a nível internacional no desenvolvimento do paradigma lógico resultaram igualmente em extensões para gestão de dados designadas **modelo lógico e dedutivo**. Os sistemas de modelo relacional foram alargados com base no cálculo de predicados e deram origem aos chamados bases de dados de conhecimentos, onde era possível realizar operações de inferência e descrever regras de integridade muito mais complexas de uma forma mais simples (ver [Kowalski 1982]). Nestes sistemas tornava-se possível obter com facilidade o fecho transitivo de uma relação, por exemplo com base numa consulta declarativa na linguagem PROLOG (recursiva) e modelar aspectos temporais (ver por exemplo [Allen 1984]).

[25] A cultura ocidental, com fortes raízes na filosofia grega, é fortemente orientada para a identificação ou abstracção do conhecimento em torno de objectos «perfeitos». O conhecimento dos objectos reais resulta em grande parte do sucesso dos processos de análise em partes e de classificação em tipos. A física, genética, biologia, psicologia e as restantes ciências, são conduzidas por esta epistemologia ou teoria do conhecimento, em que a classificação sistemática dos objectos observados e a sua decomposição em partes atómicas é orientada por «formas perfeitas». Mais recentemente, a partir do Renascimento, o método experimental ou científico vem dar um contributo inovador ao processo de aquisição e validação do conhecimento (ver por exemplo [Popper 1959]).

Por exemplo, Carl von Linneo (Carolus Linnaeus 1735) no séc. XVIII classificou e ordenou todos os seres vivos conhecidos, animais e plantas, em reinos, classes, ordens, famílias, géneros e espécies, e estabeleceu os nomes científicos das espécies em latim, tendo por exemplo chamado ao homem *Homo Sapiens*. Foi neste trabalho sistemático de classificação de Linneo em que mais tarde Charles Darwin se iria basear para propor a sua tese da evolução das espécies. A análise e comparação cuidadosa da informação contida no ADN dos seres vivos (sequenciamento de cromossomas), no fundo um trabalho conceptualmente com semelhanças com o de Linneo, tem permitido conclusões muito mais precisas e detalhadas relativamente às semelhanças e diferenças entre as espécies, permitindo decomposições e classificações úteis para engenharia genética.

4.8 Normas utilizadas

4.8.1 Normas e definições utilizadas

Regras para escrever identificadores

Tipo de Identificador	**Exemplos**	**Regras**	**Observações**
Nome de classe Nome de relação Nome de tabela	**ProdutorDeVinho** **Fornecedor**	Letra inicial da primeira palavra deve ser maiúscula A-Z. Palavras que constituem o identificador devem ser escritas sem espaço separador. Letras iniciais das restantes palavras que constituem o identificador devem ser maiúsculas.	Arial, 9 pontos, Negrito. A classe com o nome: **PV ou ProdutorDeVinho** ou com o nome: **PV (ProdutorDeVinho)** pode ser referida por **PV** ou por **ProdutorDeVinho**.
Variável de relação	**A**, **B**, **R**		
Nome de atributo de uma relação Nome de coluna em tabela	*#Produtor* *nomeProdutor* *cidadeDoProdutor* *$orçamentoProjecto* *%execução* *dataInicial*	Letra inicial pode ser um símbolo especial. Letra inicial da primeira palavra deve ser minúscula. Palavras que constituem o identificador devem ser escritas sem espaço separador e as letras iniciais das palavras seguintes devem ser maiúsculas. O atributo não deve terminar com um ponto «.». Os símbolos iniciais têm a seguinte leitura: # código. $ custo, preço ou valor. % percentagem.	Arial, 9 pontos, Itálico. Os atributos são utilizados no modelo entidade-associação, modelo de objectos, modelo de classes e modelo relacional. O atributo pode referir explicitamente a unidade de medida; por exemplo: *$orçamentoProjecto (euro).* Este atributo pode ser referido apenas por *$orçamentoProjecto*.
Valor de um atributo	08974 “José Silva” “Porto” 100.000 euro 75.45% 2000-10-29	Nomes entre aspas. No texto devem ser indicadas as unidades de medida utilizadas, excepto quando for óbvio do contexto, ou quando o atributo as indicar explicitamente.	Times New Roman, 12 pontos.

Um domínio, tal como um conjunto, é um objecto matemático ou abstracto que é definido a partir dos valores que o formam. Os atributos podem tomar valores em domínios. Vários atributos com nomes diferentes podem tomar valores num mesmo domínio. Os nomes de domínios e de variáveis são escritos no texto normal em *Times New Roman,* itálico, com 12 pontos.

As expressões da álgebra e do cálculo relacional são escritas no texto normal em Arial, com 10 pontos. Exemplos:

nome: *Texto* (o atributo “nome” toma valores no domínio “Texto”. Repare-se que neste contexto o domínio *Texto* não tem limitações em termos de números de caracteres, como acontece por exemplo numa definição em SQL).

Projecção de **Loja** em nome (expressão de álgebra relacional, onde a relação "Loja" é escrita de acordo com as convenções habituais).

Regras para apresentar relações ou tabelas

Forma gráfica: como na Figura 4.19, utilizando as convenções anteriores, e identificando os atributos da chave primária com um sublinhado (duplo). Pode não se representar qualquer informação nas linhas e podem não se indicar as chaves alheias:

V (Vinhos)					
#Vinho	*nomeVinho*	*cl*	*cor*	*região*	*#Produtor*

Forma textual: utilizando as convenções anteriores, identificando os atributos da chave primária com sublinhado e as chaves alheias com um sublinhado tracejado (opcional).

V{*#Vinho, nomeVinho, cl, cor, região, #Produtor*}

Regras para apresentar dependências funcionais

Forma gráfica: como na Figura 4.43, utilizando as convenções anteriores. É opcional a identificação das dependências funcionais que não se verificam (seta com uma cruz).

Forma escrita: utilizando as convenções anteriores. É opcional a identificação das dependências funcionais que não se verificam. Por exemplo: *#Vinho→nomeVinho, região, #Produtor*.