

Departamento de Tecnologias

*Tecnologias e Programação de Sistemas de Informação*

## **Bases de Dados**

Ano letivo 2022-2023

Autor: Secundino Domingos Marques Lopes

([secundino.lopes@ipportalegre.pt](mailto:secundino.lopes@ipportalegre.pt))

Última edição por: Pedro Miguel da Silva Roque

([pedro.roque@ipportalegre.pt](mailto:pedro.roque@ipportalegre.pt))

## **Sebenta de apoio**

|    |   |     |
|----|---|-----|
| 1. | Introdução aos Sistemas de Base de Dados..... | 6   |
| 2. | Modelo Relacional .....                       | 27  |
| 3. | SQL – <i>Structured Query Language</i> .....  | 48  |
| 4. | Linguagem T- SQL.....                         | 119 |
| 5. | Abordagem Entidade/Associação .....           | 152 |



## Conteúdo

|   |    |
|---|----|
| 1. Introdução aos Sistemas de Base de Dados.....        | 6  |
| 1.1 Introdução .....                                    | 6  |
| 1.2 Dados <i>versus</i> Informação .....                | 8  |
| 1.3 Sistemas de Gestão de Ficheiros .....               | 9  |
| 1.4 Sistemas de Bases de Dados .....                    | 12 |
| 1.5 O Sistema de Gestão de Bases de Dados .....         | 13 |
| 1.5.1 A Arquitetura ANSI/SPARC .....                    | 14 |
| 1.5.2 Requisitos Fundamentais de um SGBD .....          | 15 |
| 1.5.3 Utilizadores de Sistemas de Bases de Dados .....  | 25 |
| 2. Modelo Relacional .....                              | 27 |
| 2.1 Conceitos .....                                     | 27 |
| 2.2 Relações entre tabelas .....                        | 32 |
| 2.3 Tipos de Dados .....                                | 35 |
| 2.4 Interfaces ao Modelo Relacional .....               | 38 |
| 2.5 As Doze Regras de Codd.....                         | 42 |
| 2.6 Exercícios do Modelo Relacional .....               | 43 |
| 3. SQL – <i>Structured Query Language</i> .....         | 48 |
| 3.1 SQL – <i>Data Manipulation Language</i> (DML) ..... | 48 |
| 3.1.1 Operadores.....                                   | 53 |
| 3.1.1.1 Operadores relacionais .....                    | 53 |
| 3.1.1.2 Operadores lógicos.....                         | 55 |
| 3.1.2 Alias ( <i>renaming</i> e <i>aliasing</i> ) ..... | 57 |
| 3.1.3 Subconsultas SQL.....                             | 57 |
| 3.1.4 Ordenação .....                                   | 60 |
| 3.1.5 Junção, restrição, projeção e ordenação .....     | 60 |
| 3.1.6 Campos calculados .....                           | 61 |
| 3.1.7 Funções de Datas e Horas .....                    | 62 |
| 3.1.8 Funções que devolvem valores de caracteres.....   | 65 |
| 3.1.9 Funções de agregação .....                        | 68 |
| 3.1.10 A cláusula <i>Group By</i> .....                 | 69 |
| 3.1.11 A cláusula <i>Having</i> .....                   | 71 |
| 3.1.12 SELECT ... CASE.....                             | 72 |
| 3.1.13 SQL – Join (variantes).....                      | 73 |
| Cross join.....   | 73 |
| Equi-join .....   | 73 |
| Natural join .....                                      | 74 |

|  |     |
|--|-----|
| Inner join.....  | 74  |
| Outer join (em SQL Server) .....                         | 75  |
| • Left join .....  | 75  |
| • Right join .....                                       | 76  |
| • Full join .....  | 76  |
| 3.1.14 Operações sobre conjuntos.....                    | 77  |
| Union.....   | 77  |
| Intersect .....  | 78  |
| Except.....  | 78  |
| 3.1.15 Exercícios de SQL_DML .....                       | 79  |
| 3.2 SQL – <i>Data Definition Language</i> (DDL) .....    | 82  |
| 3.2.1 Create Database .....                              | 82  |
| 3.2.2 CREATE TABLE .....                                 | 84  |
| 3.2.2.1 CONSTRAINTS.....                                 | 86  |
| 3.2.2.2 Primary KEY .....                                | 88  |
| 3.2.2.3 Foreign KEY .....                                | 89  |
| 3.2.3 Alter Table.....                                   | 95  |
| 3.2.4 Drop Table .....                                   | 96  |
| 3.2.5 Truncate Table .....                               | 97  |
| 3.2.6 INSERT .....                                       | 97  |
| 3.2.7 UPDATE.....  | 101 |
| 3.2.8 DELETE .....                                       | 104 |
| 3.3 SQL VIEWS/Consultas.....                             | 106 |
| 3.3.1 Criar e gerir Views .....                          | 108 |
| 3.3.2 <i>Views</i> de ação.....                          | 111 |
| 3.4 Exercícios práticos de SQL-DDL.....                  | 113 |
| 4. Linguagem T- SQL.....                                 | 119 |
| 4.1 Variáveis .....                                      | 119 |
| 4.2 Mecanismos de controlo da linguagem T-SQL.....       | 123 |
| 4.3 Stored Procedures .....                              | 126 |
| 4.3.1 Tipos de Stored Procedures.....                    | 126 |
| 4.3.2 Parâmetro <i>Default</i> .....                     | 130 |
| 4.3.3 Argumento <i>With Encryption</i> .....             | 131 |
| 4.3.4 Execução Automática .....                          | 131 |
| 4.3.5 Obter o código fonte de um procedimento.....       | 132 |
| 4.3.6 Retorno de dados, parâmetros OUTPUT e RETURN ..... | 132 |
| 4.3.7 Aplicação prática de procedimentos .....           | 133 |

|        |   |     |
|--------|---|-----|
| 4.4    | <i>Triggers</i> .....   | 138 |
| 5.     | Abordagem Entidade/Associação .....                                   | 152 |
| 5.1    | Entidade.....   | 154 |
| 5.2    | Associação, aridade e multiplicidade (tipos de associação) .....      | 154 |
| 5.2.1  | Associação .....  | 154 |
| 5.2.2  | Aridade.....  | 155 |
| 5.2.3  | Multiplicidade (tipos de associação ou cardinalidade).....            | 156 |
| 5.3    | Papeis das Entidades nas Associações .....                            | 159 |
| 5.4    | Atributo.....   | 160 |
| 5.5    | Chave .....   | 161 |
| 5.6    | Entidades Fracas .....  | 161 |
| 5.7    | Particularizações/Generalizações (ISA).....                           | 162 |
| 5.8    | Associações Recursivas .....  | 163 |
| 5.9    | Restrições de Existência e de Identidade.....                         | 163 |
| 5.10   | Obrigatoriedade de participação de uma entidade numa associação ..... | 164 |
| 5.11   | Transformação para o Modelo Relacional .....                          | 164 |
| 5.11.1 | Associações 1:N .....   | 165 |
| 5.11.2 | Associações M:N .....   | 165 |
| 5.11.3 | Casos particulares. Associações recursivas .....                      | 166 |
| 5.11.4 | Casos particulares. Associações ternárias .....                       | 166 |
| 5.11.5 | Casos particulares. Generalização .....                               | 167 |
| 5.12   | Exercícios de ER.....   | 168 |

# 1. Introdução aos Sistemas de Base de Dados

## 1.1 Introdução

A divulgação das Tecnologias da informação, genericamente designadas por Informática, nas organizações, aconteceu segundo um processo gradual ditado, por um lado, pelos sucessivos avanços tecnológicos a nível dos equipamentos e *software* e, por outro lado, pelo crescente reconhecimento das suas potencialidades no tratamento de dados.

Inicialmente, dadas as suas características de rapidez de processamento e capacidade de armazenamento, era comum a sua utilização no apoio ao processamento de dados, nomeadamente, na automatização de tarefas manuais. Neste contexto, apareceram os chamados sistemas de gestão de ficheiros, tipicamente associados à linguagem de programação Cobol (*Common Business Oriented Language*). Utilizando esta tecnologia, ao longo dos anos, muito esforço foi despendido e grandes investimentos foram feitos no desenvolvimento de sistemas e na sua manutenção.

À medida que os utilizadores se foram apercebendo das potencialidades oferecidas por esta tecnologia, cresceram as solicitações de novas aplicações, cada vez mais complexas e exigentes em termos de desenvolvimento. A informática depressa alargou o seu terreno de atuação passando a ser também encarada como uma ferramenta importante de apoio à decisão nas organizações.

Este aumento na procura de "soluções informáticas", acompanhado da sua maior complexidade, originou o aparecimento de um grave problema conhecido por *backlog* de aplicações. Os utilizadores solicitam novas aplicações a que o departamento de desenvolvimento, dada a sua carga de trabalho, não consegue dar resposta. Grande parte do seu tempo, cerca de 75%, é gasta na manutenção dos sistemas existentes. Desta forma, quantos mais sistemas são desenvolvidos, maiores são as necessidades de manutenção e, portanto, menor a disponibilidade para desenvolver novos sistemas. Torna-se então urgente a utilização de uma nova tecnologia que reduza o tempo de desenvolvimento de novos sistemas e facilite, posteriormente, a sua manutenção.

Em termos de evolução tecnológica, as organizações mais recentes, ou com um "passado informático" mais recente, tendem a beneficiar pelo facto de não terem investido em tecnologia, hoje considerada obsoleta. Com efeito, o maior obstáculo a uma modernização mais rápida das infraestruturas tecnológicas, em algumas organizações, é precisamente os sistemas existentes (hardware + software), desenvolvidos com base em tecnologia mais antiga, mas ainda em plena exploração.

Se, por um lado, é essencial uma atitude de constante modernização por parte das organizações, com a adoção das tecnologias mais recentes, naturalmente mais produtivas, por outro lado, não é, geralmente, fácil o processo de transição de uma tecnologia para outra. Quer pelos investimentos já feitos em hardware, em software, em formação do pessoal, etc. e que, muito provavelmente, serão desaproveitados; quer pela dificuldade em justificar as vantagens da mudança para uma nova tecnologia, perante os custos incorridos; quer pelo risco envolvido na mudança para uma tecnologia desconhecida - no fundo trata-se de abandonar algo que, apesar de poder sofrer de vários problemas e limitações, tem vindo a suportar algumas das necessidades de processamento das organizações; quer ainda pelas possíveis dificuldades técnicas colocadas pela conversão dos sistemas existentes de uma tecnologia para outra.

Por tudo isto, os processos de evolução tecnológica quando envolvem a substituição de uma tecnologia por outra radicalmente diferente tendem a ser mal recebidos pelas organizações. A evolução tecnológica, na generalidade das organizações, dá-se de forma gradual, segundo

critérios objetivos de custo/benefício. A única forma de qualquer nova tecnologia vingar no mercado é conseguir demonstrar, de forma efetiva, os benefícios que traz a sua adoção.

Historicamente, devido, em grande parte, às condições em que decorreu a sua implantação, a informática desenvolveu-se, na generalidade das organizações, de forma claramente descoordenada, por vezes mesmo caótica. Para isso contribuíram um conjunto de fatores, de entre os quais se podem destacar:

Por condicionalismos da própria evolução tecnológica. Com efeito, dado o estado atual de desenvolvimento da tecnologia (hardware, software e comunicações) surgem, hoje, possibilidades que antes não existiam. Naturalmente, as "soluções informáticas" estão dependentes e, por isso, condicionadas por aquilo que a tecnologia permite. Ou seja, soluções, hoje viáveis, antes não eram possíveis. Da mesma forma, soluções, hoje impossíveis de pôr em prática, poderão sê-lo no futuro. Inevitavelmente, os modernos sistemas de hoje serão os sistemas legados do futuro.

Devido à falta de planeamento. Tradicionalmente, o desenvolvimento de sistemas sempre se caracterizou por alguma anarquia. Os sistemas foram desenvolvidos independentemente uns dos outros, sem preocupações de comunicação entre si. Sem a noção de solução global, o resultado é um conjunto de sistemas autónomos, com fraca interação. Desta forma, a "solução informática" não traduz, ou traduz deficientemente, a forma de funcionamento da organização.

Mais recentemente, devido aos problemas decorrentes da divulgação desenfreada e descoordenada da microinformática nas organizações, os utilizadores desenvolveram as suas próprias aplicações para resolver as suas necessidades particulares, sem a preocupação da integração no todo da organização.

O planeamento da informática diz respeito a toda a organização, sendo necessário considerar a totalidade dos seus requisitos de informação.

A forma atual de encarar a informatização tenta ver a organização como um todo, recomendando a utilização da informática como um apoio efetivo ao Sistema de Informação da organização.

Uma vez que o constituinte central de qualquer sistema de informação é a sua memória (conjunto armazenado de dados), a "solução informática", para qualquer organização, deve assentar num depósito integrado de dados – a Base de Dados.

*Uma base de dados é, por definição, um conjunto organizado de dados, disponível a todos os utilizadores ou processamentos da organização que deles tenham necessidade.*

A tecnologia de bases de dados vai, então, tentar responder a dois objetivos; por um lado, dar corpo a uma forma mais natural de pensar os sistemas de informação, surgindo como elemento integrador dos recursos informacionais da organização; por outro lado, disponibilizar meios de desenvolvimento de mais alto-nível, capazes de acelerar o processo de desenvolvimento de novos sistemas e facilitar a manutenção dos sistemas construídos segundo esta tecnologia. Desta forma, dão-se passos no sentido de reduzir o grave problema do *backlog*.

### **Base de Dados**

*Colecção de dados inter-relacionados e armazenados, sem redundância desnecessária, acessíveis a um n.º vasto de utilizadores que a eles podem ter acesso em simultâneo e independentes dos programas que os utilizam.*

*Funciona como infra-estrutura do Sistema de Informação de uma empresa. Um dos seus bens mais preciosos.*

## 1.2 Dados *versus* Informação

A informação é encarada, atualmente, como um dos recursos mais importantes de uma organização, contribuindo decisivamente para a sua maior ou menor competitividade. De facto, com o aumento da concorrência tornou-se vital melhorar as capacidades de decisão a todos os níveis.

Hoje, mais do que nunca, a tomada de decisão nas organizações é um processo complexo, dada a quantidade de informação em jogo, a sua complexidade e a frequência com que se altera. No entanto, para que possa ser utilizada como um apoio eficaz à tomada de decisão, a informação só tem valor se, se verificarem, simultaneamente, algumas condições:

- Atualidade - O valor da informação dependerá em grande parte da sua atualidade. Dado o dinamismo verificado em todos os sectores da sociedade em geral e do ambiente empresarial em particular, o período de validade da informação é cada vez mais curto. Torna-se necessário dispor de fontes de informação que acompanhem continuamente essas modificações. Só com base em informação atualizada se podem tomar decisões acertadas.
- Correção - Não basta que a informação seja atual, é também necessário que, na medida do possível, seja rigorosa. Só com informação correta se pode decidir com confiança.
- Relevância - Dado o grande volume de informação envolvida, o processo de tomada de decisão, ao contrário de ser facilitado, pode ser dificultado pelo excesso de informação. A informação deve ser devidamente filtrada de tal forma que apenas aquela com relevância para cada situação seja considerada.
- Disponibilidade - Ainda que a informação verifique os três requisitos anteriores, a sua utilidade poderá ser posta em causa se não puder ser disponibilizada de forma imediata, no momento em que é solicitada. As decisões muito ponderadas, com longo período de gestação, são, cada vez mais, situações do passado. Hoje, dadas as características do meio envolvente, o processo de tomada de decisão tem que ser quase instantâneo. Para isso, a informação tem que ser disponibilizada rapidamente, caso contrário deixa de ser útil.
- Legibilidade - Esta condição, apesar de apresentada em último lugar não é, por isso, menos importante. A informação só é informação se puder ser interpretada. De facto, de nada vale que a informação seja atual, precisa, relevante e disponibilizada em tempo oportuno se não puder ser entendida. A forma como é disponibilizada tem também grande importância.

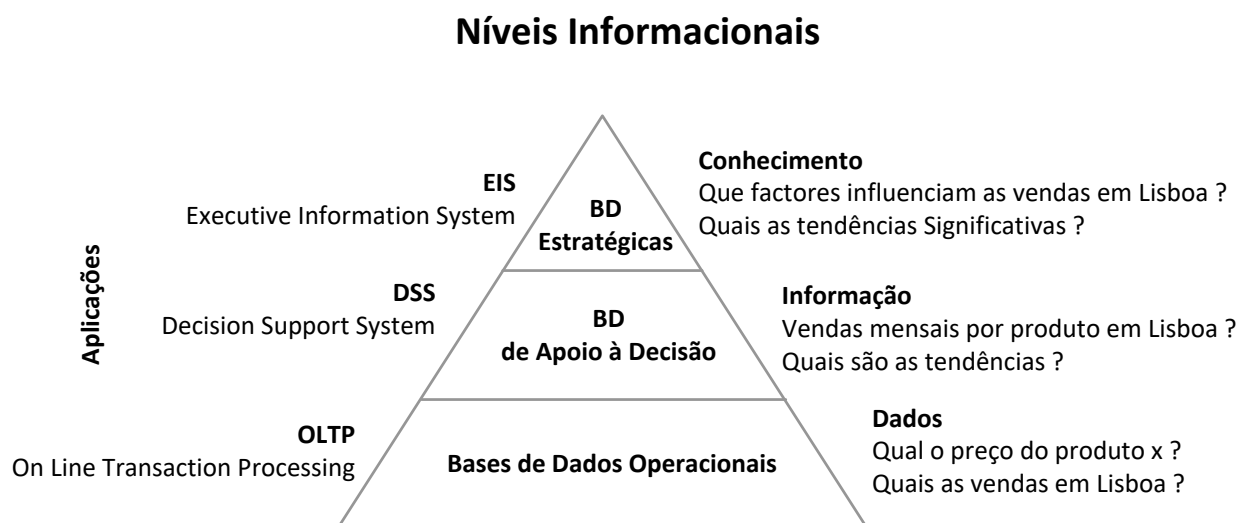
Um outro conceito intimamente relacionado com a informação é o conceito de dados. Dados e informação são coisas distintas. *Dados são apenas elementos ou valores discretos que, isoladamente, não têm qualquer valor*, só se transformam em informação quando relacionados ou interpretados de alguma forma. Ou seja, a informação é o resultado de alguma forma de processamento sobre dados. Os dados podem ser vistos, simplesmente, como a matéria-prima necessária a esse processamento.

Quanto aos requisitos de atualidade e correção da informação, para que esta possa ser correta e atual os dados de onde é derivada têm, forçosamente, que ser precisos e atualizados. Quanto aos requisitos de relevância, disponibilidade e legibilidade da informação, estes têm mais a ver com os meios utilizados para o processamento dos dados.

Neste contexto, as tecnologias da informação desempenham um papel fundamental, sendo hoje apontadas como imprescindíveis ao processo de tomada de decisão nas organizações.



De facto, já vão longe os dias em que, por processos exclusivamente manuais, se conseguia obter a informação com as características referidas atrás.



**Figura 1 - Níveis informacionais**

De entre as tecnologias da informação, a tecnologia de bases de dados tem, neste contexto, um interesse particular. Concretamente, esta tecnologia vem contribuir, de forma significativa (como se compreenderá mais à frente), para a viabilização dos requisitos de correção e atualização dos dados, fornecendo meios e ferramentas para a extração da informação relevante, na altura em que é necessária e com o formato adequado. Contribuindo, desta forma, para a qualidade da informação fornecida, melhorando os serviços prestados pelas tecnologias da informação.

### 1.3 Sistemas de Gestão de Ficheiros

Os antecessores, em termos de evolução tecnológica, dos modernos sistemas de bases de dados foram os sistemas de gestão de ficheiros. De entre estes destacaram-se, de forma particular, os sistemas desenvolvidos recorrendo à linguagem de programação COBOL, a ponto de ainda hoje existirem milhares de sistemas deste tipo em exploração e mesmo em desenvolvimento. A maioria destes sistemas foi convertida com a atualização dos sistemas para o *bug* do ano 2000.

Utilizando este tipo de tecnologia, como já foi referido anteriormente, as organizações começaram por automatizar algumas das tarefas até aí realizadas manualmente. Nesse sentido, aquelas tarefas que, pela grande quantidade de trabalho envolvido, nomeadamente de tratamento de dados, e que pelas suas características de processamento rotineiro mais se adequavam a um tratamento automático, foram sendo sucessivamente informatizadas recorrendo a esta tecnologia.

Contudo, inicialmente, esta informatização resumia-se à simples automatização de processos manuais, até aí suportados em papel, tal e qual eram executados anteriormente. O resultado final era que os processos continuavam a ser executados, no essencial, da mesma forma, só que mais rapidamente. Ora, aqui é que reside o principal problema do percurso evolutivo das tecnologias da informação nas organizações.

Dada a forma como os fluxos de informação se foram estabelecendo nas organizações, não é invulgar encontrar situações em que, por exemplo, um mesmo documento é criado em várias vias (original + 1ª via + 2ª via + ...), cada uma delas com um destino e um processamento diferentes (ver Figura 2).

Devido a esta situação, foram sendo desenvolvidos sistemas diferentes para cada um dos processamentos específicos existentes na organização, resultando em múltiplas entradas, dos mesmos dados, em sistemas distintos.

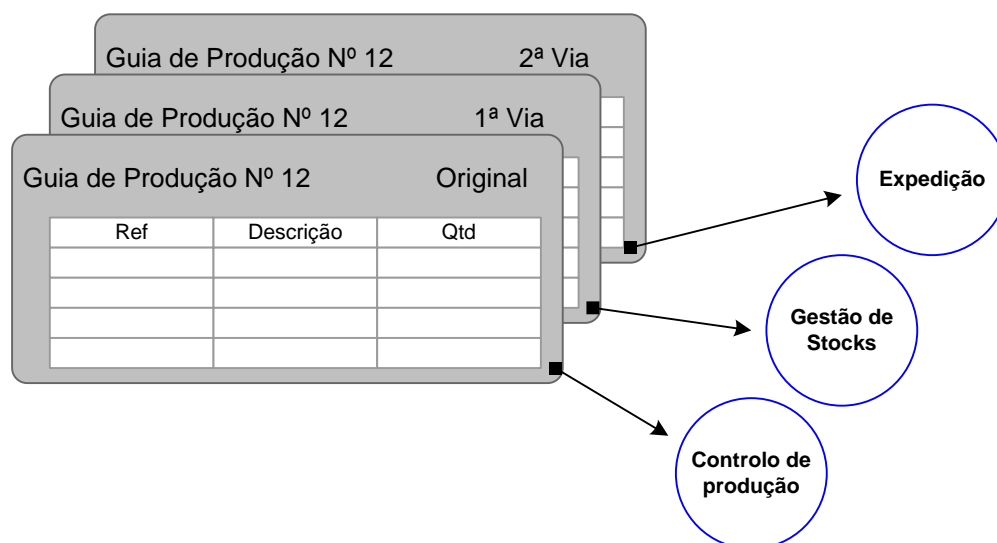


Figura 2 - Replicação de um documento por vários destinos

Recorrendo ao exemplo da Figura 1, ao original do documento Guia de Produção seria dado entrada no sistema de Controlo de Produção, enquanto à 1ª via do mesmo documento seria dada entrada no sistema de Gestão de Stocks, ao passo que à 2ª via desse mesmo documento seria dada entrada no Sistema de Expedição.

Como os sistemas eram desenvolvidos autonomamente, para cada nova aplicação identificada, definiam-se novos ficheiros de dados e desenvolviam-se os programas necessários aos processamentos específicos.

Assim sendo, os sistemas surgiam no panorama informático das organizações inopinadamente, sem qualquer relação com os sistemas já existentes, constituindo "ilhas isoladas" entre si (ver Figura 3).

Desta forma, os mesmos dados são armazenados em ficheiros diferentes e recolhidos em alturas diferentes por diferentes aplicações. Como são atualizados, independentemente, pelas respetivas aplicações, existem grandes probabilidades de ocorrerem incoerências, ou seja, a existência de dados que se contradizem.

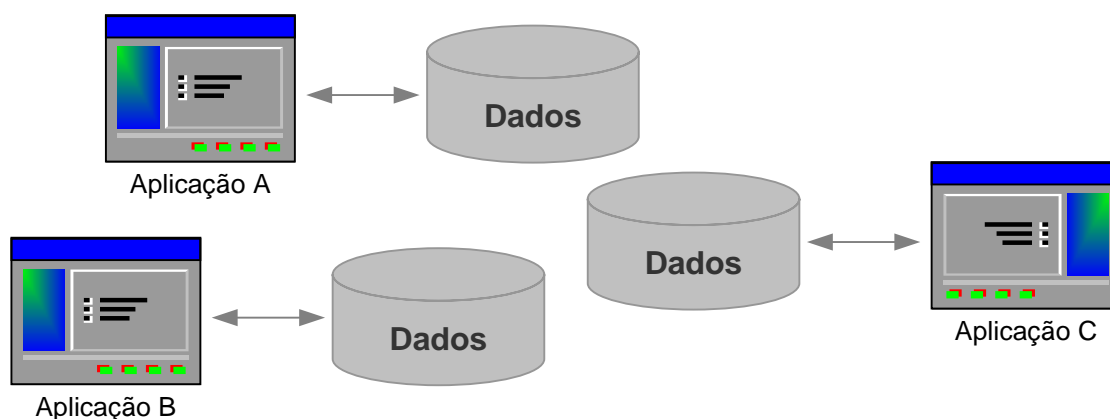


Figura 3 - Múltiplos sistemas isolados entre si

Naturalmente que a utilização de tecnologia de sistemas de gestão de ficheiros, só por si, não obriga à existência de toda esta redundância. De facto, é possível, com esta tecnologia,

desenvolver um outro tipo de solução segundo a qual, todos os dados necessários às várias aplicações são organizados num único conjunto de ficheiros, acessível a todas elas. Dessa forma, evita-se a redundância anterior e todos os problemas que ela acarreta. Contudo, ainda que se resolvam alguns problemas graves, persistem outros igualmente importantes.

Com efeito, uma das características principais dos sistemas de gestão de ficheiros é que os programas de aplicação que os constituem e os ficheiros de dados que processam se encontram ligados por interfaces físicas. Nestes sistemas, a estrutura física dos dados e a sua organização nos ficheiros é parte integrante da lógica dos programas. Sendo assim, uma aplicação que utilize um determinado ficheiro terá uma especificação do interface físico com esse ficheiro e, se esse ficheiro for utilizado por várias aplicações, a especificação desse interface terá que estar definida em cada uma das aplicações (Figura 4).

Esta característica tem consequências graves em termos de manutenção de sistemas. Basta que ocorra uma alteração num ficheiro para que essa alteração se propague, necessariamente, para todas as aplicações que o utilizam. Assim sendo, é evidente que a partilha de dados num ambiente de sistemas de gestão de ficheiros apresenta problemas ao nível da manutenção dos próprios sistemas.

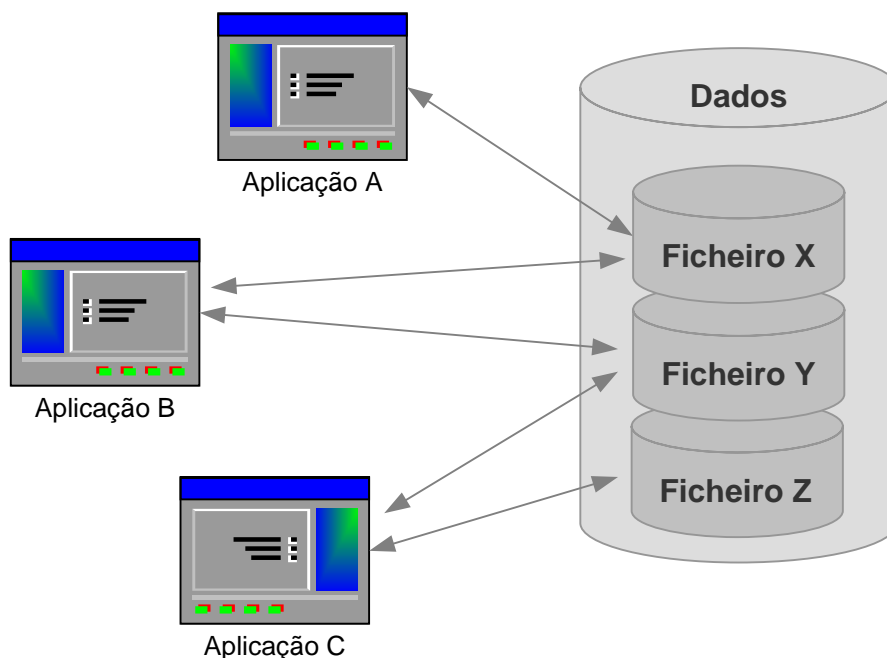


Figura 4 - Interfaces entre as aplicações e os ficheiros que utilizam

Mais grave que isso é o facto de os problemas do acesso concorrente aos dados partilhados terem de ser resolvidos ao nível das próprias aplicações, o que vem pôr em causa a fiabilidade dos próprios sistemas.

Por estas razões, mas também como consequência da forma como a informática evoluiu nas organizações, os sistemas que utilizam esta tecnologia evitam partilhar dados, são autónomos.

Ainda em relação á interface com os dados, é necessário realçar que os programas de aplicação interagem diretamente com o sistema operativo utilizando as funções disponibilizadas por este para aceder aos ficheiros. Trabalhando a tão baixo-nível é natural que o esforço de desenvolvimento e manutenção tenha tão baixa produtividade. Os sistemas de bases de dados surgem como tentativa de resolver estes problemas.

## 1.4 Sistemas de Bases de Dados

A abordagem pelos sistemas de bases de dados tem uma característica fundamental - os dados são organizados num único conjunto. Isto é, em vez de estarem separados por várias unidades independentes, os dados encontram-se integrados numa só unidade de armazenamento.

Adicionalmente, todos os acessos aos dados passam sempre por uma entidade designada **Sistema de Gestão de Bases de Dados** (SGBD) que centraliza em si o acesso físico à base de dados (ver Figura 5).

Ao contrário dos sistemas de gestão de ficheiros, as aplicações apenas têm uma interface - a interface com o SGBD e, mais que isso, essa interface é apenas um interface lógico e não físico. De facto, não tem qualquer interesse, para o nível aplicacional, conhecer os detalhes físicos de armazenamento e organização dos dados, desde que o SGBD os forneça no formato pretendido pelas aplicações.

A base de dados encontra-se, evidentemente, armazenada num conjunto de ficheiros, organizados em algum tipo de memória de características não voláteis, mas de forma transparente aos utilizadores e a todo o nível aplicacional, que não desce a esse nível de detalhe. O SGBD será, então, a única entidade que manipula a base de dados, atendendo todas as solicitações do nível aplicacional.

### SGBD

- É, basicamente, um sistema automático, computadorizado de armazenamento de dados, isto é, um sistema cujo objetivo global é registar e manter a informação.
- Conjunto de *software* especializado no tratamento de bases de dados.
- Proporciona o controle centralizado dos dados.
- Estabelece a interface/ligação entre os dados/informação e os utilizadores/programas.

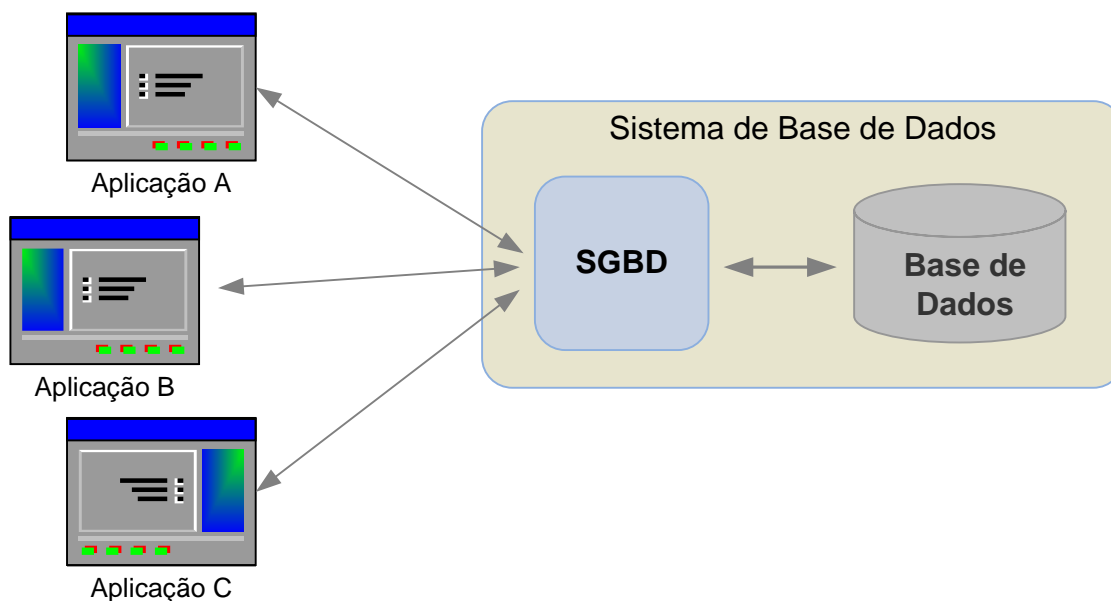


Figura 5 - O SGBD como interface entre as aplicações e a base de dados

A interface lógica entre o nível aplicacional e a base de dados é conseguido à custa do armazenamento na base de dados, não só dos dados propriamente ditos, mas também das suas descrições (*metadados*) numa entidade conhecida por dicionário de dados (também chamado catálogo). O dicionário de dados, atuando como um filtro, permite ao SGBD, entre outras coisas, interpretar a estrutura dos dados armazenados, disponibilizando ao nível aplicacional uma interface lógica.

Num ambiente de sistemas de gestão de ficheiros, o conceito de *metadados*, apesar de presente, é implementado de forma diferente. Nestes sistemas, ao contrário dos sistemas de bases de dados, o conceito é traduzido no nível aplicacional, incorporando os *metadados* nos programas que manipulam os ficheiros.

Desta forma é possível que o conteúdo de um mesmo ficheiro (dados) possa ser interpretado de forma diferente por aplicações diferentes (ou seja, com diferentes *metadados*).

A abordagem preconizada pelos sistemas de bases de dados é, sem dúvida, uma abordagem mais elegante já que permite, ao nível aplicacional, uma abstração total destes detalhes. Daí resulta uma aproximação de mais alto-nível ao desenvolvimento de sistemas, condição importante para conseguir uma maior produtividade no seu desenvolvimento e manutenção.

Apesar da tecnologia de bases de dados ter sido originalmente pensada como uma forma de reduzir ou eliminar a redundância e os problemas dela decorrentes, o principal ganho com esta tecnologia foi o isolamento ou separação conseguida entre os programas e os dados. Algo que, como se verá mais à frente, tenderá cada vez mais a atenuar-se com os novos sistemas de bases de dados.

## 1.5 O Sistema de Gestão de Bases de Dados

Devido às vantagens em manter um único conjunto lógico e organizado de dados, completamente autónomo das aplicações que os processam, torna-se necessário desenvolver um mecanismo que faça a gestão desse conjunto, garantindo que todas as solicitações do nível aplicacional à base de dados passem por si. Esse mecanismo é o SGBD (Sistema de Gestão de Bases de Dados).

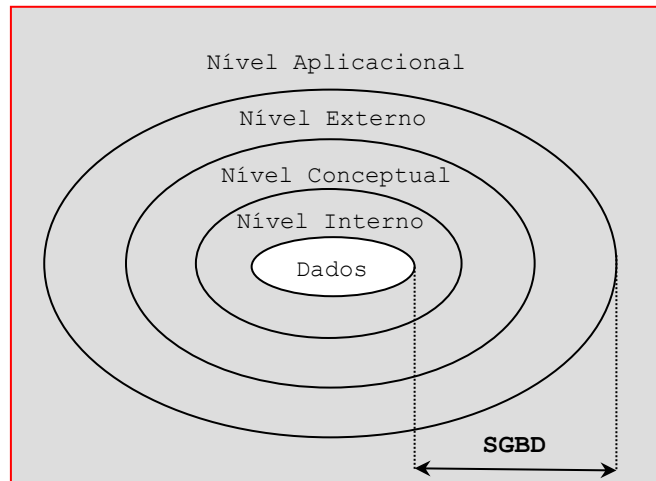
Necessariamente complexo, o SGBD, além de permitir a partilha concorrente dos dados, incorpora mecanismos que asseguram a validade, a segurança e a recuperação dos mesmos, em caso de falhas ou acidentes.

Por definição, o SGBD é um conjunto de *software*, destinado a gerir todo o armazenamento e manipulação dos dados do sistema, fazendo a interface entre o nível aplicacional e a base de dados propriamente dita.

Mais do que uma interface, que "esconde" os detalhes de armazenamento físico dos dados, o SGBD proporciona ao nível aplicacional um grau de abstração elevado, tal que, em certos modelos de bases de dados, nomeadamente nos mais recentes, o nível aplicacional trabalha com conceitos já muito próximos do mundo real. Com efeito, como será discutido mais à frente, a evolução ocorrida com a tecnologia de bases de dados tem sido no sentido de permitir ao nível aplicacional distanciar-se o mais possível da máquina, aumentando o nível de abstração dos conceitos que manipula.

### 1.5.1 A Arquitetura ANSI/SPARC

Numa tentativa de estabelecer um padrão para toda a "indústria" de desenvolvimento de tecnologia de bases de dados, foi proposto pela *American National Standards Institute* (ANSI), nomeadamente pelo seu *Standards Planning and Requirements Committee* (SPARC), uma arquitetura de três níveis independentes (arquitetura ANSI/SPARC), cada um deles descrevendo a base de dados a um nível diferente de abstração.



Arquitetura ANSI/SPARC

#### **Nível interno.**

Quando se refere o armazenamento físico dos dados (organização de ficheiros, métodos de acesso, etc.) numa base de dados, está-se a referir, implicitamente, o seu nível interno. Entidades, atributos ou outros elementos relativos ao modelo conceptual de dados não têm aqui um interesse direto. A principal preocupação, a este nível, será a definição das estruturas físicas que permitam obter um bom nível de desempenho.

*Neste nível são tratados pontos tais como métodos de acesso, técnicas de representação de associações e ambiente em que a Base de Dados vai operar. Embora estes pontos sejam de natureza física, este nível não tem de se preocupar, por exemplo, com as características específicas de um periférico magnético.*

Como é evidente, o nível interno deverá ser organizado com o objetivo de beneficiar as operações que, previsivelmente, ocorram com maior frequência, ou então, para proporcionar o melhor desempenho possível de todas as operações em geral.

#### **Nível conceptual.**

Também conhecido por esquema conceptual. Neste nível representa-se o modelo conceptual de dados, independentemente de qualquer utilizador ou aplicação particular, constituindo o chamado esquema ou estrutura da base de dados. Esta é a camada que permite esconder do nível aplicacional os detalhes de implementação física dos ficheiros que armazenam os dados.

*O nível Conceptual permite representar abstrações de subconjuntos do mundo real que sejam necessários a determinados projetos. Neste nível é necessário que não se façam referências acerca da estrutura de armazenamento ou estratégias de acesso aos dados para que se possa garantir a independência dos dados em relação ao software que os maneja;*

*É também neste nível que se encontram as verificações de autorização de acesso aos dados e os respetivos procedimentos de validação. Pode mesmo afirmar-se que o objetivo principal deste nível é o de manter descrições o mais completas possível dos dados, das formas de os utilizar, das transferências de informação que podem acontecer ou dos controlos de auditoria a aplicar.*

#### **Nível externo.**

Cada utilizador final não deve, a não ser que tenha necessidade disso, ser confrontado com a totalidade do esquema conceptual. Normalmente, para cada utilizador final é definida uma "vista" (*view*), estabelecendo como que uma janela sobre o esquema conceptual global, que

lhe permite trabalhar apenas com a parte do esquema onde estão os dados que lhe dizem respeito. Como é evidente, o mesmo se passa para cada aplicação. Aplicações diferentes são desenvolvidas com base em sub-esquemas diferentes, de acordo com os dados que necessitam processar.

Com esta arquitetura, obtêm-se algumas facilidades ou características de grande importância. Nomeadamente, é possível alterar a estrutura ou características de um nível sem ter que alterar, obrigatoriamente, o nível imediatamente superior. Surgem assim os conceitos de independência física e de independência lógica:

Este é o nível que contém as visões externas e particulares a cada um dos utilizadores do sistema. Cada uma delas corresponde a um subconjunto específico das definições presentes no nível conceptual, pelo que se podem considerar pequenas Bases de Dados a que cada utilizador tem acesso, consoante a sua autorização.

- Independência física. Por questões de afinação do desempenho ou outras, será possível alterar aspetos relativos ao esquema interno da base de dados (por exemplo, substituição das estruturas de armazenamento ou organização dos ficheiros), sem afetar o esquema conceptual. Ou seja, alterações no nível interno não se repercutem no nível conceptual.
- Independência lógica. Será possível, na generalidade dos casos, alterar o esquema conceptual sem ter que alterar também o esquema externo. Ou seja, alterações no nível conceptual não interferem, de forma obrigatória, com as "vistas" estabelecidas no nível externo.'

O SGBD tem a responsabilidade de estabelecer a interface entre os três níveis da arquitetura fazendo os mapeamentos necessários. Para esse efeito, a informação relativa a cada nível e aos mapeamentos entre as suas estruturas está armazenada no dicionário de dados.

A associação dos efeitos de independência física e independência lógica desta arquitetura dá origem a uma das características mais importantes dos sistemas de bases de dados - a independência programas/dados. Na generalidade, alterações que envolvam a estrutura dos dados, ou a sua implementação física, não obrigam a alterações no nível aplicacional.

### 1.5.2 Requisitos Fundamentais de um SGBD

De seguida, analisam-se, com algum pormenor, algumas das tarefas mais importantes da gestão de uma base de dados, como é o caso da segurança e da integridade dos dados, do controlo da concorrência nos acessos e da recuperação/tolerância a falhas. O SGBD, sendo o mecanismo central coordenador de todo o sistema, tem a responsabilidade de assegurar que estes aspetos estejam permanentemente garantidos, quer atuando diretamente, quer disponibilizando meios que os permitam assegurar.

#### Funções genéricas dos SGBD

- Gestão de objetos (criação, alteração, remoção, consulta)
- Tratamento da informação (ordenação, relatórios, etc., ...)
- Linguagem de comandos para execução dos pontos anteriores
- (LINGUAGEM DE INTERROGAÇÃO RELACIONAL - Structured Query Language)
- Ligação a linguagens de programação de alto nível.

## Segurança

O objetivo das medidas de segurança dos sistemas de bases de dados é proteger os dados armazenados de acessos não autorizados, garantindo que apenas os utilizadores autorizados acedem ao sistema, de acordo com os seus privilégios.

A segurança pode assumir, basicamente, duas perspetivas:

- Segurança física. Este tipo de segurança, mais comum no passado, implica que o sistema esteja, fisicamente, fora do alcance de pessoas não autorizadas. Atualmente, dada a dispersão de meios informáticos, e de infraestruturas de comunicações que os interligam, os pontos de acesso aos sistemas informáticos encontram-se de tal forma dispersos que invalidam qualquer tentativa de impedir o acesso físico.
- Segurança lógica. Da mesma forma que alguns sistemas operativos, nomeadamente os multiutilizadores, protegem os recursos do sistema de acessos não autorizados através de mecanismos lógicos de controlo de acesso (*usernames*, *passwords*, etc.), também os SGBDs podem estar preparados com estes tipos de mecanismos. Contudo, no contexto particular dos sistemas de bases de dados, este tipo de controlo de acessos é demasiado limitado. O controlo dos acessos tem que ser mais sofisticado. É necessário ter meios de definir, não só quem tem acesso, mas também a quê e como lhe pode aceder.

Além de impedir pessoas não autorizadas de aceder à base de dados, é ainda necessário definir, em relação aos utilizadores autorizados, os limites das suas autorizações. Ou seja, a que é que têm acesso e o que é que podem fazer. A isso chama-se definir o perfil de um utilizador.

Genericamente, o perfil de um utilizador define-se à custa de dois tipos de restrições. A que partes da base de dados pode aceder (especificado através de "vistas") e de que forma lhes pode aceder (especificado através de regras de autorização). Naturalmente, o resultado da definição do perfil de cada utilizador será armazenado no dicionário de dados e consultado pelo SGBD para validar toda a interação desse utilizador com a base de dados.

Relativamente às "vistas", como já foi referido atrás, estas são definidas no nível externo da arquitetura ANSI/SPARC como forma de personalizar a base de dados, especificando um subconjunto do modelo de dados global, adequado às necessidades de cada utilizador ou aplicação particular. Na realidade, o objetivo da definição de "vistas" não é somente facilitar a interação de cada utilizador com a base de dados. As "vistas", ao impedirem que os utilizadores acedam a outras partes da base de dados, cumprem também objetivos de segurança.

Em relação às regras de autorização, estas permitem especificar o conjunto de operações que cada utilizador está autorizado a executar sobre a sua "vista". Em SGBDs mais evoluídos é possível, não só especificar quais as operações permitidas, mas também em que condições são permitidas.

Outra das funções das medidas de segurança será também dissuadir utilizadores autorizados de tentarem aceder a zonas da base de dados que lhes estão proibidas ou efetuar operações para as quais não estão autorizados. Para isso, cada utilizador identifica-se perante o sistema e todas as operações realizadas durante uma sessão poderão ser registadas.

Uma outra medida de segurança, mais radical, é usada em situações em que a confidencialidade dos dados é de tal forma importante que há necessidade de os manter secretos. Para estes casos, a solução poderá passar por encriptar os dados. Ou seja, o SGBD além de implementar as outras medidas de segurança ainda poderá fornecer mecanismos



para codificar/descodificar os dados, disponibilizando um nível mais elevado de segurança. O preço a pagar é a carga adicional (codificações e descodificações) a suportar pelo sistema.

Um aspeto a ter em conta em questões de segurança é a definição da sua resolução ou granularidade, ou seja, qual a unidade base a que se deve controlar os acessos. "Como é evidente, quanto maior for a granularidade maior será o grau de precisão no controlo de acessos, mas maior será também a sobrecarga colocada sobre o sistema."

Como a história tem vindo a demonstrar, é impossível conseguir uma proteção absoluta contra acessos não autorizados. Contudo, os meios de dissuasão utilizados podem ser suficientemente capazes de deter, senão todas, pelo menos a maioria das tentativas de acesso fraudulento. A ideia é estabelecer um nível de dificuldade adequado ao valor confidencial dos dados a proteger de tal forma que o risco e os custos das tentativas de acessos não autorizados sejam superiores aos benefícios potenciais desses acessos.

## **Integridade**

Por definição, uma base de dados está num estado de integridade se contém apenas dados válidos, isto é, que não contradizem a realidade que estão a representar, antes a refletem corretamente.

Ao contrário das medidas de segurança que se preocupam, fundamentalmente, em proteger a base de dados de acessos não autorizados, a manutenção da integridade pressupõe proteger a base de dados de acessos menos válidos por parte dos utilizadores autorizados, impedindo-os de executar operações que ponham em risco a correção dos dados armazenados. Trata-se, portanto, de um conjunto de medidas de segurança muito especial que, atuando dentro do perfil de autorizações de cada utilizador, pretende evitar que estes, acidentalmente, executem operações que conduzam a base de dados para estados não válidos, contribuindo assim para uma maior robustez do sistema.

Durante a sua existência, uma base de dados vai evoluindo ao longo de estados ditos de integridade. Por definição, cada transação que envolva modificação do conteúdo da base de dados (inserção, alteração ou remoção de dados), e seja finalizada com sucesso, faz avançar a base de dados para um novo estado (Figura 6). Interessa que esse seja um estado de integridade.

Como é evidente, apenas as operações de atualização da base de dados podem pôr em causa a sua integridade. As operações de simples consulta, dado que não originam mudança de estado, não põem em risco a integridade dos dados armazenados.

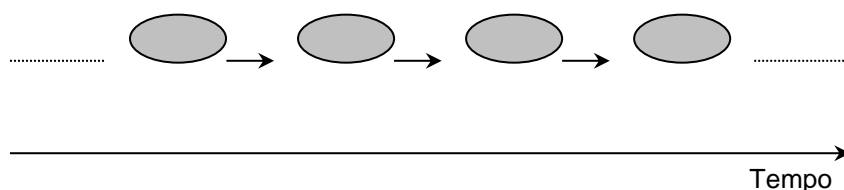


Figura 6 - Sucessivos estados da evolução de uma base de dados

Para que a integridade da base de dados possa ser garantida, as atualizações são governadas por um conjunto de regras, designadas restrições de integridade, que definem o que é válido e o que não é válido. Só para dar alguns exemplos, avulsos, de restrições de integridade:

- "Um funcionário não pode pertencer a mais que um departamento".
- "Um estudante só pode inscrever-se em disciplinas do seu curso".
- "O preço de venda de um produto deverá ser superior ao seu custo".
- "O peso de cada lote deverá ser maior ou igual que 100Kg".

- "A referência de cada produto deve ser única".

Genericamente, as restrições de integridade podem ser classificadas em dois grupos:

- Implícitas. São as restrições de integridade próprias de cada modelo de base de dados. Impedem as atualizações que ponham em causa o funcionamento do próprio modelo.
- Explícitas. Também chamadas restrições de integridade semântica. São as restrições próprias da realidade modelada pela base de dados, independentes do modelo de base de dados particular que a vai suportar.

Relativamente às restrições de integridade explícitas, é vulgar classificá-las em dois tipos:

- a. Restrições estáticas. Estabelecem as condições de validade para os estados. Isto é, são condições que se devem verificar em cada estado para que este possa ser considerado um estado de integridade. Por exemplo, "o salário de um funcionário deve ser superior ou igual ao salário mínimo nacional".
- b. Restrições dinâmicas. Definem quais as transições de estado permitidas, impedindo que certas transições inválidas ocorram. Por exemplo, "o salário de um funcionário nunca poderá decrescer, só aumentar".

Estes dois tipos de restrições de integridade complementam-se, já que nenhuma delas, por si só, basta para garantir a integridade de uma base de dados.

Para dar um exemplo concreto da atuação complementar destes dois tipos de restrição de integridade, considere-se o sistema de base de dados que suporta um universo constituído pelos seguintes dados de funcionários:

- Cod\_Func
- Nome
- Salário

Supondo que este universo, governado pelas restrições de integridade atrás descritas, é constituído por um único funcionário, e que salário mínimo é 100, considere-se a situação representada na Figura 7, correspondente a uma atualização do salário do funcionário.

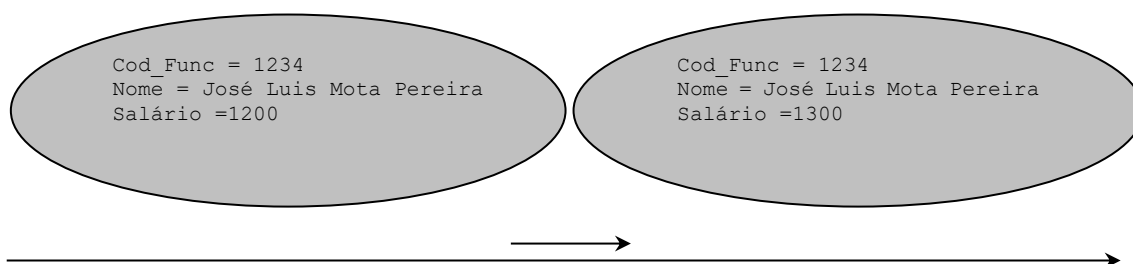


Figura 7 - Mudança de estado sucedida (atualização válida de um campo)

O estado final da base de dados satisfaz a restrição de integridade estática (o novo salário é superior ao salário mínimo nacional). De igual modo, a restrição de integridade dinâmica foi também satisfeita (o novo salário é superior ao salário anterior) pelo que o novo estado da base de dados é um estado de integridade.

Agora, considere-se a situação representada na Figura 8, correspondente a uma atualização ao salário do funcionário em que a integridade da base de dados é violada.

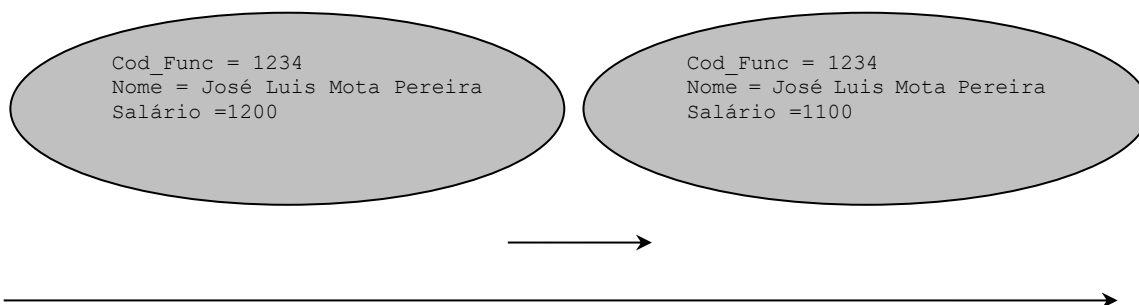


Figura 8 - Mudança de estado não sucedida (tentativa de atualização inválida)

Neste caso, apesar de a restrição de integridade estática ser satisfeita no novo estado (o novo salário é superior ao salário mínimo nacional), a transição levaria a base de dados para um estado de não-integridade pois uma das restrições da realidade (universo) modelada estaria a ser violada (o salário do funcionário estaria a decrescer). A existência da restrição de integridade dinâmica impede que esta transição se faça, mantendo a base de dados num estado de integridade.

De notar que o levantamento das restrições de integridade presentes num dado universo é feito na fase de modelação de dados sendo parte integrante do modelo conceptual. Assim sendo, da mesma forma que a estrutura lógica dos dados vai ser implementada no nível conceptual da arquitetura ANSI/SPARC de bases de dados, o mesmo deveria acontecer com as restrições de integridade. Nesta perspetiva, as restrições de integridade, da mesma forma que os dados, seriam mantidas sob a coordenação do SGBD e disponibilizadas, de forma partilhada, ao nível aplicacional.

Além das vantagens evidentes em manter as restrições de integridade sob o controlo central do SGBD, libertar-se-ia consideravelmente o nível aplicacional contribuindo para uma maior produtividade. A definição das restrições de integridade pode representar cerca de **90% das especificações de uma base de dados típica**.

Lamentavelmente, a realidade é bem diferente. A generalidade dos sistemas atuais de bases de dados são pobres em termos de tratamento da integridade, por vezes mesmo ao nível das restrições de integridade implícitas. Relativamente às restrições de integridade explícitas, a grande maioria dos SGBDs existentes ou não as suporta ou fornece um suporte algo limitado, de tal forma que grande parte desse tratamento é, atualmente, realizado no nível aplicacional com todos os problemas que isso acarreta.

Na realidade, existem já vários SGBDs que dispõem de algumas facilidades, como *stored procedures* e *triggers* (código armazenado na base de dados que pode ser ativado para controlar o acesso à informação), com as quais é possível traduzir essas restrições de integridade. Contudo, trata-se ainda de soluções algo pobres. Por um lado, ao contrário das restrições de integridade, que são construções eminentemente declarativas, essas soluções são, geralmente, de tipo procedimental. Isto é, implicam o desenvolvimento do código necessário para as implementar. Por outro lado, esse código é completamente dependente do SGBD utilizado.

Seria muito mais conveniente que as restrições de integridade pudessem ser parte integrante do esquema conceptual da base de dados. Traduzidas num SGBD da mesma forma que este (ou seja, declarativamente) e, como este, armazenadas no dicionário de dados.

## Controlo da Concorrência

Um dos pressupostos fundamentais dos sistemas de bases de dados é a partilha dos dados armazenados pelo nível aplicacional. O controlo da concorrência relaciona-se com a coordenação dessa partilha por várias aplicações e utilizadores. Trata-se, portanto, de um

problema específico dos sistemas de bases de dados multiutilizador, e a ideia principal é garantir que cada utilizador ou aplicação interage com a base de dados como se fosse o único a utilizar os seus serviços.

A unidade base do controlo da concorrência é a transação. Por definição, uma transação consiste num conjunto de ações originadas por um utilizador ou aplicação que, como um todo, acedem a uma base de dados para consultar ou modificar o seu conteúdo. Trata-se de uma unidade de trabalho que só pode ser executada na totalidade, isto é, ou todas as ações que a constituem são executadas ou nenhuma é executada.

Num sistema de bases de dados multiutilizador, as transações podem ser executadas, basicamente, de duas formas:

- Execução em série. As várias transações submetidas ao sistema são executadas sequencialmente, só se iniciando uma quando a anterior tiver finalizado. Desta forma, a concorrência, pelo menos a nível dos acessos à base de dados, pura e simplesmente não existe, obtendo-se como resultado um nível baixo de utilização do sistema.
- Execução concorrente. Dado que a execução de uma transação é constituída, entre outras coisas, por vários acessos de leitura e escrita à base de dados, poder-se-á pensar em executar as várias transações concorrentemente, combinando e intercalando, quando possível, as suas operações de leitura e escrita na base de dados, no sentido de maximizar a utilização do sistema.

Supondo quatro transações T1, T2, T3 e T4 submetidas ao sistema, por esta ordem, segue-se uma comparação esquemática da sua execução segundo as duas abordagens referidas, sendo

$T_s$  - Tempo total de execução em série.

$T_c$  - Tempo total de execução em concorrência.

$T_a$  - Instante de tempo intermédio para comparação.

A execução série das transações está representada na Figura 9.

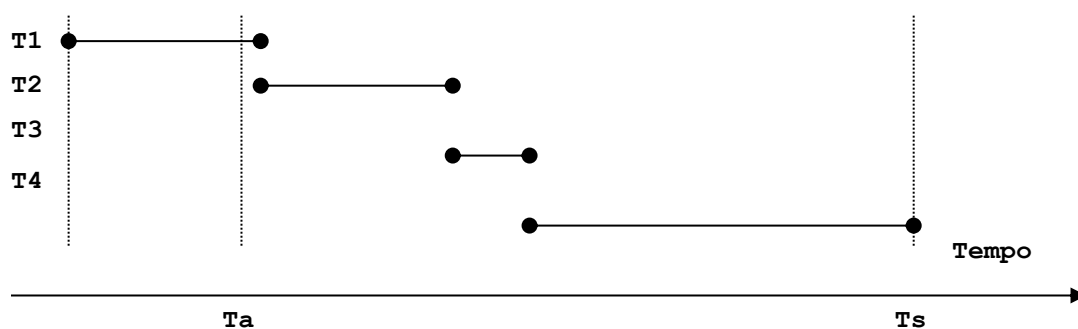
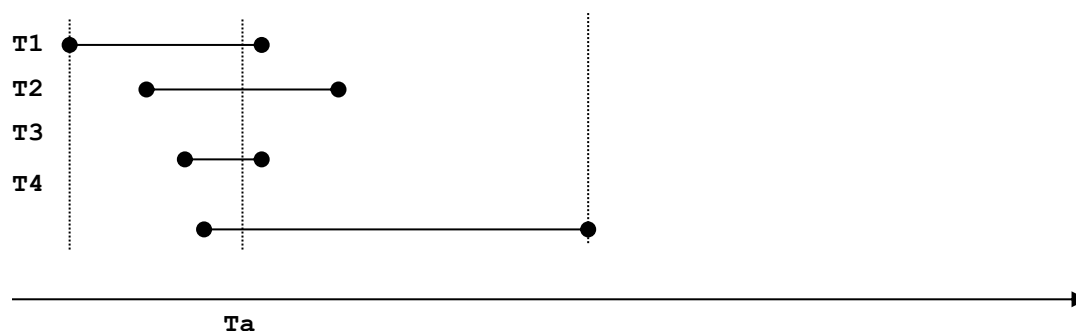


Figura 9 - Execução série de transações

Uma provável execução concorrente das mesmas transações é a que está representada na Figura 10.

De notar que a abordagem pela execução concorrente de transações, dada a melhor utilização que faz dos recursos do sistema, poderá proporcionar melhores desempenhos. De facto, para a generalidade dos casos,  $T_c$  será sempre inferior a  $T_s$ .



Apesar dos ganhos obtidos com a execução das transações em concorrência, é necessário garantir que o resultado final da execução concorrente do conjunto das transações seja o mesmo que se as transações fossem executadas em série. Ou, por outras palavras, é necessário assegurar que as transações concorrentes conduzem a base de dados entre sucessivos estados de integridade.

Quando duas ou mais transações são executadas concorrentemente, em algumas situações, podem surgir problemas decorrentes de interferências entre si, com consequências ao nível da integridade da base de dados.

Se todas as aplicações e utilizadores estão apenas a realizar operações de consulta, os problemas da concorrência não se põem. Estes só acontecem quando duas ou mais transações acedem aos mesmos dados envolvendo, pelo menos uma delas, a execução de modificações.

Considere-se um exemplo típico em que duas transações (T1 e T2) atuam sobre uma mesma conta bancária para debitar o seu saldo.

| T1                  | T2                  |
|---------------------|---------------------|
| ler Saldo           | ler Saldo           |
| Saldo = Saldo - 100 | Saldo = Saldo - 400 |
| escrever Saldo      | escrever Saldo      |

Supondo que o Saldo inicial da conta é 1000, se as duas transações se efetuarem concorrentemente, podem surgir três resultados completamente diferentes, apesar de só um ser correto:

| Tempos | Operações           | Saldo             |
|--------|---------------------|-------------------|
| t1     | T1 (ler Saldo)      | 1000              |
| t2     | T1 (escrever Saldo) | 900               |
| t3     | T2 (ler Saldo)      | 900               |
| t4     | T2 (escrever Saldo) | 500 (saldo final) |

O quadro anterior, correspondente à execução série das duas transações, dá como resultado final um valor de Saldo = 500 (o valor correto!).

Duas outras sequências que, espontaneamente, poderão surgir são as seguintes:

| Tempos | Operações           | Saldo             |
|--------|---------------------|-------------------|
| t1     | T1 (ler Saldo)      | 1000              |
| t2     | T2 (ler Saldo)      | 1000              |
| t3     | T1 (escrever Saldo) | 900               |
| t4     | T2 (escrever Saldo) | 600 (saldo final) |

| Tempos | Operações           | Saldo             |
|--------|---------------------|-------------------|
| t1     | T1 (ler Saldo)      | 1000              |
| t2     | T2 (ler Saldo)      | 1000              |
| t3     | T2 (escrever Saldo) | 600               |
| t4     | T1 (escrever Saldo) | 900 (saldo final) |

Conclusão, a execução das duas transações concorrentemente, sem qualquer coordenação, pode originar resultados incorretos.

Nestes dois últimos casos, o que se passou foi que se perdeu uma das atualizações do saldo, prevalecendo apenas a última atualização. Esta é apenas uma das formas típicas de transações concorrentes interferirem entre si, conhecida por *lost update*.

Uma outra situação que reflete bem a interferência entre transações concorrentes, conhecida por *dirty read*, é a seguinte. Uma transação T1 atualiza determinado elemento de dados; antes de T1 terminar, uma outra transação T2 lê esse elemento de dados; caso a transação T1 aborte (e, portanto, todas as suas atualizações na base de dados sejam desfeitas), a transação T2 leu e processou um valor que, como não foi *committed*, é como se nem sequer tivesse existido.

Pode-se assim concluir que, no caso genérico dos sistemas de bases de dados multiutilizador, as restrições de integridade não são suficientes para garantir a integridade da base de dados. A execução concorrente de transações pode originar interferências entre si, de tal forma graves, que podem pôr em risco a integridade da base de dados. Isso, apesar de cada transação envolvida, quando executada separadamente, poder produzir resultados corretos.

O objetivo é maximizar o nível de concorrência entre as transações sem que estas interfiram entre si, ou seja, produzindo os mesmos resultados que produziriam se tivessem sido executadas em série.

Basicamente, é necessário definir procedimentos e/ou utilizar mecanismos que impeçam duas transações concorrentes de acederem, simultaneamente, ao mesmo conjunto de dados, envolvendo atualização dos mesmos.

## Recuperação/Tolerância a Falhas

Dada a importância dos dados armazenados para a organização, e a necessidade de os manter constantemente operacionais, torna-se indispensável garantir que a base de dados esteja permanentemente disponível num estado de integridade ou, se indisponível, que durante uma fração de tempo reduzida se faça a sua reposição para um estado de integridade.

Naturalmente, um sistema de bases de dados, como qualquer outro sistema, está sujeito à ocorrência de falhas que podem pôr em causa a integridade da base de dados. Se se pretende que essa integridade nunca seja abalada, o sistema de bases de dados deve estar preparado para ultrapassar qualquer falha, reagindo diretamente ou fornecendo meios que permitam

atuar, no sentido de fazer a reposição da base de dados para um estado de integridade anterior a essa falha. Esse estado passado deverá ser o mais próximo possível do momento em que ocorreu a falha, de tal forma que as perdas, se existirem, sejam minimizadas.

A recuperação/tolerância a falhas é a atividade que tem por objetivo o restaurar da base de dados, após a ocorrência de uma qualquer falha, para um estado de integridade garantido.

As falhas a que um sistema de base de dados está sujeito são de várias ordens e gravidades. O SGBD terá que dispor de mecanismos que suportem a recuperação da base de dados, desde situações tão graves como a perda de toda a base de dados, até situações menos drásticas, mas igualmente importantes, em que, por exemplo, por falha de uma transação individual, a base de dados atingiu um estado de não integridade, sendo necessário recuar para o estado imediatamente anterior a essa transação, desfazendo todas as atualizações que a transação efetuou.

Os mecanismos de recuperação dos sistemas de bases de dados baseiam-se, fundamentalmente, na utilização de formas de redundância que, na prática, quase duplicam a própria base de dados.

Existem, basicamente, dois tipos de mecanismos de recuperação, com finalidades distintas, mas que se completam:

- **Backups.** São cópias de segurança, executadas periodicamente, abrangendo toda a base de dados. Constituem o ponto de partida para a reconstrução da base de dados no caso de falhas mais catastróficas. A sua principal desvantagem é que permite uma recuperação algo limitada, uma vez que apenas reflete o estado da base de dados num momento passado, o que significa que na sua reposição, a base de dados irá perder todos os estados seguintes à realização desse *backup*. Esta desvantagem pode ser combatida aumentando a periodicidade dos *backups*, contudo, como é evidente, o processo de backup de toda a base de dados é um processo pesado, consumidor de recursos e que obriga, normalmente, à paralisação de todo o sistema. Desta forma, a sua periodicidade dependerá, significativamente, do valor dos dados, do seu volume e da frequência com que são alterados.
- **Transaction logging.** Este é o mecanismo de recuperação que, entre outras coisas, vem eliminar a principal fraqueza apresentada pelos *backups*. De facto, se a reconstrução da base de dados, a partir do seu *backup*, for completada com a reposição dos estados seguintes a esse backup (permitida pelo *transaction logging*) então será possível reconstituir a base de dados, praticamente, até ao momento em que ocorreu a falha. Para que isso seja possível, torna-se necessário que o sistema mantenha o registo das operações efetuadas à base de dados por todas as transações posteriores ao último backup. Só dessa forma será possível refazer mais tarde, se necessário, toda a evolução da base de dados. Para esse efeito, o backup da base de dados é completado com um ficheiro especial designado *transaction log*, contendo os dados necessários ao refazer de transações passadas.

Enquanto o *backup* é um mecanismo de recuperação que permite atuar ao nível de toda a base de dados, repondo-a por completo num estado passado, o *transaction log* permite refazer as transações ocorridas desde esse último *backup* (processo conhecido por *rollforward*). Para que o refazer de uma transação seja possível guarda-se, no ficheiro *transaction log*, além da identificação da própria transação, aquilo que se designa por *after-images*, isto é, cópias dos novos valores atualizados por essa transação. Existem assim os meios para efetuar a recuperação de uma base de dados partindo de um estado de integridade passado até um momento no futuro.

Contudo, nem todos os tipos de falhas se resolvem desta maneira. Existe um tipo de falhas cuja solução implica, não o refazer da base de dados, mas sim o desfazer. Dadas as

características de atomicidade das transações, a integridade da base de dados, como já foi discutido no ponto anterior, apenas é garantida antes de cada transação se ter iniciado e após ter terminado (*committed*). Durante a execução da transação a base de dados poderá estar num estado de *não-integridade*, por essa razão, se uma transação não terminar normalmente (devido a uma falha) é necessário desfazer quaisquer dos seus efeitos intermédios sobre a base de dados, recuando para o estado anterior a essa transação. A este processo designa-se *rollback*. Para que este processo seja possível, o ficheiro *transaction log* terá também que armazenar os valores anteriores a quaisquer atualizações executadas por uma transação (*before-images*). A cada atualização corresponderá então a criação de *before-images* e *after-images* no ficheiro *transaction log*.

A conjugação destes dois mecanismos (*backups* mais *transaction logs*) permite implementar duas das características fundamentais das transações. Por um lado, permite desfazer os efeitos de uma transação não sucedida (atomicidade), por outro lado, em caso de perda da base de dados permite refazer os efeitos de todas as transações bem-sucedidas (persistência).

Nos ficheiros *transaction log*, assim como nos ficheiros que suportam a base de dados, as atualizações são efetuadas em zonas da memória central (volátil) que representam *buffers* desses ficheiros em disco. Por sua vez, estes *buffers* são periodicamente escritos em memória não-volátil, independentemente entre si, em função de critérios que, como se verá mais à frente, visam minimizar o número de acessos a disco (gestão de *buffers*). Como é evidente, em caso de falha do sistema, o conteúdo desses *buffers* é perdido, podendo daí resultar incoerências (dados no *transaction log* referindo atualizações à base de dados que na verdade se perderam, atualizações na base de dados que não estão registadas no *transaction log*), ou mesmo perdas efetivas de dados (atualizações que não figuram nem no *transaction log* nem na base de dados).

Evidentemente, para que os efeitos de uma transação sejam persistentes, na altura do seu *commit* todas as suas atualizações devem ser salvaguardadas em memória não-volátil, quer no ficheiro *transaction log*, quer diretamente nos ficheiros da base de dados, quer sob as duas formas. Contudo, por razões que se tornarão mais claras quando se analisar a gestão de *buffers*, não é razoável uma abordagem em que no *commit* de cada transação todas as suas atualizações tenham que ser escritas de imediato na base de dados em disco, dada a sobrecarga em termos de I/O. A solução passa pelo ficheiro *transaction log*.

O *transaction log* guarda informação suficiente acerca de cada transação a ponto de permitir ao sistema reexecutá-la ou, pelo contrário, desfazer os seus efeitos como se não tivesse existido. Desde que o *transaction log* seja escrito em memória não-volátil na altura do *commit* de cada transação, não há necessidade de escrever imediatamente as atualizações à base de dados em memória não-volátil. Em caso de falha o *transaction log* pode ser utilizado para refazer as transações cujos efeitos ainda não se tinham traduzido na base de dados em memória não-volátil.

A esta abordagem designa-se *write-ahead logging*, ou seja, no *commit* de cada transação os seus efeitos sobre o *transaction log* são escritos em memória não-volátil antes de a base de dados em memória central ser atualizada. Desta forma, na eventualidade de uma falha do sistema, apenas as transações ativas nesse momento serão perdidas, todas aquelas que já fizeram o *commit* estão, garantidamente, salvaguardadas.

Conforme a natureza da falha, assim o processo de recuperação pode ser mais ou menos complexo, mais ou menos abrangente. Existem, para os diferentes tipos de falhas que podem ocorrer durante a operação normal de um sistema de bases de dados, vários métodos de recuperação:

- Falha de Disco. Corresponde à situação em que o(s) disco(s), onde se encontra armazenada a base de dados, ficam inutilizado(s). Esta é a falha mais grave, uma vez



que obriga à reconstrução de toda a base de dados. Para isso, é necessário proceder ao carregamento do último *backup* e à sua atualização utilizando as *after-images* do ficheiro *transaction log*, processo conhecido por *rollforward*.

- Falha de sistema. Devido a problemas de hardware ou software (erros do sistema operativo ou do próprio SGBD) podem ocorrer falhas internas de sistema. Nestas situações, por uma questão de segurança, considera-se que o conteúdo de memória central está corrompido, ou seja, a base de dados pode estar disponível, mas não é garantida a sua validade. Torna-se então necessário regressar a um estado anterior válido. Para isso, através de um processo de *rollback*, recuperam-se as *before-images* das transações para, posteriormente, através das suas *after-images* repor a base de dados num estado válido anterior à falha.
- Falha de Transação. A falha mais inofensiva em termos de recuperação é a falha de transação individual. Para este caso, basta recorrer ao ficheiro *transaction log* e recuperar as *before-images* da dita transação, desfazendo assim os seus efeitos.

### 1.5.3 Utilizadores de Sistemas de Bases de Dados

Relativamente ao tipo de utilizadores que interagem com os sistemas de bases de dados, podem-se distinguir, neste contexto, duas classes de indivíduos com interesses e posturas diferentes perante o sistema. O ABD (Administrador da Base de Dados) e os utilizadores propriamente ditos:

- Administrador da base de dados - é o responsável máximo pelo bom funcionamento de todo o sistema. Tem como principais responsabilidades, a especificação do esquema conceptual da base de dados e a sua manutenção. Sendo responsável pelo funcionamento do sistema, define procedimentos de *backup* e recuperação, monitoriza a sua utilização (distribuição de cargas de utilização, etc.), e mantém o seu bom o seu desempenho. Controla a segurança, dando e retirando acessos aos utilizadores. Dependendo da complexidade do sistema, a função poderá ser desempenhada por uma ou mais pessoas.
- Utilizadores - podem-se distinguir, neste contexto, basicamente, dois tipos de utilizadores:
  - Utilizadores finais - são a razão de existir da base de dados. Acedem ao sistema para consultar, alterar, adicionar ou remover dados (debaixo das respetivas condições de acesso), não poderão nunca é alterar a estrutura da base de dados (esquema conceptual). Dentro deste grupo surgem os menos sofisticados, que se limitam a correr aplicações sobre o sistema, e os que, casualmente, nos casos em que o próprio SGBD dispõe dessa facilidade, utilizam linguagens de interrogação para obter, interactivamente, respostas específicas do sistema - as chamadas questões *ad hoc*.
  - Programadores - utilizando linguagens de alto-nível e tendo por base "vistas" sobre o esquema conceptual da base de dados, desenvolvem as aplicações que os utilizadores finais irão depois utilizar.

Um outro interveniente importante, apesar de não diretamente relacionado com os sistemas de bases de dados, é o administrador de dados da organização.

Como já foi referido atrás, é reconhecido que, nos dias que correm, a informação é um dos mais valiosos recursos das organizações e, como tal, deve ser bem gerida.

Assim sendo, justifica-se a função de administração de dados cujo objetivo é descobrir as necessidades da organização em termos de dados. O administrador de dados é quem executa essa função. Basicamente, decide quais são os dados relevantes para a organização, isto é, que suportam todos os seus requisitos de informação, e padroniza as suas definições e tipos.

Segundo Date ([Date 1995]), o administrador de dados é um gestor e não um técnico ("apesar de razoavelmente informado das facilidades postas à disposição pelas tecnologias da informação"). O técnico responsável por implementar as decisões do administrador de dados é o ABD, cujo perfil deverá corresponder a um profissional de informática.

## 2. Modelo Relacional

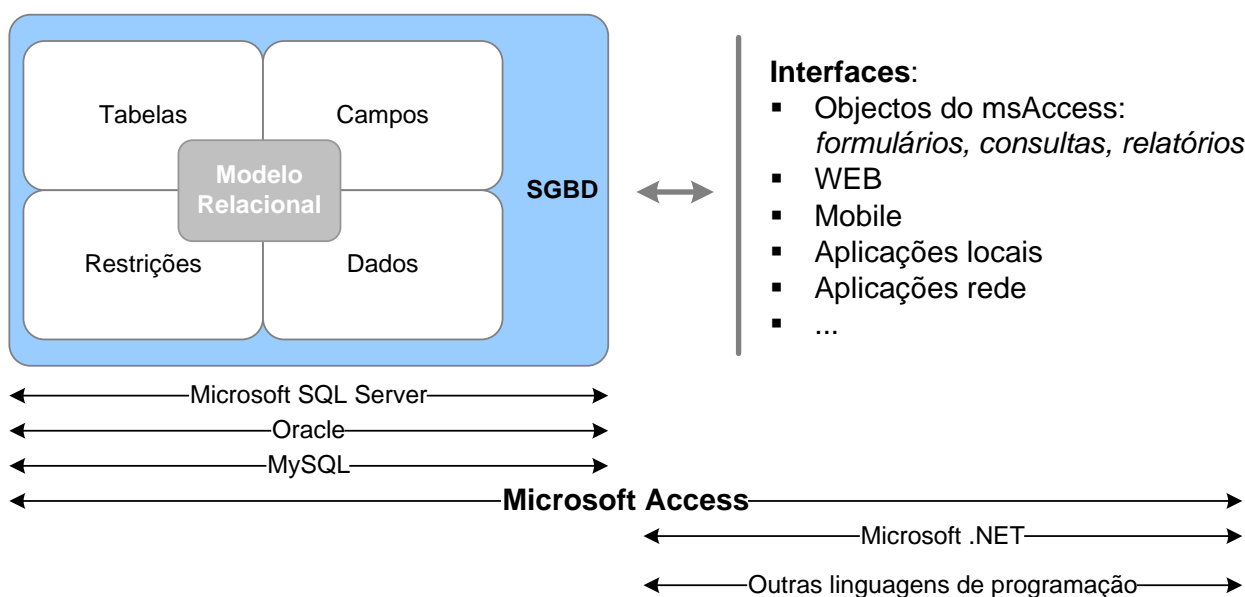


Figura 10 - Modelo relacional

Contrariamente aos modelos hierárquico e rede, o modelo relacional não evoluiu a partir das técnicas de processamento de ficheiros. Antes pelo contrário, foi fruto de um desenvolvimento teórico, tendo por base a teoria dos conjuntos.

1970, E. F. Codd, na altura investigador da IBM Corp., publicou um artigo com os fundamentos teóricos do modelo relacional. Entre publicação do artigo e o aparecimento de um primeiro sistema baseado no modelo relacional passaram vários anos. De facto, só em 1979/80 surgiu, disponibilizado pela *Relational Software Inc.* (atualmente designada Oracle Corp.), o primeiro produto com características relacionais - o SGBD Oracle.

Apesar do modelo relacional ser atualmente, considerado como o "melhor" modelo de entre os modelos convencionais (1ª e 2ª gerações), a aceitação dos sistemas de bases de dados relacionais não foi imediata. Enquanto os seus defensores argumentavam que o modelo relacional era muito mais simples e flexível, os seus detratores contrapunham que o desempenho dos sistemas relacionais nunca iriam permitir a sua utilização comercial.

O tempo encarregou-se de provar o contrário. O modelo relacional, após terem sido resolvidos alguns problemas iniciais, nomeadamente de desempenho, implantou-se no mercado de uma forma e a um ritmo nunca vistos, contribuindo decisivamente para a massificação da utilização da tecnologia de bases de dados nas organizações.

Atualmente existem vários fornecedores de tecnologia relacional no mercado. Entre os SGBDs relacionais mais representativos incluem-se o CA-OpenIngres da Computer Associates Inc., o DB/2 da IBM Corp., o Informix-OnLine Dynamic Server da Informix Software Inc., o Microsoft SQL Server da Microsoft Corp., o Oracle Server da Oracle Corp. e o Sybase SQL Server da Sybase Inc.

### 2.1 Conceitos

A estrutura fundamental do modelo relacional é a relação, também designada tabela. Uma relação é uma estrutura bidimensional com um determinado esquema e zero ou mais instâncias. O esquema de uma relação é constituído por um ou mais atributos que traduzem

o tipo de dados a armazenar. A cada instância do esquema de uma relação designa-se tuplo (registo).

Para cada atributo de uma relação define-se o seu domínio. O domínio de um atributo traduz a gama de valores possíveis que esse atributo pode tomar. Por exemplo, para um atributo que pretenda representar "preços de produtos" poderá definir-se como domínio o conjunto dos valores reais positivos.

Ao número de atributos que constituem o esquema de uma relação dá-se o nome de grau da relação. Por sua vez, o número de tuplos de uma relação designa-se por cardinalidade da relação.

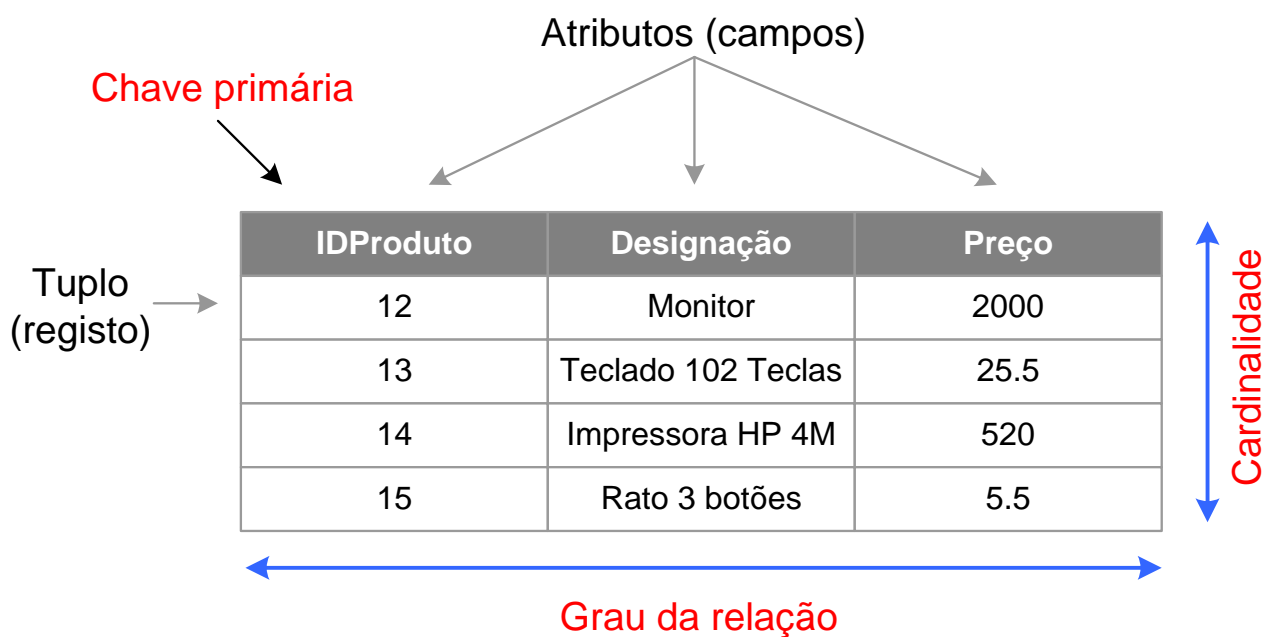


Figura 11 - Exemplo de uma relação

No modelo relacional as relações possuem algumas características que convém identificar:

- O valor de cada atributo num tuplo é atómico. Ou seja, no cruzamento de uma linha (tuplo) com uma coluna (atributo) só é possível encontrar um valor.
- Os atributos de uma relação devem ter identificadores distintos. Apesar de num mesmo esquema de base de dados relacional poderem surgir atributos com identificadores idênticos, estes terão que pertencer a relações distintas (tabelas).
- Os tuplos de uma relação devem ser distintos. Isto é, numa relação não podem existir dois tuplos com os mesmos valores para todos os atributos.
- A ordem dos tuplos numa relação, da mesma forma que a ordem dos atributos no seu esquema, não tem qualquer significado.
- Todos os valores de um dado atributo provêm de um dado domínio. O valor de alguns atributos num tuplo poderão ser desconhecidos ou simplesmente, não existir. Para estes casos existe um valor especial designado null.

Ainda em relação às tabelas ou relações, definem-se dois tipos:

- a. Relações base que constituem o esquema da base de dados, onde estão realmente armazenados os dados.

- b. Relações virtuais ou *views* que não têm existência própria, pois são derivadas a partir das tabelas base com o objetivo de proporcionar "vistas" parciais sobre o esquema da base de dados.

Um outro conceito fundamental no contexto do modelo relacional é o conceito de chave. Podemos distinguir vários tipos:

- Chave candidata*. Atributo ou conjunto de atributos que permitem identificar de forma inequívoca qualquer tuplo dessa relação. O conjunto não pode ser reduzido sem perder essa qualidade.
- Chave primária*. Também chamada chave principal, é selecionada de entre as várias chaves candidatas. A Chave Primária terá que ser:
  - Unívoca: o atributo (ou atributos) da chave primária tem um valor único para qualquer tuplo da relação.
  - Não nula: Não pode haver tuplos da relação que tenham o atributo (ou atributos) da chave primária nulo (sem qualquer valor).
  - Não redundante: Se algum dos atributos que a constituem for retirado os restantes deixam de identificar univocamente o tuplo.

*Exemplo:*

| Nome   | B.I     | N_contribuinte | N_eleitor | Freguesia | Concelho |
|--------|---------|----------------|-----------|-----------|----------|
| Maria  | 1234567 | 123456722      | 2222      | S. Pedro  | Covilhã  |
| Manuel | 3377229 | 234156233      | 3333      | Conceição | Covilhã  |
| Paulo  | 2233337 | 233333567      | 3456      | S. Maria  | Covilhã  |
| Paula  | 2876909 | 222333333      | 6782      | S. Maria  | Pedrogão |

Chaves candidatas: {B.I.}, {N\_Contribuente}, {N\_Eleitor}, {Freguesia, Concelho}  
Chave Primária?

- Superchave*. Associação de um ou mais atributos cujos valores, em conjunto, identificam univocamente cada tuplo. Como é evidente, no limite, a associação de todos os atributos de uma relação constitui uma superchave.
- Chave estrangeira*. Também designada chave importada, trata-se de um conjunto constituído por um ou mais atributos que é chave primária numa outra relação. Subconjunto de atributos que constituem a chave primária de uma outra relação permitindo estabelecer a associação entre tuplos de diferentes relações.

*Exemplos da relação "Fornecimentos"*

| N_fornecimento | N_obra | N_fornecedor | N_material | Qtd_fornecida |
|----------------|--------|--------------|------------|---------------|
| Fr1            | O1     | F1           | M1         | 10 000        |
| Fr2            | O1     | F2           | M3         | 5 000         |
| Fr3            | O3     | F3           | M2         | 500           |
| Fr4            | O3     | F4           | M1         | 1 000         |
| Fr5            | O3     | F2           | M1         | 50 000        |

- N\_fornecimento é chave primária da relação Fornecimentos

- N\_obra é chave estrangeira da relação Fornecimentos porque é chave primária na relação Obras
- N\_fornecedor é chave estrangeira da relação Fornecimentos porque é chave primária na relação Fornecedores
- N\_material é chave estrangeira da relação Fornecimentos porque é chave primária na relação Materiais.

A existência de uma chave estrangeira numa relação prende-se com a necessidade de manter a interligação entre essa relação e a relação onde esse conjunto de atributos é chave principal.

No modelo relacional, a única forma de relacionar dados existentes em diferentes tabelas é através de atributos comuns (chaves importadas) às tabelas que se relacionam. Se tuplos individuais de duas relações têm o mesmo valor no(s) atributo(s) comum(comuns) é porque estão relacionados. Por exemplo, dadas duas relações - **Departamentos** e **Funcionários**, com vários atributos caracterizando cada departamento e respetivos funcionários, a única forma de traduzir que um dado funcionário trabalha num dado departamento é incluir na relação **Funcionários** a identificação do departamento onde esse funcionário trabalha (ou seja, importando para a relação **Funcionários** a chave primária da relação Departamentos). Esta situação encontra-se representada na Figura 13.

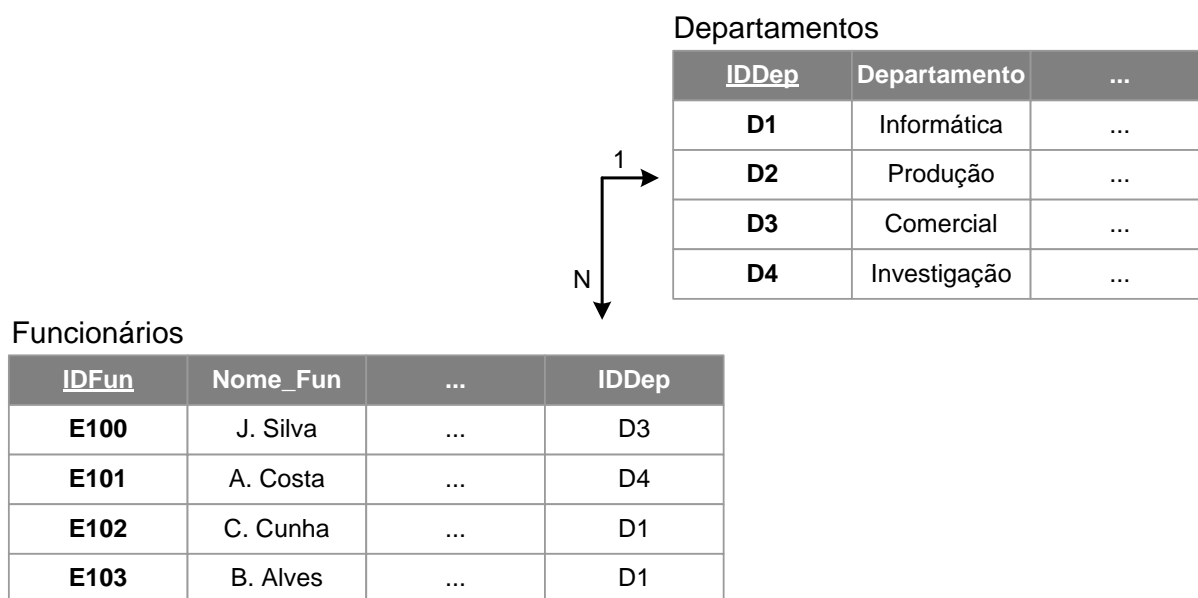




Figura 12 - Relacionamento entre tabelas recorrendo a chaves importadas

Ou seja, **IDDep**, que é chave primária na relação **Departamentos**, é chave estrangeira na relação **Funcionários**. Se este atributo não existisse na relação **Funcionários** não seria possível relacionar as duas tabelas.

Relativamente às restrições de integridade implícitas ao modelo relacional, estas resumem-se ao seguinte:

- *Integridade de domínio*. Esta é a forma mais elementar de restrição de integridade [Korth and Silberschatz 1991]. Existem certas restrições ao domínio dos atributos que poderão ser garantidas pelo SGBD. Por exemplo, gamas de valores permitidos, obediência a padrões, etc.. São regras que se aplicam aos atributos de uma dada tabela, definindo o domínio de cada atributo.

| Table - dbo.tabTrabalhadores  |                         |               |                                     |
|---|-------------------------|---------------|-------------------------------------|
| Summary   |                         |               |                                     |
|   | Column Name             | Data Type     | Allow Nulls                         |
|  | IDTrabalhador           | int           | <input type="checkbox"/>            |
|   | [Nome(s)]               | nvarchar(50)  | <input type="checkbox"/>            |
|   | [Apelido(s)]            | nvarchar(50)  | <input type="checkbox"/>            |
|   | Sexo                    | nvarchar(1)   | <input type="checkbox"/>            |
|   | NºIDCivil               | int           | <input checked="" type="checkbox"/> |
|  | DataNascimento          | datetime      | <input type="checkbox"/>            |
|   | IDNacionalidade         | int           | <input type="checkbox"/>            |
|   | NºIdentificaçãoFiscal   | int           | <input checked="" type="checkbox"/> |
|   | NºSegurançaSocial       | int           | <input checked="" type="checkbox"/> |
|   | NºUtenteSaude           | int           | <input checked="" type="checkbox"/> |
|   | IDCategoriaProfissional | int           | <input type="checkbox"/>            |
|   | Observacoes             | nvarchar(MAX) | <input checked="" type="checkbox"/> |
|   | Foto                    | image         | <input checked="" type="checkbox"/> |
|   |                         |               | <input type="checkbox"/>            |

- *Integridade da entidade.* Em todas as relações existe uma chave primária constituída por um ou mais atributos, cujos valores permitem identificar inequivocamente cada tuplo. Não pode haver nenhum tuplo para o qual o valor dessa chave seja nulo, nem sequer conter partes nulas.

#### Empregados

| <u>Emp#</u> | Nome          | Categoria   | Salário | Dep# | Data_Nasc  |
|-------------|---------------|-------------|---------|------|------------|
| 1           | António Sousa | Programador | 1000    | 5    | 20-03-1980 |
| 2           | Ana Amaral    | Programador | 1000    | 6    | 22-03-1970 |
| 3           | José Costa    | Analista    | 2000    | 7    | 12-04-1964 |
| 2           | Carlos Silva  | Operador    | 500     | 5    | 20-08-1980 |

Ao declararmos um atributo como chave primária da relação o SGBD não deixa que a relação tenha dois tuplos com o mesmo valor nesse atributo

- *Integridade referencial.* O valor de uma chave estrangeira ou é null ou então contém um valor que é chave primária na relação de onde foi importada. Por outras palavras, se numa relação o valor de uma chave importada não é nulo, então esse valor terá que existir na relação onde essa chave é primária.

#### Empregados

| <u>Emp#</u> | Nome          | Categoria   | Salário | Dep# | Data_Nasc  |
|-------------|---------------|-------------|---------|------|------------|
| 1           | António Sousa | Programador | 1000    | 5    | 20-03-1980 |
| 2           | Ana Amaral    | Programador | 1000    | 6    | 22-03-1970 |
| 3           | José Costa    | Analista    | 2000    | 7    | 12-04-1964 |
| 4           | Carlos Silva  | Operador    | 500     | 5    | 20-08-1980 |

## Departamentos

| <u>Dep#</u> | Nome | Local  |
|-------------|------|--------|
| 5           | D1   | Lisboa |
| 6           | D2   | Porto  |
| 7           | D3   | Lisboa |

O atributo Dep# na tabela Empregados é chave estrangeira (ou externa) sendo chave primária na tabela Departamentos. Se, se indica que Dep# é chave estrangeira da relação Empregados então cada valor do Atributo Dep# na tabela Empregados tem obrigatoriamente que existir na tabela Departamentos.

## 2.2 Relações entre tabelas

Um dos objetivos de uma boa estrutura de base de dados consiste em remover redundância de dados (dados duplicados). Para alcançar esse objetivo, deverá dividir os dados em muitas tabelas baseadas em assunto, para que cada facto seja representado apenas uma vez.

- 1 Chave primária
- 2 Linha que representa a relação
- 3 Chave estrangeira

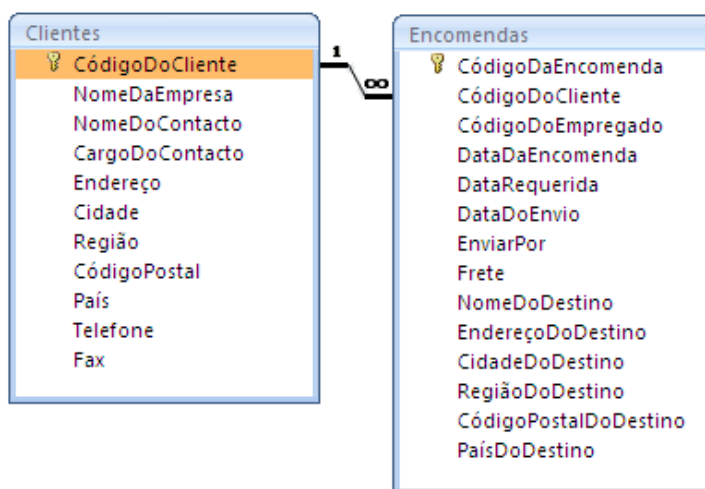


### Tipos de relações de tabela

O modelo relacional suporta três tipos de relações entre tabelas.

- Uma relação **um-para-muitos**

Considere uma base de dados de registo de encomendas que inclua uma tabela Clientes e uma tabela Encomendas. Um cliente pode efetuar qualquer número de encomendas. Por conseguinte, por qualquer cliente representado na tabela Clientes, podem existir muitas encomendas representadas na tabela Encomendas. A relação entre a tabela Clientes e a tabela Encomendas e, por conseguinte, uma relação um-para-muitos.



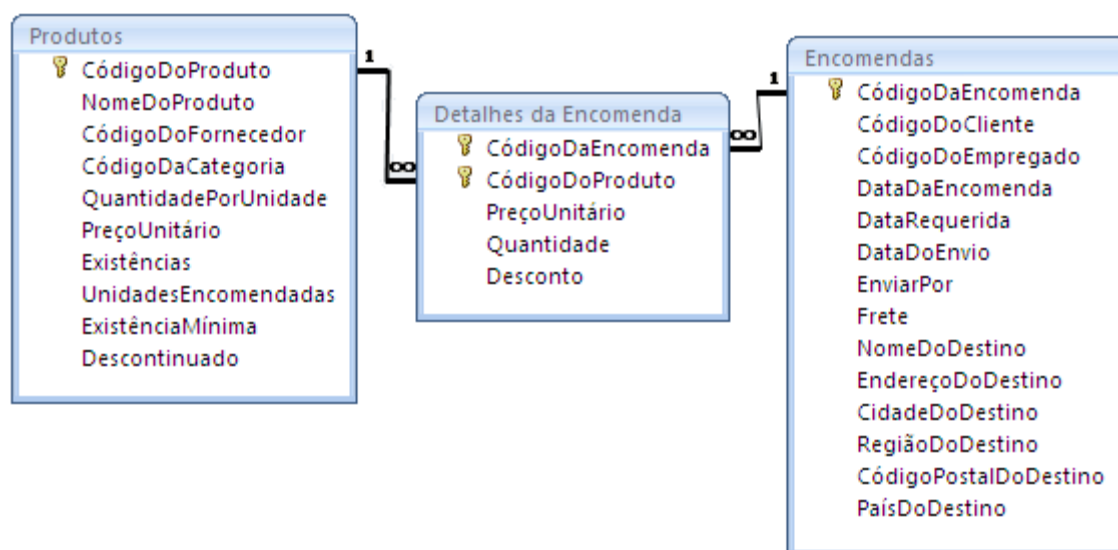
Para representar uma relação um-para-muitos na estrutura da base de dados, adicione a chave primária do lado "um" da relação como campo ou campos adicionais à tabela no lado "muitos" da



relação. Neste caso, por exemplo, deverá adicionar um novo campo (o campo Código da tabela Clientes) à tabela Encomendas e atribuir-lhe o nome CódigoDoCliente..

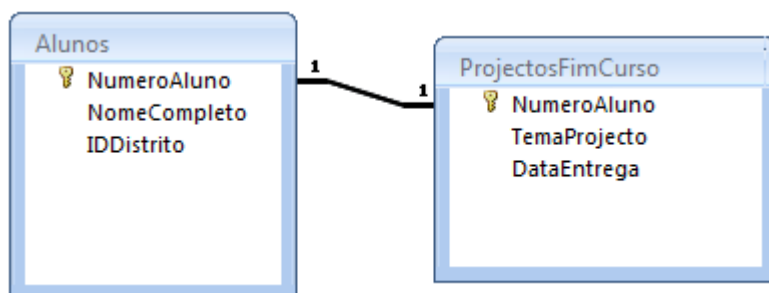
- Uma relação **muitos-para-muitos (indiretamente)**

Considere a relação entre uma tabela *Produtos* e uma tabela *Encomendas*. Uma única encomenda pode incluir mais de um produto. Por outro lado, um único produto pode aparecer em várias encomendas. Como tal, para cada registo na tabela Encomendas, podem existir vários registos na tabela Produtos. Além disso, para cada registo na tabela Produtos, podem existir vários registos na tabela Encomendas. Este tipo de relação é designada por uma relação *muitos-para-muitos*, pois podem existir várias encomendas para qualquer produto e vários produtos para qualquer encomenda. Tenha em atenção que, para detetar relações muitos-para-muitos existentes entre as tabelas, é importante considerar ambos os lados da relação.



Para representar uma relação muitos-para-muitos, terá de criar uma terceira tabela, frequentemente apelidada de *tabela de união/associação*, a qual divide a relação muitos-para-muitos em duas relações um-para-muitos. A chave primária de cada uma das duas tabelas é introduzida na terceira tabela. Como tal, a terceira tabela regista cada ocorrência, ou instância, da relação. Por exemplo, a tabela Encomendas e a tabela Produtos têm uma relação muitos-para-muitos que é definida criando-se duas relações um-para-muitos para a tabela Detalhes da Encomenda. Uma encomenda pode ter muitos produtos e cada produto pode aparecer em muitas encomendas.

- Uma relação **um-para-um**



Numa relação *um-para-um*, cada registo na primeira tabela só pode ter um registo correspondente na segunda tabela, e cada registo na segunda tabela só pode ter um registo correspondente da primeira tabela. Este tipo de relação não é comum porque, na maioria das vezes, as informações relacionadas desta forma são armazenadas na

mesma tabela. Poderá utilizar uma relação um-para-um para dividir uma tabela com muitos campos, para isolar parte de uma tabela por motivos de segurança ou para armazenar informações aplicáveis apenas a um subconjunto da tabela principal. Quando uma relação deste tipo é identificada, ambas as tabelas têm de partilhar um campo em comum.

## Terminologia

| Formal   | Alternativa 1 | Alternativa 2 |
|----------|---------------|---------------|
| Relação  | Tabela        | Ficheiro      |
| Tuplo    | Linha         | Registo       |
| Atributo | Coluna        | Campo         |

## Considere a seguinte relação de dados.

| ID   | Nome      | IDFornecedor | IDModelo | IDCategoria | Peso | Preço | Cor |
|------|-----------|--------------|----------|-------------|------|-------|-----|
| 1001 | Bicicleta | F12          | M1       | AS3         | 12   | 124   | A   |
| 1002 | Roda      | F12          | M1       | AS23        | 5    | 25,5  | P   |
| 1003 | Roda      | F13          | M1       | AS24        | 5    | 35,6  | V   |
| 1004 | Travão    | F134         | M1       | ZX7         | 0.25 | 6.25  | V   |
| 1005 | Travão TD | F134         | M1       | ZX7         | 0.24 | 7.89  | A   |

Qual a cardinalidade da relação?

Qual o grau da relação?

Qual o Atributo ou conjunto de atributos que podem ser chave candidata? (represente os conjuntos entre parêntesis, Ex. {ID, Peso}).

Das chaves candidatas indicadas selecione a chave primária?

Quais dos atributos constituem Chaves Estrangeiras?

## 2.3 Tipos de Dados

O *SQL Server* suporta 28 tipos de dados de sistema e ainda permite que os programadores criem os seus próprios tipos de dados.

Quando se atribui um tipo de dado a um objeto é necessário considerar dois atributos:

- O tipo de dados do objeto
- O tamanho ou comprimentos de armazenamento

|          |                               |           |            |             |
|----------|-------------------------------|-----------|------------|-------------|
| bigint   | binary                        | bit       | char       | cursor      |
| datetime | decimal                       | float     | image      | int         |
| money    | nchar                         | ntext     | numeric    | nvarchar    |
| real     | smalldatetime                 | smallint  | smallmoney | sql_variant |
| table    | text                          | timestamp | tinyint    | varbinary   |
| varchar  | uniqueidentifier <sup>1</sup> | xml       |            |             |

### Dados numéricos (inteiros e quantidades)

Armazenam dados numéricos inteiros sem componente decimal

| Tipo de dados   | Espaço de armazenamento | Valor mínimo e máximo                      |
|-----------------|-------------------------|--|
| <b>bit</b>      | 1/8 byte                | 0 ou 1                                     |
| <b>tinyint</b>  | 1 byte                  | 0 a 255                                    |
| <b>smallint</b> | 2 bytes                 | -32768 a 32767                             |
| <b>int</b>      | 4 bytes                 | -2147483648 a 2147483647                   |
| <b>bigint</b>   | 8 bytes                 | -9223372036854775808 a 9223372036854775807 |

### Dados numéricos (números precisos e dados de contabilidade)

Armazenam dados numéricos com partes decimais. Para o registo destes dados o *SQL Server* oferece quatro tipos de dados: decimal, money, numeric e smallmoney. A diferença entre cada tipo de dados é o intervalo de números que eles suportam e o espaço de armazenamento que cada coluna desse tipo de dados ocupa. Outros factores importantes incluem a precisão e a escala do tipo de dados.

*Precisão* constitui o número máximo de dígitos que se quer que um tipo de dados armazene. Por exemplo, o número 12.345,12 contém sete dígitos. A precisão inclui todos os dígitos

---

<sup>1</sup> Tipo de dados utilizado para armazenar identificadores globalmente únicos (globally unique identifiers – GUIDs). Um GUID é um mecanismo de numeração que garante que o mesmo número não será gerado em nenhuma tabela, bases de dados ou servidor em rede em todo o mundo. Um GUID é um número de 16 bytes, como este: 50295F55-4666-4358-B6C6-B9B709755BAC. GUIDs são úteis para identificar, copiar e controlar registos de bases de dados distribuídos.

antes e depois da vírgula de fração decimal. A escala do tipo de dados é o número de dígitos decimais; portanto, a escala de 12.345,12 é dois.

| Tipo de dados                  | Precisão | Escala | Intervalo de valores                         |
|--------------------------------|----------|--------|--|
| <b>decimal</b>                 | 9        | 0-5    | -9999999999 a 9999999999                     |
| <b>numeric</b>                 | 19       | 0-9    | $-10^{19} + 1$ a $10^{19-1}$                 |
| <b>money</b> <sup>2</sup>      | 18       | 4      | -922337203685477,5808 a 922337203685477,5808 |
| <b>smallmoney</b> <sup>3</sup> | 9        | 4      | -214748,3648 a 214748,3647                   |

- **decimal[(p[,s])]** – para valores decimais com precisão fixa (p) e escala (s) desde  $-10^{38} + 1$  até  $10^{38} - 1$ . A precisão pode ser qualquer número entre 1 e 38; a precisão predefinida é 18. A escala pode ser qualquer número entre 0 e a precisão. A precisão predefinida é 0.
- **numeric[(p[,s])]** – sinónimo de *decimal*. Para valores monetários, com quatro casas decimais, desde -922.337.203.685.477,5808 até 22.337.203.685.477,5807. Sinónimo de *decimal(19,4)*.
- **smallmoney** – para valores monetários, com quatro casas decimais, desde -214.748,3648 até 214.748,3647. Sinónimo de *decimal(10,4)*.

## Dados do tipo STRING

Para o registo de informação do tipo string o SQL Server dispõe de quatro diferentes tipos de dados: char, nchar, varchar e nvarchar.

| Tipo de String | Caracteres | Unicode  |
|----------------|------------|----------|
| fixo           | char       | nchar    |
| variável       | varchar    | nvarchar |

Tipos de dados variáveis utilizam o prefixo *var* e tipos de dados Unicode utilizam o prefixo *n*. Tipos de dados Unicode são um padrão de indústria desenhados para permitir que texto e símbolos de todos os idiomas sejam representados consistentemente e manipulados pelos computadores. Ou seja dados armazenados nos tipos de dados Unicode podem se transmitidos entre computadores pois vão manter a sua integridade. O que já não é garantido para dados do tipo Caracteres.

O tipo de dados *String* pode ser fixo ou de comprimento variável. Tipos de dados fixos (char, nchar) utilizam uma quantidade permanente de espaço independentemente do valor da coluna ou da variável. Por exemplo, se declarar uma coluna com o tipo char(10) e ela tiver o valor “demo”, esta coluna utiliza 10 bytes, mesmo que “demo” tenha apenas 4 letras. A mesma coluna como nchar(10) utiliza vinte bytes.

<sup>2</sup> Pode ser utilizado para valores financeiros

<sup>3</sup> Pode ser utilizado para preços

Em comparação, tipos de dados de comprimento variável ajustam o seu armazenamento para registrar o valor da coluna, mas exigem dois bytes adicionais para controlar o comprimento do valor. Ou seja, se declarar uma coluna como o tipo de dados `varchar(10)` e uma linha tiver o valor “demo”, ela utiliza seis bytes para armazenar o valor, dois bytes para controle do armazenamento e quatro bytes para armazenar o valor real.

- **char[(n)]** – para valores alfanuméricos de tamanho fixo até um limite de  $n=8000$  caracteres. O valor predefinido é 1.
- **varchar[(n)]** – para valores alfanuméricos de tamanho variável até um limite de  $n=8000$  caracteres. O valor predefinido é 1. Note que o número de bytes utilizado para armazenar a cadeia de caracteres depende do seu tamanho.
- **nchar(n)** – para valores alfanuméricos *unicode* de tamanho fixo até um limite de  $n=4000$  caracteres. O valor predefinido é 1.
- **nvarchar(n)** – para valores alfanuméricos *unicode* de tamanho variável até um limite de  $n=4000$  caracteres. O valor predefinido é 1. Note que o número de bytes utilizado para armazenar a cadeia de caracteres depende do seu tamanho e que são necessários 2 bytes para armazenar cada carácter *unicode*.

## Dados do tipo DATA e HORA

Em processos de negócios, a necessidade de armazenar atributos de data e hora é frequente. Não existem tipos de dados separados para registrar a data e a hora.

*Não utilize tipos de dados string para armazenar datas. Armazenar datas como strings torna a validação dos dados mais difícil e prejudica o desempenho da base de dados.*

| Tipo de dados        | Tamanho | Exatidão           | Valor mínimo e máximo    |
|----------------------|---------|--------------------|--------------------------|
| <b>datetime</b>      | 8 Bytes | 3,33 Milissegundos | 1-jan-1753 a 31-Dez-9999 |
| <b>Smalldatetime</b> | 4 Bytes | 1 Segundo          | 1-jan-1900 a 6-Jun-2079  |

## Dados do tipo BINÁRIOS

Variadas aplicações necessitam de armazenar imagens (como arquivos JPG, GIF e BMP) ou documentos (como ficheiros do WORD ou Excel). Para suportar essas necessidades, o SQL Server suporta tipos de dados binários. Informações binárias podem ser armazenadas em três tipos de dados: `binary(n)`, `varbinary(n)`, e `varbinary(Max)`.

| Tipo de dados         | Descrição                              | Valor máximo        |
|-----------------------|--|---------------------|
| <b>binary(n)</b>      | Dados binários de largura fixa         | 8000 Bytes          |
| <b>varbinary(n)</b>   | Dados binários de comprimento variável | 8000 Bytes          |
| <b>varbinary(Max)</b> | Dados binário longos                   | 2,147,483,647 Bytes |

## Validação dos dados

Determinar a forma como vai ser implementada a integridade de uma base de dados é sem dúvida uma tarefa importante no desenho da base de dados. Neste sentido deve ser identificado para cada campo ou atributo da tabela quais os *valores válidos*. O SQL Server fornece alguns mecanismos para reforçar a integridade da base de dados: NOT NULL, DEFAULT, PRIMARY KEY, UNIQUE, CHECK E FOREIGN KEY.

- **PRIMARY KEY** – para especificar que, em cada registo, a(s) coluna(s) tenha(m) um valor único e não nulo.
- **NOT NULL** – para impedir que sejam armazenados valores nulos (NULL) na coluna.
- **UNIQUE** – para especificar que, em cada registo, a coluna deve ter valores distintos.
- **DEFAULT** – para definir, em cada novo registo, um valor predefinido para a coluna (que pode ser obviamente alterado).
- **CHECK** – para limitar os valores admissíveis para a coluna.
- **[FOREIGN KEY] REFERENCES** – para impor a integridade referencial entre a(s) coluna(s) na nova tabela e a(s) coluna(s) de uma tabela relacionada,
- Também pode utilizar a palavra-chave **IDENTITY** na definição de uma coluna, de modo a que seja o *SQL SERVER* a gerar os valores para a coluna. Note, contudo, que apenas pode existir uma coluna *IDENTITY* por tabela.

## 2.4 Interfaces ao Modelo Relacional

Além de conceber o modelo relacional, Codd propôs ainda duas interfaces para a sua manipulação: a *álgebra relacional* e o *cálculo relacional*.

Tanto a álgebra relacional como o cálculo relacional, ao contrário das linguagens definidas para os modelos hierárquico e rede, são *set-oriented*, isto é, atuam sobre conjuntos e devolvem como resultados conjuntos.

Relativamente à álgebra relacional, dado que uma relação é um conjunto, então todas as operações da teoria de conjuntos (união, intersecção, diferença e produto artesiano) podem ser aplicadas às relações. Adicionalmente, definem-se outros operadores específicos do modelo relacional para:

- Seleção de tuplos numa relação.
- Seleção de atributos de uma relação (projeção).
- Junção de duas relações, originando uma terceira.
- Divisão de duas relações, originando uma terceira.
- Apresentando, resumidamente, cada uma destas operações:

**União (join).**  $\text{relação}_3 = (\text{relação}_1 \cup \text{relação}_2)$

Esta operação exige que as duas relações iniciais tenham esquemas compatíveis (i.é. além do mesmo grau é necessário que os domínios dos atributos correspondentes sejam os mesmos). Os tuplos da relação resultado ( $\text{relação}_3$ ) são tuplos da  $\text{relação}_1$  ou da  $\text{relação}_2$  ou de ambas. Os tuplos duplicados são eliminados (uma relação é um conjunto!).

| A  | B  | C  |
|----|----|----|
| a1 | b2 | c1 |
| a5 | b1 | c2 |
| a2 | b4 | c4 |
| a3 | b3 | c3 |

 $\cup$ 

| A  | B  | C  |
|----|----|----|
| a2 | b3 | c2 |
| a1 | b2 | c1 |
| a2 | b4 | c4 |

 $=$ 

| A  | B  | C  |
|----|----|----|
| a1 | b2 | c1 |
| a5 | b1 | c2 |
| a2 | b4 | c4 |
| a3 | b3 | c3 |
| a2 | b3 | c2 |

Seleciona cada registo de uma relação, associa-o com o registo correspondente da outra relação e apresenta-os como se fizessem parte de um único registo.

*Exemplo: Supondo as seguintes relações: Fornecedores e Fornecimentos. Junção das relações - Fornecedores e Fornecimentos sobre NF (da relação Fornecedores) e NF (da relação Fornecimentos)*

| Fornecedores |       |         | Fornecimentos |    |    |     |
|--------------|-------|---------|---------------|----|----|-----|
| NF           | Nome  | Cidade  | N_Fr          | NF | NM | Qtd |
| F1           | João  | Lisboa  | Fr1           | F1 | M1 | 300 |
| F2           | Maria | Porto   | Fr2           | F1 | M2 | 200 |
| F3           | Mario | Coimbra | Fr3           | F1 | M3 | 400 |
|              |       |         | Fr4           | F2 | M1 | 300 |
|              |       |         | Fr5           | F2 | M2 | 400 |
|              |       |         | Fr6           | F3 | M2 | 200 |
|              |       |         | Fr7           | F4 | M1 | 500 |

 $\cup$ 

| NF | Nome  | Cidade  | N_Fr | NM | Qtd |
|----|-------|---------|------|----|-----|
| F1 | João  | Lisboa  | Fr1  | M1 | 300 |
| F1 | João  | Lisboa  | Fr2  | M2 | 200 |
| F1 | João  | Lisboa  | Fr3  | M3 | 400 |
| F2 | Maria | Porto   | Fr4  | M1 | 300 |
| F2 | Maria | Porto   | Fr5  | M2 | 400 |
| F3 | Mário | Coimbra | Fr6  | M2 | 200 |

**Intersecção** relação = (relação1  $\cap$  relação2)

Da mesma forma que a anterior, a operação de intersecção exige que as duas relações iniciais tenham esquemas compatíveis. A relação final (relação3) consiste nos *tuplos* comuns as duas relações. Exemplo:

| A  | B  | C  |
|----|----|----|
| a1 | b2 | c1 |
| a5 | b1 | c2 |
| a2 | b4 | c4 |

 $\cap$ 

| A  | B  | C  |
|----|----|----|
| a2 | b3 | c2 |
| a1 | b2 | c1 |
| a2 | b4 | c4 |

 $=$ 

| A  | B  | C  |
|----|----|----|
| a1 | b2 | c1 |
| a2 | b4 | c4 |

**Diferença** relação3 = (relação1 - relação2)

Esta operação, da mesma forma que as duas anteriores, exige que as duas relações iniciais tenham esquemas compatíveis. A relação final (relação3) consiste nos *tuplos* da relação1 que não existem na relação2.

Exemplo:

| A  | B  | C  |
|----|----|----|
| a1 | b2 | c1 |
| a5 | b1 | c2 |
| a2 | b4 | c4 |

 $-$ 

| A  | B  | C  |
|----|----|----|
| a2 | b3 | c2 |
| a1 | b2 | c1 |
| a2 | b4 | c4 |

 $=$ 

| A  | B  | C  |
|----|----|----|
| a5 | b1 | c2 |

**Produto cartesiano** relação3 = (relação1  $\times$  relação2)

Esta operação produz uma relação final cujo esquema é a “soma” dos esquemas das relações iniciais (relação1 e relação2). Os *tuplos* desta relação correspondem a todas as combinações dos *tuplos* da relação1 com os *tuplos* da relação2. Ou seja, a sua cardinalidade é a multiplicação das cardinalidades das duas relações iniciais.

Exemplo:

| A  | B  | X | C  | D  | = | A  | B  | C  | D  |
|----|----|---|----|----|---|----|----|----|----|
| a1 | b2 |   | c2 | d3 |   | a1 | b2 | c2 | d3 |
| a5 | b1 |   | c1 | d2 |   | a1 | b2 | c1 | d2 |
| a2 | b4 |   |    |    |   | a5 | b1 | c2 | d3 |
|    |    |   |    |    |   | a5 | b1 | c1 | d2 |
|    |    |   |    |    |   | a2 | b4 | c2 | d3 |
|    |    |   |    |    |   | a2 | b4 | c1 | d2 |

**Seleção** relação2 =  $\sigma$  <condição>(relação1)

Esta relação toma uma relação inicial (relação1) e origina uma outra relação (relação2) com o mesmo esquema da primeira, mas em que apenas os *tuplos* que verificam uma dada condição estão presentes.

Exemplo:

| $\sigma(A=a2)$ | A  | B  | C  | = | A  | B  | C  |
|----------------|----|----|----|---|----|----|----|
|                | a2 | b3 | c2 |   | a2 | b3 | c2 |
|                | a1 | b2 | c1 |   | a2 | b4 | c4 |
|                | a2 | b4 | c4 |   |    |    |    |

**Projeção** relação2 =  $\Pi$ <lista\_atributos>(relação1)

A operação de projeção toma uma relação inicial (relação1) e origina uma outra relação (relação2) cujo esquema é reduzido ao conjunto de atributos presentes na lista\_atributos.

Exemplo:

| $\Pi (A,C)$ | A  | B  | C  | = | A  | C  |
|-------------|----|----|----|---|----|----|
|             | a2 | b3 | c4 |   | a2 | c4 |
|             | a1 | b2 | c1 |   | a1 | c1 |
|             | a2 | b4 | c4 |   |    |    |

Exemplo:

*Empregados* (Emp#, Nome, Categoria, Dep#); Emp# é chave da relação *Empregados*.

Projeção da tabela *Empregados* sobre os atributos Dep# e Categoria.

| $\Pi (Dep\#,Cat)$ | Empregados |      |     |      | = | Dep# | Cat |
|-------------------|------------|------|-----|------|---|------|-----|
|                   | Emp#       | Nome | Cat | Dep# |   | d1   | c1  |
|                   | e1         | n1   | c1  | d1   |   | d1   | c3  |
|                   | e2         | n2   | c2  | d2   |   | d2   | c1  |
|                   | e3         | n3   | c3  | d1   |   | d2   | c2  |
|                   | e4         | n4   | c1  | d2   |   | d3   | c2  |
|                   | e5         | n5   | c2  | d3   |   |      |     |
|                   | e6         | n6   | c2  | d3   |   |      |     |
|                   | e7         | n7   | c1  | d1   |   |      |     |



**Junção natural**  $\text{relação}_3 = (\text{relação}_1 \bowtie \text{relação}_2)$

Também designada simplesmente de junção esta operação é utilizada para combinar *tuplos* provenientes das duas relações distintas (*relação*<sub>1</sub> e *relação*<sub>2</sub>), com atributos comuns, numa só relação (*relação*<sub>3</sub>). A relação final contém um tuplo por cada combinação de *tuplos* das duas relações que possuam valores iguais nos atributos comuns. O esquema da relação final consiste na “soma” dos esquemas das relações iniciais. Exemplo:

| A  | B  | C  |           | C  | D  | = | A  | B  | C  | D  |
|----|----|----|-----------|----|----|---|----|----|----|----|
| a1 | b2 | c1 | $\bowtie$ | c2 | d3 |   | a1 | b2 | c1 | d2 |
| a5 | b1 | c2 |           | c1 | d2 |   | a5 | b1 | c2 | d3 |
| a2 | b4 | c4 |           | c2 | d1 |   | a5 | b1 | c2 | d1 |

**Divisão**  $\text{relação}_3 = (\text{relação}_1 \div \text{relação}_2)$

As relações iniciais (*relação*<sub>1</sub> e *relação*<sub>2</sub>) devem ser compatíveis para a divisão, ou seja, se X é o conjunto de atributos da relação 1 e Y é o conjunto de atributos da relação<sub>2</sub>, então  $X \supseteq Y$ . Os atributos presentes na relação final são os atributos da relação 1 que não figuram na relação<sub>2</sub>. Os *tuplos* da relação final são aqueles que existem na relação 1 em combinação com todos os *tuplos* da relação<sub>2</sub>.

Exemplo:

| A  | B  | C  | D  |        | C  | D  | = | A  | B  |
|----|----|----|----|--------|----|----|---|----|----|
| a1 | b2 | c2 | d3 | $\div$ | c2 | d3 |   | a1 | b2 |
| a5 | b1 | c1 | d2 |        | c1 | d2 |   | a2 | b4 |
| a2 | b4 | c1 | d2 |        |    |    |   |    |    |
| a3 | b5 | c4 | d4 |        |    |    |   |    |    |
| a2 | b4 | c2 | d3 |        |    |    |   |    |    |
| a1 | b2 | c1 | d2 |        |    |    |   |    |    |

Dadas as relações,

**Atribuição**

| Emp# | Proj# | Função |
|------|-------|--------|
| e1   | p1    | r1     |
| e2   | p3    | r1     |
| e2   | p2    | r2     |
| e3   | p2    | r1     |
| e3   | p3    | r1     |
| e4   | p1    | r1     |
| e5   | p3    | r2     |
| e6   | p1    | r3     |
| e6   | p2    | r3     |
| e6   | p3    | r3     |
| e7   | p1    | r1     |

**Projectos**

| Proj# | Designação | Fundos |
|-------|------------|--------|
| p1    | t1         | 2000   |
| p2    | t2         | 3000   |
| p3    | t3         | 50000  |

| Emp# | Função |
|------|--------|
| e6   | r3     |

## 2.5 As Doze Regras de Codd

Existem, atualmente vários sistemas de gestão de bases de dados que se dizem relacionais. A realidade é que, por vezes, não são mais que sistemas de gestão de ficheiros evoluídos, com algumas características de associação de tabelas [Date 1995].

Para impor alguma ordem no desenvolvimento de SGBD relacionais, Codd enunciou, em 1985, 12 principais regras. Destas, pelo menos 6 devem ser satisfeitas para que um SGBD seja considerado relacional. Há ainda a REGRA 0 que é geral.

0. **Manipulação de Bases de Dados Relacionais** - Para que um sistema seja (ou pretenda ser) um sistema de Bases de Dados Relacional, esse sistema tem de ser capaz de gerir bases de dados inteiramente pela sua capacidade relacional.
1. **Representação da Informação** - numa base de dados relacional, todos os dados, incluindo o próprio dicionário de dados são representados de uma só forma, em tabelas bidimensionais.
2. **Acessibilidade Lógica Garantida** - cada elemento de dados fica bem determinado pela combinação de nome da tabela onde está armazenado, valor da chave primária e respetiva coluna atributo. *Cada pedaço de informação (valor atômico) numa base de dados relacional é garantidamente acessível por uma combinação do nome da tabela, valor chave e nome da coluna;*
3. **Representação sistemática de informação não constante** - valores nulos (*nulls*) são suportados para representar informação não disponível ou não aplicável, independentemente do domínio dos respetivos atributos. Valores nulos (distintos de uma *string* de caracteres vazia, ou de uma *string* preenchida a espaços ou de um número a zeros) são suportados em modelos integralmente relacionais para representar informação não constante e inaplicável de uma maneira sistemática independentemente do tipo de dados;
4. **Catálogo Permanentemente Dinâmico** - os *metadados* são representados e acedidos da mesma forma que os próprios dados. A descrição da base de dados é representada a nível local da mesma maneira que outros dados, para que utilizadores autorizados possam aplicar a mesma linguagem relacional para sua interrogação como o fazem para outro tipo de informação.
5. **Sub-linguagem Compreensível** - apesar de um sistema relacional poder suportar várias linguagens, deverá existir pelo menos uma linguagem com as seguintes características:
  - Manipulação de dados, com possibilidade de utilização interativa ou em programas de aplicação.
  - Definição de dados.
  - Definição de *views*.
  - Definição de restrições de integridade.
  - Definição de acessos (autorizações).
  - Manipulação de transações (*commit*, *rollback*, etc.).
6. Numa *view*, todos os dados atualizáveis que forem modificados, devem ver essas modificações traduzidas nas tabelas base.
7. **Inserção, Alteração e Anulação de Alto Nível** - capacidade de tratar uma tabela (base ou virtual) como se fosse um simples operando (ou seja, utilização de uma linguagem *set-oriented*), tanto em operações de consulta como de atualização.

8. Independência Física dos Dados - alterações na organização física dos ficheiros da base de dados ou nos métodos de acesso a esses ficheiros (nível interno) não devem afetar o nível conceptual - *independência física*.
9. Independência Lógica dos Dados - alterações no esquema da base de dados (nível conceptual), que não envolvam remoção de elementos, não devem afetar o nível externo - *independência lógica*.
10. Independência na Distribuição - as restrições de integridade devem poder ser especificadas numa linguagem relacional, independentemente dos programas de aplicação, e armazenadas no dicionário de dados. Pontos relativos à integridade específica de uma base de dados devem ser definidos na sub-linguagem e guardados no catálogo e não no programa que a manipula.
11. O facto de uma base de dados estar centralizada numa máquina, ou distribuída por várias máquinas, não deve repercutir-se ao nível da manipulação dos dados. *Quer um sistema suporte ou não distribuição de bases de dados, tem de ter uma sub-linguagem que suporte bases de dados distribuídas sem que isso bloqueie a atividade de programas ou utilizações diretas.*
12. Regra da Não-Subversão - se existir no sistema uma linguagem de mais baixo-nível (tipo *record-oriented*), ela não deverá permitir ultrapassar as restrições de integridade e segurança. *Se um sistema relacional tem uma linguagem de baixo nível, esse baixo nível não pode ser usado para subverter ou sobrepor-se a regras de integridade expressas no nível superior da linguagem relacional.*

Atualmente, não existe nenhuma implementação do modelo relacional que, à luz das doze regras de Codd, possa ser considerada completamente relacional. Aliás, segundo alguns autores, não parece provável que isso alguma vez venha a acontecer.

## 2.6 Exercícios do Modelo Relacional

**Ex.MR.01** - Analise as seguintes relações e indique o valor lógico de cada uma das proposições:

Alunos

| <u>Numero</u> | Nome | Data_Nasc | Morada | CodigoPostal |
|---------------|------|-----------|--------|--------------|
|               |      |           |        |              |
|               |      |           |        |              |
|               |      |           |        |              |

|  |                              |                              |
|--|------------------------------|------------------------------|
| Podem existir múltiplos alunos com o mesmo número?   | <input type="checkbox"/> sim | <input type="checkbox"/> não |
| O maior valor possível para o número de um aluno, consiste na totalidade de alunos existentes? | <input type="checkbox"/> sim | <input type="checkbox"/> não |
| Podem existir alunos que possuam a mesma morada?   | <input type="checkbox"/> sim | <input type="checkbox"/> não |
| Todos os alunos têm nome diferente?  | <input type="checkbox"/> sim | <input type="checkbox"/> não |
| Podem existir alunos com nome igual?   | <input type="checkbox"/> sim | <input type="checkbox"/> não |
| Podem existir alunos com nome, morada e data de nascimento igual?                              | <input type="checkbox"/> sim | <input type="checkbox"/> não |

**Ex.MR.02** – Um departamento de uma universidade tem cinco carros para utilização dos seus funcionários e docentes, para deslocações em serviço. Necessita de registar a utilização dos carros pelos docentes de cada departamento, e pelos funcionários, assim como os diferentes custos associados a cada utilização (combustíveis, autoestradas, parking, etc.), e os Kms percorridos.

- Identifique as entidades de dados existentes
- Identifique as relações entre as entidades de dados
- Defina as restrições de integridade necessárias
- Construa e teste a base de dados

**Ex.MR.03** - Inicie uma sessão no *SQLServer* instalado na sua estação de trabalho. Pretende-se que crie numa nova base de dados, com o nome “Ex.MR.03”, as tabelas apresentadas de seguida, realizando as tarefas necessárias para impor integridade nos dados (chave primária, chave estrangeira e relações entre tabelas).

#### **dbo.Alunos**

| PK/FK | Nome Campo     | Tipo Campo | Null | Descrição                    |
|-------|----------------|------------|------|------------------------------|
| PK    | NumeroAluno    | int        | Não  | Número do aluno              |
| -     | NomeCompleto   | nchar (50) | Não  | Nome completo do aluno       |
| -     | EstadoCivil    | nchar (1)  | Não  | Estado Civil IN (S,C,V,D,U). |
| -     | DataNascimento | date       | Não  | Data de nascimento           |
| FK    | IDCurso        | tinyint    | Não  | Ligação a Tables\dbo.Cursos  |

#### **dbo.Cursos**

| PK/FK | Nome Campo      | Tipo Campo | Null | Descrição     |
|-------|-----------------|------------|------|---------------|
| PK    | IDCurso         | tinyint    | Não  | IDCurso       |
|       | DesignaçãoCurso | nchar(50)  | Não  | Nome do curso |

Registe informação nas tabelas iniciadas de forma a testar o modelo de dados.

Considere a seguinte informação para melhorar o modelo de dados:

- Pretende-se que altere o modelo de dados com o objetivo de suportar a Nacionalidade para os alunos inscritos. Considera-se que um aluno apenas tem uma Nacionalidade. Considere o ficheiro “Nacionalidades.xls” disponível para apoiar a implementação deste requisito.
- É necessário suportar o requisito de um aluno mudar de curso no ano letivo seguinte. Contudo num ano letivo um aluno apenas se pode inscrever num curso. Pretende-se neste caso que não se perca a informação do curso anterior, caso o aluno mude de curso.

**Ex.MR.04** - Pretende-se que desenvolva uma base de dados para o registo e controlo do pagamento de mensalidades referentes à frequência de alunos numa escola privada.

Desenvolva as tarefas necessárias para implementar os seguintes requisitos: (a) Criar uma nova base de dados com o nome “Ex.MR.04”; (b) criar as tabelas, campos, tipo de campos, relações e restrições apresentadas no dicionário de dados que se segue.

## Alunos

| PK/FK | Nome Campo       | Tipo Campo | Null | Descrição                      |
|-------|------------------|------------|------|--------------------------------|
| PK    | NumeroAluno      | Número     | N    | Número do aluno                |
| -     | PrimeirosNomes   | Texto (20) | N    | Primeiro (s) nome (s) do aluno |
| -     | UltimosNomes     | Texto (20) | N    | Último (s) nome (s) do aluno   |
| -     | DataNascimento   | Data       | N    | Data de nascimento             |
| -     | Sexo             | Texto (1)  | N    |                                |
| -     | ValorMensalidade | Moeda      | S    | Valor da mensalidade           |

Restrições de integridade a implementar para *Alunos*:

- O campo *NumeroAluno* deve ser incrementado em 1 a cada novo registo com o primeiro registo a iniciar em 100.
- Adicione uma restrição ao campo *DataNascimento* que impeça que o valor registado seja superior ou igual à data do computador.

## Pais

| PK/FK | Nome Campo       | Tipo Campo | Null | Descrição                        |
|-------|------------------|------------|------|----------------------------------|
| PK    | NumeroPessoa     | Número     | N    | Número da Pessoa                 |
| -     | PrimeirosNomes   | Texto (20) | N    | Primeiro (s) nome (s) do pai/mãe |
| -     | UltimosNomes     | Texto (20) | N    | Último (s) nome (s) do pai/mãe   |
| -     | NIC              | Número     | N    | Número de Identificação Civil    |
| -     | Profissão        | Texto (20) | N    | Profissão atual                  |
| -     | VencimentoMensal | Moeda      | S    | Vencimento Mensal Bruto          |

Considere como informação a registar nas seguintes tabelas:

## Alunos

| NumeroAluno | PrimeirosNomes | UltimosNomes | DataNascimento      | Sexo | ValorMensalidade |
|-------------|----------------|--------------|---------------------|------|------------------|
| 100         | Pedro          | Costa        | 2009-11-05 00:00:00 | M    | 0,00             |
| 101         | Maria          | Costa        | 2009-11-05 00:00:00 | F    | 0,00             |
| 102         | João           | Silva        | 2009-12-12 00:00:00 | M    | 0,00             |
| 103         | Joaquim        | Costa        | 2008-01-06 00:00:00 | M    | 0,00             |
| 104         | Tiago          | Lopes        | 2008-02-12 00:00:00 | M    | 0,00             |
| 105         | Madalena       | Lopes        | 2009-12-26 00:00:00 | F    | 0,00             |

## Pais

| NumeroPessoa | PrimeirosNomes | UltimosNomes | NIC      | Profissão          | VencimentoMensal |
|--------------|----------------|--------------|----------|--------------------|------------------|
| 1            | Henrique       | Costa        | 12345678 | Técnico Superior   | 2500,00          |
| 2            | Manuela        | Jesus Costa  | 12345679 | Docente            | 2200,00          |
| 3            | José           | Silva        | 87654321 | Técnico Superior   | 3500,00          |
| 4            | Maria          | Silva        | 98765432 | Educadora Infantil | 2100,00          |
| 5            | Carlos         | Lopes        | 22334455 | Docente            | 2150,00          |
| 6            | Fátima         | Lopes        | 33445566 | Técnico Superior   | 2700,00          |

Desenvolva o modelo de dados de forma a implementar os seguintes requisitos:

- Pretende-se registar a filiação dos alunos inscritos. Para cada aluno deve ser indicado o Pai e a Mãe, sendo que podem existir mais que um aluno com a mesma filiação. Os alunos com os números 100, 101 e 103 são filhos do Henrique Costa (Pai) e Manuela Jesus Costa (Mãe); o aluno com o número 102 é filho de José Silva (Pai) e Maria Silva (Mãe); os alunos com os números 104 e 105 são filhos de Carlos Lopes (Pai) e Fátima Lopes (Mãe).

- b. *Adicione um campo denominado “AnoLectivo” à tabela Alunos, sendo que deve apresentar para todos os inscritos o valor 2011-2012.*

**Ex.MR.05** - O exemplo seguinte foca-se no contexto da gestão de títulos/livros de uma Biblioteca assim como os processos de requisição dos mesmos por parte dos seus leitores. Crie uma nova base de dados atribuindo-lhe a designação “Ex.MR.05”. Implemente na base de dados a seguinte tabela.

### Livros

| PK/FK | Nome Campo          | Tipo Campo  | Null | Descrição      |
|-------|---------------------|-------------|------|----------------|
| PK    | COTA                | Texto (10)  | N    | Ex: TGI-1876-9 |
| -     | Titulo              | Texto (100) | N    |                |
| -     | AnoPublicação       | Texto(4)    | N    | Ex: 1982       |
| -     | NumeroPáginas       | Número      | N    |                |
| -     | ISBN                | Texto(25)   | N    |                |
| -     | PermissãoEmprestimo | Sim/Não     | S    |                |

Desenvolva o modelo de dados com o objetivo de implementar os seguintes requisitos que se pretende que a base de dados suporte:

- É necessário registar o empréstimo de livros aos leitores da Biblioteca, previamente registados na base de dados. Considere que num registo de empréstimo podem ser requisitados mais do que um livro. É ainda necessário registar para um empréstimo a data de início e a data de término do mesmo. Sobre o leitor é necessário que exista na base de dados a seguinte informação: Número Leitor (de numeração automática), Data Registo, Primeiro Nome(s), Último Nome(s), Profissão. Um leitor não deve poder realizar empréstimos de livros se não estiver registado na base de dados.
- É necessário registar a coleção a que um livro pertence. Exemplo de coleções: Línguas, Matemática, Estatística, Informática, Design, Gestão e Engenharia. Um livro apenas pertence a uma coleção.

Para a estrutura da base de dados implementada introduza alguns registos de forma a testar o seu funcionamento e a sua integridade (três no mínimo).

**Ex.MR.06** - Pretende-se registar o aluguer de quartos numa cadeia de hotéis a nível nacional. Um cliente pode num único processo de reserva, reservar mais do que um quarto com datas que podem ser não coincidentes, e em hotéis diferentes. Ou seja, pode reservar um quarto para dois dias e outro para quatro dias no mesmo hotel, ou, se necessário fazer uma reserva de dois dias para um quarto num hotel e em simultâneo mais dois dias num outro hotel do grupo. É, sempre, necessário registar as pessoas que vão ficar em cada um dos quartos incluídos no processo de reserva (através do registo obrigatório do seu NIC, Primeiros Nomes, Últimos Nomes, Sexo, e a sua Idade).

- Identifique as entidades de dados existentes
- Identifique as relações entre as entidades de dados
- Defina as restrições de integridade necessárias
- Construa e teste a base de dados

**Ex.MR.07** – O exemplo seguinte tem por finalidade suportar a venda ONLINE de produtos de várias categorias. Crie uma nova base de dados, atribuindo-lhe a designação “Ex.MR.07”. Estructure uma nova tabela de acordo com os campos apresentados na tabela seguinte:

### Produtos

| PK/FK | Nome Campo         | Tipo Campo           | Null | Descrição                        |
|-------|--------------------|----------------------|------|----------------------------------|
| PK    | CódigoDoProduto    | Numeração automática | N    | Código atribuído automaticamente |
| -     | NomeDoProduto      | Texto (100)          | N    |                                  |
| -     | IDCategoria        | Número               | N    | Ligação à tabela Categorias      |
| -     | PreçoUnitário      | Moeda                | N    |                                  |
| -     | Existências        | Número               | N    |                                  |
| -     | ExistênciasMinimas | Número               | N    |                                  |

É necessário indicar a categoria do produto. Considere que um produto apenas deve ter atribuída uma categoria, e na mesma categoria podem estar vários produtos. Contudo esta tarefa fica mais complexa quando é necessário atribuir a um produto uma subcategoria.

Uma subcategoria apenas faz parte de uma categoria de nível 0, mas pode incluir várias subcategorias dependentes. Pode, contudo, acontecer que uma subcategoria possa mudar de dependência, obrigando assim a mudar também todas as suas dependentes. As subcategorias são utilizadas para melhor classificar os produtos. Uma categoria pode estar segmentada em várias subcategorias, e uma subcategoria em várias subcategorias de 2º nível (ou sub-subcategoria). O exemplo seguinte implementa este requisito até ao 3º nível de dependência.

Pretende-se que desenvolva o modelo de dados para suportar este requisito de acordo com o modelo apresentado.

### Modelo

*Categoria A*

*Subcategoria A1*

*Subcategoria A2*

*Subcategoria A2.1*

*Subcategoria A2.2*

*Subcategoria A2.2.1*

*Subcategoria A2.2.2*

...

*Subcategoria A2.2.n*

...

*Subcategoria A2.n*

...

*Subcategoria An*

*Categoria B*

...

*Categoria N*

### Exemplo

*Informática*

*Impressoras*

*Computadores*

*Secretária*

*Portáteis*

*Netbook*

*Ultrabook*

*Mobilidade*

*Tablets*

*Software*

*Antivírus*

*Multimédia*

*Office*

*Livros*

...

*Música*

Um outro requisito a suportar pela base de dados está relacionado com o facto de um produto pode ser fornecido por mais que um fornecedor. É necessário, contudo que seja definido um ranking dos fornecedores para um produto, definindo qual a prioridade de encomendar produtos a um fornecedor. A determinado momento um fornecedor pode alterar a sua posição no ranking se apresentar mais vantagens que os restantes. Os fornecedores podem ser de diferentes nacionalidades. É necessário registar a informação necessária sobre cada fornecedor de forma a suportar todo o processo de encomendas”.

### 3. SQL – *Structured Query Language*

A SQL (*Structured Query Language*) é uma linguagem padrão para a interrogação de bases de dados relacionais. A SQL é uma linguagem declarativa (não procedimental). Uma pesquisa (*query*) em SQL especifica o que (*what*) deve ser procurado, mas não como (*how*) deve ser procurada.

A SQL consiste numa linguagem de definição de dados (**DDL – *Data Definition Language***) e numa linguagem de manipulação de dados (**DML – *Data Manipulation Language***) e numa linguagem de controlo de acesso a dados (**DCL – *Data Control Language***).

**DML** – para manipular dados (inserir dados nas tabelas, atualizar dados existentes e eliminar dados das tabelas) e para interrogar (baseado na álgebra relacional) a base de dados.

**DDL** – para criar/alterar/eliminar objetos (tabelas, restrições, índices, etc.) da base de dados.

**DCL** – para tratar os aspetos relacionados com a autorização de acesso aos dados. Permite que o utilizador controle quem tem acesso para visualizar ou manipular dados dentro da base de dados (comandos: GRANT, REVOKE e DENY)).

#### 3.1 SQL – *Data Manipulation Language* (DML)

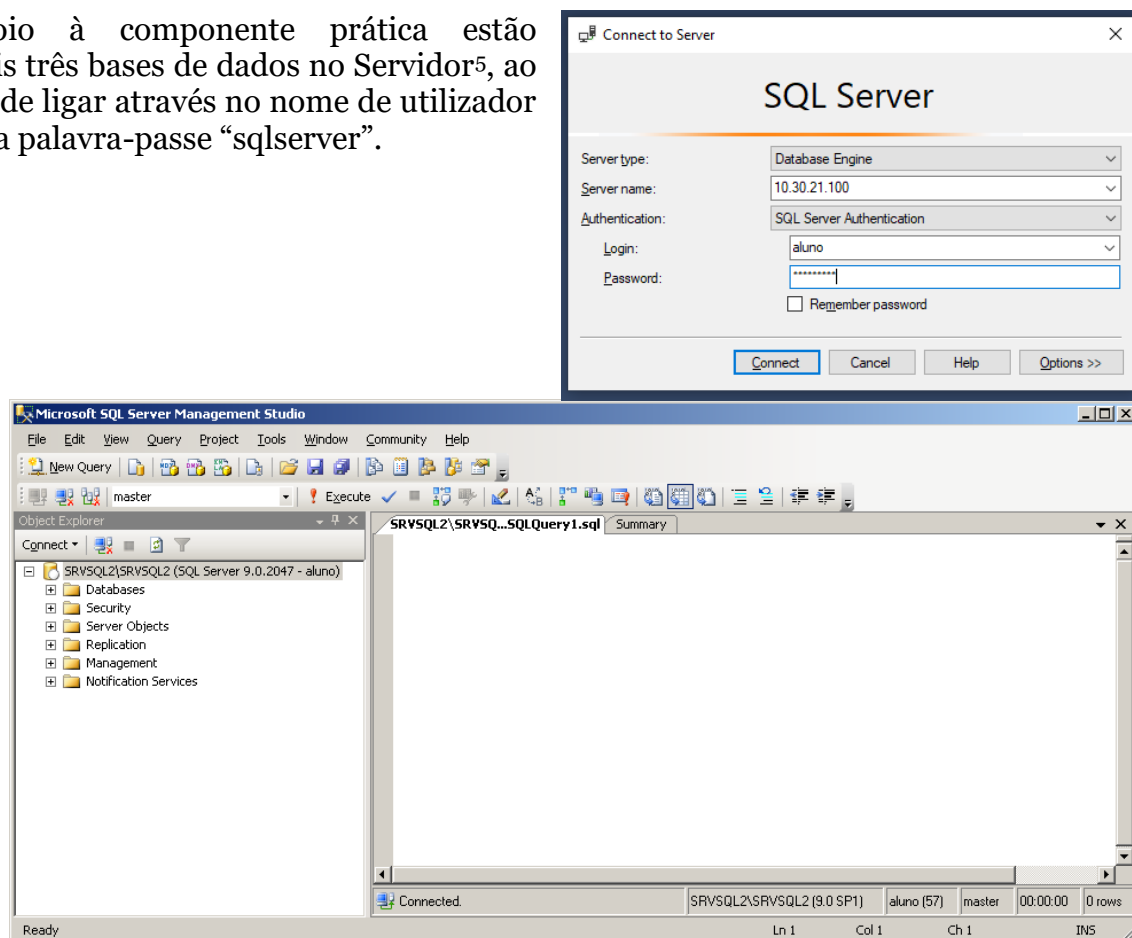
A documentação do *SQL Server* fornece-nos a seguinte sintaxe:

```
SELECT [ ALL | DISTINCT ] select_list
[ INTO new_table ]
FROM table_source
[ WHERE search_condition ] - aplicada a registos individuais.
[ GROUP BY group_by_expression ]
[ HAVING search_condition ] - aplicada a grupos de registos.
[ ORDER BY order_expression [ ASC | DESC ] ]
```



No servidor com o IP **10.30.21.100**, estão disponíveis, várias bases de dados para apoio ao trabalho prático. Estabeleça uma sessão com o servidor utilizando a aplicação “SQL Server Management Studio”<sup>4</sup>.

Para apoio à componente prática estão disponíveis três bases de dados no Servidor<sup>5</sup>, ao qual se pode ligar através no nome de utilizador “aluno” e a palavra-passe “sqlserver”.



As instruções SQL apresentadas neste documento, estão preparadas para três bases de dados:

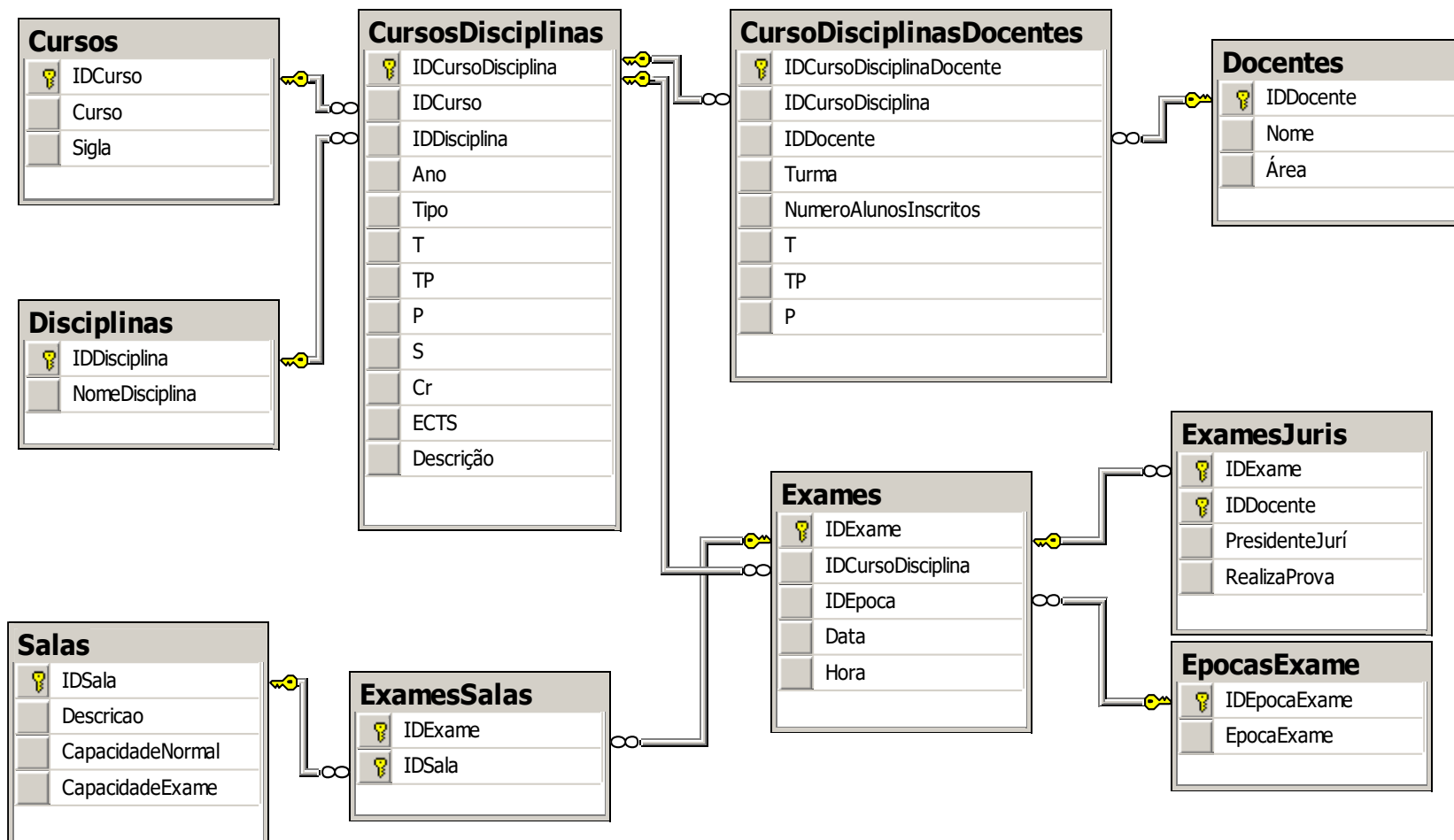
1. Base de dados EXAMES - constitui uma base de dados de apoio a processo de construção do calendário de exames para os cursos da ESTG. Permite agendar um exame por cada época de exames para cada uma das disciplinas dos cursos existentes. Por outro lado permite que seja nomeado um júri para cada um dos exames agendados. Os dados existentes dizem respeito ao calendário de exames do ano letivo 2004.2005.
2. Base de dados VENDAS - constitui uma versão limitada da base de dados *AdventureWorks* apenas para o cenário de vendas de bicicletas e de outros componentes.
3. Base de dados ProDados

---

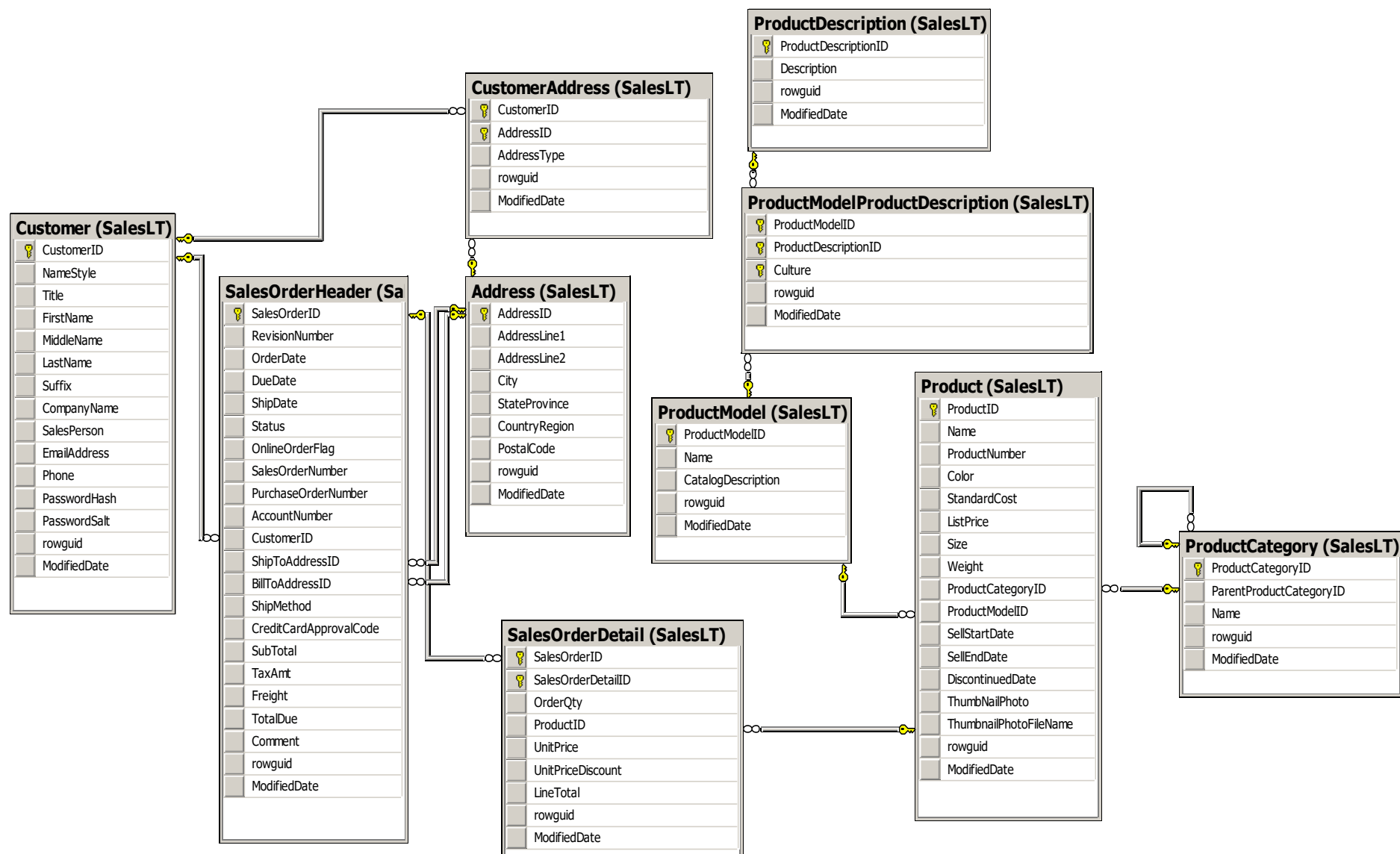
<sup>4</sup> Este servidor está apenas disponível na rede local da Escola.

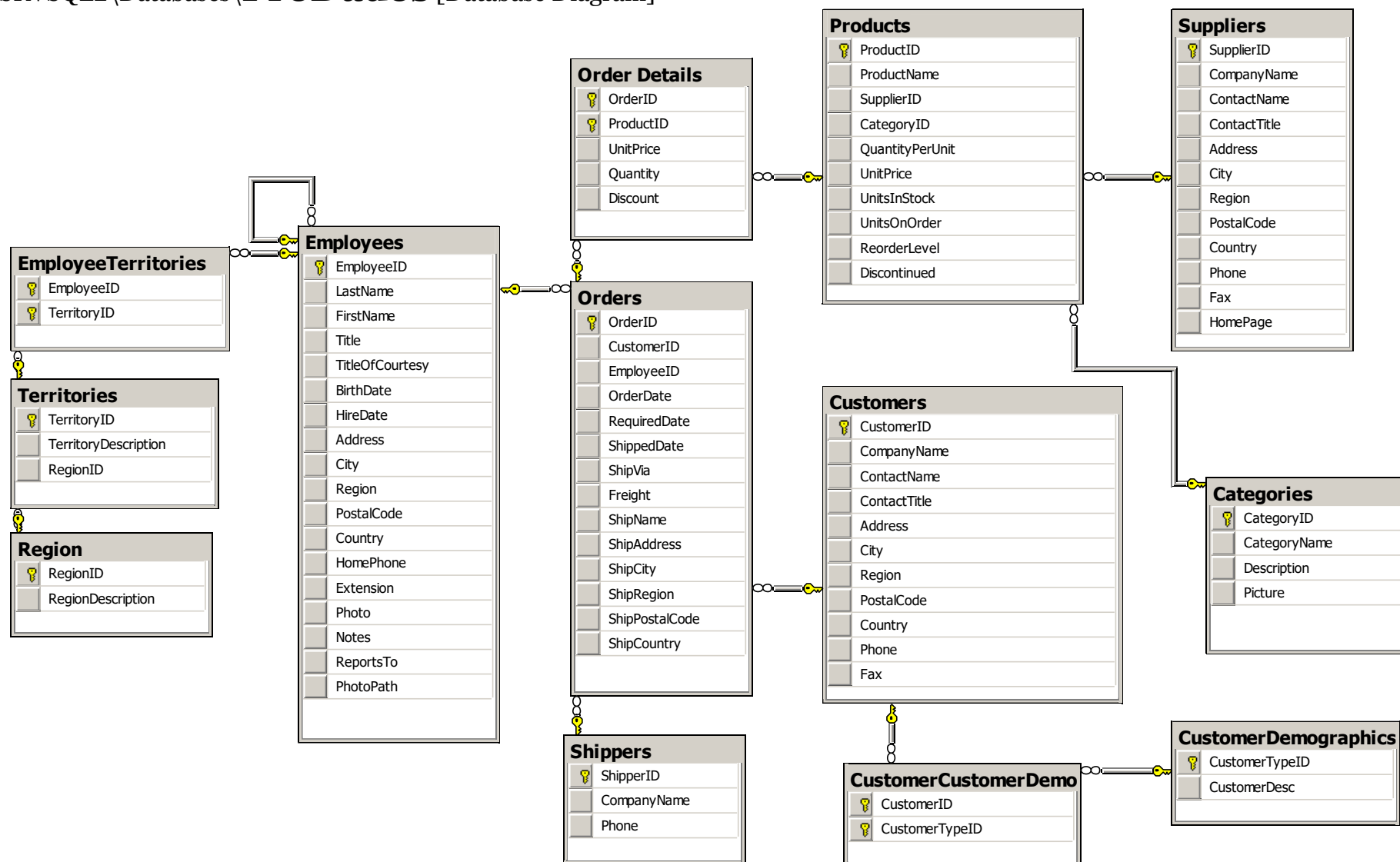
<sup>5</sup> Servidor com sistema operativo Windows Server 2003

SRVSQL2\Databases\Exames [Database Diagram]



SRVSQL2\Databases\ **Vendas** [Database Diagram]





Se iniciou a sessão com sucesso, crie uma nova *Query*, para a realização das instruções DML (*Data Manipulation Language*) que se seguem:

- 1    `Use Exames`  
     `Select * from Cursos`  
     (o que faz o “\*” ?)
- 2    `Select Curso from Cursos`
- 3    `Use Vendas`  
     `Select Name from SalesLT.Product`  
     (Quantos registos de produtos existem ? \_\_\_\_\_)
- 4    `Use Exames`  
     `Select Curso, Sigla from Cursos`
- 5    `Use Vendas`  
     `Select FirstName, LastName, CompanyName`  
     `From SalesLT.Customer`  
     (Quantos registos de clientes existem ? \_\_\_\_\_)
- 6    `Select CompanyName from SalesLT.Customer`  
     (De quantas empresas são os clientes existentes ? \_\_\_\_\_)
- 7    `Select Distinct CompanyName from SalesLT.Customer`  
     (E agora ? \_\_\_\_\_), Este campo é redundante ? Sim/Não
- 8    `Use Exames`  
     `Select IDCurso from Cursos`  
     (não aparecem registos duplicados, porquê?)
- 9    `Select IDCurso from CursosDisciplinas`  
     (aparecem registos duplicados!)
- 10   `Select Distinct IDCurso from CursosDisciplinas`  
     (Ainda são listados registos duplicados ?)
- 11   Utilize a base de dados “Exames”. Quantas disciplinas existem na tabela Disciplinas”?

### 3.1.1 Operadores

#### 3.1.1.1 Operadores relacionais

(=) Igual a

##### Exemplo 1

```
Use Exames
Select * from CursosDisciplinas where IDCurso = 2
```

## Exemplo 2

```
Use Vendas
select * from SalesLT.SalesOrderHeader
Where CustomerID=61
```

**Exemplo 3:** Na base de dados “Exames” existe algum exame agendado para o “E1.2 Laboratório de Informática”? Se sim, quantos ?

## (>) Maior que

**Exemplo 4:** Quantos registos de vendas apresentam um valor total (TotalDue) superior a 50000 ?

```
Use Vendas
Select * from SalesLT.SalesOrderHeader
Where TotalDue>50000
```

## (<) Menor que

**Exemplo 5:** Quantos registos de vendas apresentam um valor total (TotalDue) inferior a 50000?

```
Use Vendas
Select * from SalesLT.SalesOrderHeader
Where TotalDue<50000
```

## (>=) Maior ou igual

**Exemplo 6:** Na base de dados “Vendas” pretende-se listar os produtos com um Custo Base (StandardCost) maior ou igual a 1550.

## (<=) Menor ou igual

**Exemplo 7:** Na base de dados “Vendas” pretende-se listar os produtos com um Custo Base (StandardCost) menor ou igual a 1550.

**Exemplo 8:** Na base de dados “Exames” pretende-se saber quais os exames agendados para o mês de Julho?

```
Use Exames
Select * from Exames
where Data >= '2005-07-01' and data <= '2005-07-31'
```

## (<>,!=) Diferente

**Exemplo 9:** Na base de dados “Exames” pretende-se saber quais os exames agendados excepto para a “Época Normal – Fevereiro”?

```
Use Exames
Select * from EpocasExame
Select * from Exames where IDEpoca <> 1
```

### 3.1.1.2 Operadores lógicos

**AND** Condição1 **AND** Condição2

#### Exemplo 10

```
Use Exames
Select * from CursosDisciplinas
where IDCurso = 2 and Tipo = '1ºSem'
```

#### Exemplo 11

```
Use Exames
Select * from CursosDisciplinas
where IDCurso = 2 and Ano = 2 and Tipo = '1ºSem'
```

**Exemplo 12:** Na base de dados “Vendas” pretende-se listar os produtos de cor (Color) “Blue” e apenas os do tamanho (Size) igual a M.

**OR** Condição1 **OR** Condição2

**Exemplo 13:** Na base de dados “Vendas” quais os produtos das categorias “Mountain Frames” ou “Road Frames”?

**NOT** Condição

Using NOT negates an expression. The following table shows the results of comparing TRUE and FALSE values using the NOT operator.

|         | NOT     |
|---------|---------|
| TRUE    | FALSE   |
| FALSE   | TRUE    |
| UNKNOWN | UNKNOWN |

**Exemplo 14:** Na base de dados “Exames”, pretende-se listar todos os exames agendados, excepto os exames agendados para a Época Especial (Exames.IDEpoca = 3)

```
Use Exames
Select * from Exames
Where NOT IDEpoca = 3
```

**Between** – permite especificar intervalos:

Sintaxe: *Where* valor [**NOT**] **BETWEEN** valor1 **AND** valor2

#### Exemplo 15

```
Use Exames
Select * from CursosDisciplinas
where ( IDCurso Between 2 and 4) and Ano = 2
```

### Exemplo 16

```
Use Vendas
Select * from SalesLT.SalesOrderHeader
Where (TotalDue Between 10000 and 30000)
```

**Exemplo 17** - Na base de dados “Vendas” quais os produtos que pertencem à gama de modelos entre 10 e 22 ?

**IN** – permite especificar conjuntos de valores

Sintaxe: *Where* valor **[NOT] IN** (valor1, valor2, ..., valork)

**Exemplo 18**: Na base de dados “Vendas” quais as subcategorias de produtos que fazem parte das categorias 1 e 2 ?

```
Use Vendas
Select * from salesLT.Productcategory
Where ParentProductCategoryID IN (1,2)
```

**Exemplo 19**: Na base de dados “Vendas” quais as subcategorias de produtos que **não** fazem parte das categorias 1 e 2 ?

```
Use Vendas
Select * from salesLT.Productcategory
Where ParentProductCategoryID NOT IN (1,2)
```

**IS** – tratamento de nulos (NULL)

Sintaxe: *Where* valor **IS [NOT] NULL**

**Exemplo 20**: Na base de dados “Vendas” quais as produtos em que os campos Color e Size são NULL ?

```
Use Vendas
select * from SalesLt.Product
Where Color IS NULL and Size IS NULL
```

**Exemplo 21**: Na base de dados “Vendas” quais as produtos em que o campo Color não é NULL mas o campo Size é NULL ?

```
Use Vendas
select * from SalesLt.Product
Where Color IS NOT NULL and Size IS NULL
```

**LIKE** – Comparação de *Strings*

Sintaxe: *WHERE* nome **LIKE** ‘%da%’

**Exemplo 22**: Na base de dados “Exames”, quais os Docentes em que o nome do docente inicia com a letra “M”. Ordene a consulta pelo nome do docente.

```
Use Exames
Select * from Docentes
Where Nome Like 'M%'
Order By Nome
```



### Exemplo 23:

```
Use Exames
Select * From Docentes
Where nome like '%D%'
```

**Exemplo 24:** Na base de dados “Vendas” quais os produtos em que o nome do produto inicia com a letra “M”?

### 3.1.2 Alias (renaming e aliasing)

O comprimento de uma instrução torna mais fácil cometer erros de sintaxe, e instruções longas são mais complicadas de entender. As *Alias* são utilizadas para simplificar e evitar erros nas cláusulas SELECT e FROM. Permitem substituir nomes longos de tabelas por expressões mais curtas.

Exemplo:

*SalesLT.SalesOrderHeader.OrderDate*

Pode ser substituído por *SOH.OrderDate*

**Exemplo 25:** Realize a seguinte pesquisa na base de dados “Vendas”

```
Use Vendas
Select P.ProductNumber, P.Name from SalesLt.Product as P
Where P.ProductModelID = 7
```

Resultado

|   | ProductNumber | Name                          |
|---|---------------|-------------------------------|
| 1 | FR-T98Y-60    | HL Touring Frame - Yellow, 60 |
| 2 | FR-T98Y-46    | HL Touring Frame - Yellow, 46 |
| 3 | FR-T98Y-50    | HL Touring Frame - Yellow, 50 |
| 4 | FR-T98Y-54    | HL Touring Frame - Yellow, 54 |
| 5 | FR-T98U-46    | HL Touring Frame - Blue, 46   |
| 6 | FR-T98U-50    | HL Touring Frame - Blue, 50   |
| 7 | FR-T98U-54    | HL Touring Frame - Blue, 54   |
| 8 | FR-T98U-60    | HL Touring Frame - Blue, 60   |

### 3.1.3 Subconsultas SQL

É possível inserir uma instrução SQL dentro de outra instrução. Pode ser feito através da cláusula **WHERE** ou através da cláusula **HAVING**. O operador de comparação utilizado pode ser =, >, <, >=, <=, !=. Também pode ser utilizado um operador de texto como o **"LIKE"**.

A sintaxe é a seguinte:

```
SELECT "column_name1"
FROM "table_name1"
WHERE "column_name2" [Operador de comparação]
    (SELECT "column_name3"
     FROM "table_name2"
     WHERE [condição])
```

**Exemplo 26:** Na base de dados “Exames”, quais as disciplinas para o curso de Engenharia Informática?.

```
Use Exames
Select * From CursosDisciplinas
Where IDCurso =
(Select IDCurso From Cursos Where Curso = 'Engenharia Informática')
```

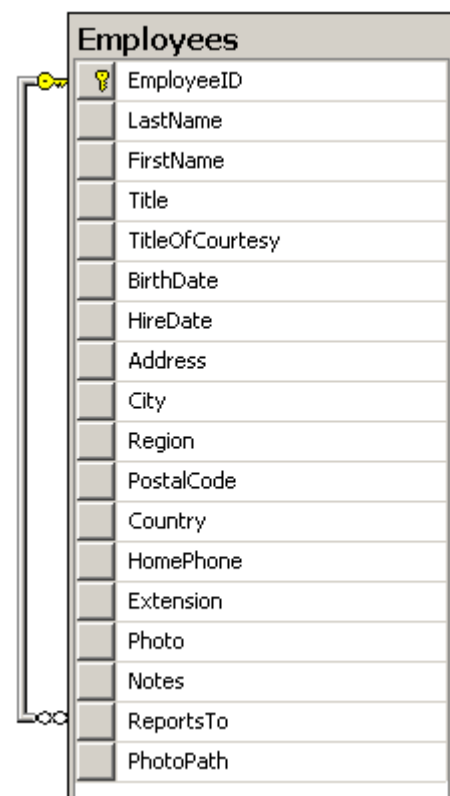
**Exemplo 27:** Na base de dados “Vendas”, quais os produtos da categoria “Brakes”?.

```
Use Vendas
Select * from SalesLT.Product
Where SalesLT.Product.ProductCategoryID =
(Select SalesLT.ProductCategory.ProductCategoryID from
SalesLT.ProductCategory where SalesLT.ProductCategory.Name =
'Brakes')
```

**Exemplo 28:** Também podemos utilizar subconsultas para realizar pesquisas de campos relacionados dentro da mesma tabela. O exemplo seguinte pretende-se saber quem é o responsável/manager de cada trabalhador registado na tabela “Employees”.

```
USE Prodados
Select EmployeeID, LastName,
firstName, ReportsTo from Employees
Where EmployeeID IN (Select Distinct
Reportsto from Employees)
```

| EmployeeID | LastName | firstName | ReportsTo |
|------------|----------|-----------|-----------|
| 2          | Fuller   | Andrew    | NULL      |
| 5          | Buchanan | Steven    | 2         |



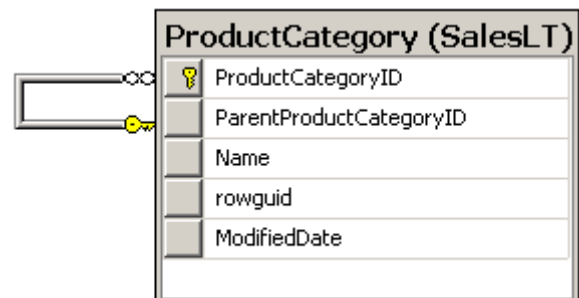
na

USE Prodados

```
Select E1.EmployeeID, E1.LastName, E1.firstName, E1.ReportsTo,
(Select E2.FirstName + ' ' + E2.LastName From Employees as E2
Where E1.ReportsTo = E2.EmployeeID) as 'ReportsTo'
From Employees AS E1
```

|   | EmployeeID | LastName  | firstName | ReportsTo | ReportsTo       |
|---|------------|-----------|-----------|-----------|-----------------|
| 1 | 1          | Davolio   | Nancy     | 2         | Andrew Fuller   |
| 2 | 2          | Fuller    | Andrew    | NULL      | NULL            |
| 3 | 3          | Leverling | Janet     | 2         | Andrew Fuller   |
| 4 | 4          | Peacock   | Margaret  | 2         | Andrew Fuller   |
| 5 | 5          | Buchanan  | Steven    | 2         | Andrew Fuller   |
| 6 | 6          | Suyama    | Michael   | 5         | Steven Buchanan |
| 7 | 7          | King      | Robert    | 5         | Steven Buchanan |
| 8 | 8          | Callahan  | Laura     | 2         | Andrew Fuller   |
| 9 | 9          | Dodsworth | Anne      | 5         | Steven Buchanan |

**Exemplo 29:** Utilize a base de dados Vendas. Na tabela de categorias de produtos, uma categoria faz sempre parte de uma categoria principal (ParentProductCategoryID).



USE Vendas

```
Select P1.Name, (Select P2.Name From
SalesLT.ProductCategory as P2
Where P1.ParentProductcategoryId = P2.ProductCategoryID)
'ParentName' From SalesLT.ProductCategory as P1
```

|    | Name            | ParentName |
|----|-----------------|------------|
| 1  | Bikes           | NULL       |
| 2  | Components      | NULL       |
| 3  | Clothing        | NULL       |
| 4  | Accessories     | NULL       |
| 5  | Mountain Bikes  | Bikes      |
| 6  | Road Bikes      | Bikes      |
| 7  | Touring Bikes   | Bikes      |
| 8  | Handlebars      | Components |
| 9  | Bottom Brackets | Components |
| 10 | Brakes          | Components |

### 3.1.4 Ordenação

(ORDER BY Campo [ASC|DESC], campo [ASC|DESC],...)

#### Exemplo 30

```
Use Exames
Select * from Docentes
Order By Nome
```

#### Exemplo 31

```
Use Exames
Select Área, Nome from Docentes
Order By Área
```

#### Exemplo 32

```
Use Vendas
Select Name, ListPrice from SalesLT.Product
Order By ListPrice DESC
```

#### Exemplo 33

```
Select Color, SIZE, Name from SalesLT.Product
ORDER BY Color DESC, SIZE
```

| Color  | SIZE | Name                         |
|--------|------|------------------------------|
| Yellow | 38   | Road-550-W Yellow, 38        |
| Yellow | 38   | ML Road Frame-W - Yellow, 38 |
| Yellow | 40   | ML Road Frame-W - Yellow, 40 |
| Yellow | 40   | Road-550-W Yellow, 40        |
| Yellow | 40   | Road-350-W Yellow, 40        |
| Yellow | 42   | Road-350-W Yellow, 42        |
| Yellow | 42   | Road-550-W Yellow, 42        |
| Yellow | 42   | ML Road Frame-W - Yellow, 42 |
| Yellow | 44   | ML Road Frame-W - Yellow, 44 |

### 3.1.5 Junção, restrição, projeção e ordenação

#### Verdadeiro/Falso

#### Exemplo 34

```
Use Exames
Select * from Docentes
where IDDocente = IDDocente
```

#### Exemplo 35

```
Use Exames
Select * from Docentes
where IDDocente <> IDDocente
```

## Junção + restrição + projeção

**Exemplo 36:** Qual o resultado da seguinte pesquisa?

Use Exames

```
Select * from Cursos, CursosDisciplinas
```

E agora qual o resultado?

Use Exames

```
Select * from Cursos, CursosDisciplinas
```

```
Where Cursos.IDCurso = CursosDisciplinas.IDCurso
```

**Exemplo 37:** Na base de dados “Exames” quais as disciplinas (Nome da Disciplina) do curso de Engenharia Informática ?

**Exemplo 38 :** Utilize a base de dados “Vendas”. Desenvolva as instruções necessárias para obter o resultado apresentado na tabela seguinte.

|   | ProductModelID | Name             | ProductNumber | Name                          |
|---|----------------|------------------|---------------|-------------------------------|
| 1 | 7              | HL Touring Frame | FR-T98Y-60    | HL Touring Frame - Yellow, 60 |
| 2 | 7              | HL Touring Frame | FR-T98Y-46    | HL Touring Frame - Yellow, 46 |
| 3 | 7              | HL Touring Frame | FR-T98Y-50    | HL Touring Frame - Yellow, 50 |
| 4 | 7              | HL Touring Frame | FR-T98Y-54    | HL Touring Frame - Yellow, 54 |
| 5 | 7              | HL Touring Frame | FR-T98U-46    | HL Touring Frame - Blue, 46   |
| 6 | 7              | HL Touring Frame | FR-T98U-50    | HL Touring Frame - Blue, 50   |
| 7 | 7              | HL Touring Frame | FR-T98U-54    | HL Touring Frame - Blue, 54   |
| 8 | 7              | HL Touring Frame | FR-T98U-60    | HL Touring Frame - Blue, 60   |

### 3.1.6 Campos calculados

**Exemplo 39**

Use vendas

```
Select CustomerID, SubTotal, SubTotal*0.05 as 'Desconto 5%'  
from SalesLT.SalesOrderHeader
```

Resultado da instrução

|   | CustomerID | SubTotal   | Desconto 5% |
|---|------------|------------|-------------|
| 1 | 609        | 880,3484   | 44.017420   |
| 2 | 106        | 78,81      | 3.940500    |
| 3 | 340        | 38418,6895 | 1920,93475  |

**Exemplo 40:** Adicione ao exemplo anterior o primeiro nome do “Customer” de acordo com o “CustomerID” listado?

**Exemplo 41:** Utilize a base de dados “Vendas”. Calcule 2,34% do preço unitário (SalesLT.Product.ListPrice) de cada produto, mas apenas para os produtos com ProductModelID = 22.

**Exemplo 42:** Utilize a base de dados “Vendas”. Considerando que a moeda utilizada é o US DOLAR, pretende-se que faça a sua transformação para a moeda em EUROS<sup>6</sup> para o campo SalesLT.Product.ListPrice.

### 3.1.7 Funções de Datas e Horas

#### Funções que obtêm partes de data e hora

| Função/Sintaxe   |               |
|--|---------------|
| DAY (date) : Retorna um inteiro que representa a parte do dia da date especificada.  |               |
| MONTH (date) : Retorna um inteiro que representa a parte do mês de uma date especificada.  |               |
| YEAR (date): Retorna um inteiro que representa a parte do ano da date especificada.  |               |
| DATEPART (datepart , date)<br>Retorna um inteiro que representa a datepart especificada da date especificada. The weekday (dw) datepart returns a number that corresponds to the day of the week, for example: Sunday = 1, Saturday = 7. The number produced by the weekday datepart depends on the value set by SET DATEFIRST. This sets the first day of the week. |               |
| <i>Datepart</i>  | Abbreviations |
| <i>year</i>  | yy, yyyy      |
| <i>quarter</i>   | qq, q         |
| <i>month</i>   | mm, m         |
| <i>dayofyear</i>   | dy, y         |
| <i>day</i>   | dd, d         |
| <i>week</i>  | wk, ww        |
| <i>weekday</i>   | dw            |
| <i>hour</i>  | hh            |
| <i>minute</i>  | mi, n         |
| <i>second</i>  | ss, s         |
| <i>millisecond</i>   | ms            |
| DATENAME (datepart , date)<br>Retorna uma cadeia de caracteres que representa a datepart especificada da data especificada.  |               |
| SET DATEFORMAT { format   @format_var }<br>Valid parameters are mdy, dmy, ymd, ydm, myd, and dym. The DATEFORMAT ydm is not supported for date, datetime2 and datetimeoffset data types. SET DATEFORMAT overrides the implicit date format setting of SET LANGUAGE.  |               |

#### Exemplo 43

```
SELECT GETDATE() AS 'Data actual'
```

<sup>6</sup> Taxas de câmbios disponível em <http://www.bportugal.pt>

#### Exemplo 44

```
SELECT DATEPART(month, GETDATE()) AS 'Número de Mês'
```

#### Exemplo 45

```
SELECT DATEPART(m, 0), DATEPART(d, 0), DATEPART(yy, 0)
```

#### Exemplo 46

```
SELECT DATENAME(month, GETDATE()) AS 'Mês'
```

|   | Mês     |
|---|---------|
| 1 | October |

#### Exemplo 47

Use Exames

```
Select Data, Year(Data) 'Ano', Month(Data) 'Mês', Day(Data) 'Dia'
from Exames
```

|   | Data                    | Ano  | Mês | Dia |
|---|-------------------------|------|-----|-----|
| 1 | 2005-02-17 00:00:00.000 | 2005 | 2   | 17  |
| 2 | 2005-02-25 00:00:00.000 | 2005 | 2   | 25  |
| 3 | 2005-02-11 00:00:00.000 | 2005 | 2   | 11  |

**Exemplo 48:** Utilize a base de dados “Exames”. Pretende-se saber qual o dia da semana dos exames agendados.

Use Exames

```
Select Data, Day(Data) 'Dia', DatePart(WeekDay, Data) 'Dia da
Semana' from Exames
```

|   | Data                    | Dia | Dia da Semana |
|---|-------------------------|-----|---------------|
| 1 | 2005-02-17 00:00:00.000 | 17  | 5             |
| 2 | 2005-02-25 00:00:00.000 | 25  | 6             |
| 3 | 2005-02-11 00:00:00.000 | 11  | 6             |
| 4 | 2005-02-18 00:00:00.000 | 18  | 6             |

OU

Use Exames

```
Select Data, Day(Data) 'Dia', DateName(WeekDay, Data) 'Dia da
Semana' from Exames
```

|   | Data                    | Dia | Dia da Semana |
|---|-------------------------|-----|---------------|
| 1 | 2005-02-17 00:00:00.000 | 17  | Thursday      |
| 2 | 2005-02-25 00:00:00.000 | 25  | Friday        |
| 3 | 2005-02-11 00:00:00.000 | 11  | Friday        |
| 4 | 2005-02-18 00:00:00.000 | 18  | Friday        |

## Funções que obtêm diferença de data e hora

### Função/Sintaxe

DATEDIFF (*datepart* ,*startdate* , *enddate*)

Retorna o número de limites de *datepart* de data ou hora entre duas datas especificadas.

### Exemplo 49

```
SELECT DATEDIFF(Day, '12-12-2008', '12-12-2009') AS NúmeroDias
```

|   | NúmeroDias |
|---|------------|
| 1 | 365        |

### Exemplo 50

```
USE Vendas
Select DATEDIFF(day, OrderDate, ShipDate) AS NúmeroDias
FROM SalesLT.SalesOrderHeader
```

**Exemplo 51:** Utilize a base de dados “Exames”. Quantos dias foram disponibilizados para os exames de Época Normal – Fevereiro.

```
Use Exames
Select DateDiff (Day, Min(Data), Max(Data)) as DiasExames From
Exames
Where Exames.IDEpoca = 1
```

## Funções que validam valores de data e hora

### Função/Sintaxe

ISDATE (*expression*)

Determina se a expressão de entrada **datetime** ou **smalldatetime** é um valor válido de data ou hora.

### Exemplo 52

```
Select ISDATE ('12-2-2007')
```

|   | Resultado |
|---|-----------|
| 1 | 1         |

### Exemplo 53

```
Select ISDATE ('ESTG') 'Resultado'
```

|   | Resultado |
|---|-----------|
| 1 | 0         |



### 3.1.8 Funções que devolvem valores de caracteres

As funções escalares a seguir executam uma operação em um valor de entrada de cadeia de caracteres e retornam uma cadeia de caracteres ou valor numérico:

|  |
|--|
| Descrição  |
| <a href="#">Função e Sintaxe</a>   |
| Argumentos   |
| Retorna o valor do código ASCII do caracter mais à esquerda de uma expressão de caracteres.<br><a href="#">ASCII ( character_expression )</a>  |
| Converte um código de ASCII <b>int</b> num caracter.<br><a href="#">CHAR ( integer_expression )</a><br>integer_expression - É um número inteiro de 0 a 255. NULL será retornado se a expressão de inteiro não estiver nesse intervalo.   |
| Pesquisa a expression2 na expression1 e retorna sua posição inicial, se for localizada. A pesquisa inicia em start_location.<br><a href="#">CHARINDEX ( expression1 ,expression2 [ , start_location ] )</a> <ul style="list-style-type: none"><li>• expression1 - É uma expressão de caracter que contém a sequência a ser localizada. Expression1 limita-se a 8000 caracteres.</li><li>• expression2 - É uma expressão de caracteres a ser pesquisada.</li><li>• start_location - É um inteiro ou expressão <b>bigint</b> em que a pesquisa inicia. Se start_location não for especificado, for um número negativo ou 0, a pesquisa começará no início da expression2.</li></ul>                                |
| Retorna um valor inteiro que indica a diferença entre os valores SOUNDEX de duas expressões de caracter.<br><a href="#">DIFFERENCE ( character_expression , character_expression )</a> <ul style="list-style-type: none"><li>• character_expression - É uma expressão do tipo <b>char</b> ou <b>varchar</b>. Character_expression também pode ser do tipo <b>text</b>; Contudo somente os primeiros 8.000 bytes são significativos.</li></ul>  |
| Retorna a parte da esquerda de uma cadeia de caracteres com o número de caracteres especificado.<br><a href="#">LEFT ( character_expression , integer_expression )</a> <ul style="list-style-type: none"><li>• character_expression - É uma expressão de dados de caracter ou binários. character_expression pode ser uma constante, variável ou coluna. character_expression pode ser de qualquer tipo de dados, com exceção de <b>text</b> ou <b>ntext</b>, que possam ser implicitamente convertidos em <b>varchar</b> ou <b>nvarchar</b>.</li><li>• integer_expression - É um inteiro positivo que especifica quantos caracteres da character_expression serão retornados.</li></ul>                         |
| Retorna o número de caracteres da uma expressão, excluindo espaços em branco à direita.<br><a href="#">LEN ( string_expression )</a>   |
| Retorna uma expressão de caracteres depois de converter para minúsculas os dados de caracteres em maiúsculas.<br><a href="#">LOWER ( character_expression )</a>  |
| Retorna uma expressão de caracteres depois de remover espaços em branco à esquerda.<br><a href="#">LTRIM ( character_expression )</a>  |
| Retorna o caracter Unicode com o código inteiro especificado, como definido pelo padrão do Unicode.<br><a href="#">NCHAR ( integer_expression )</a> <ul style="list-style-type: none"><li>• integer_expression - É um número inteiro positivo de 0 até 65535. Se for especificado um valor fora deste intervalo, será retornado NULL.</li></ul>  |
| Retorna a posição inicial da primeira ocorrência de um padrão em uma expressão específica, ou zeros se o padrão não for encontrado, em todos os tipos de dados de caracteres e de texto válidos.<br><a href="#">PATINDEX ( '%pattern%' , expression )</a> <ul style="list-style-type: none"><li>• Pattern - É uma cadeia de caracteres literal. Caracteres wildcard podem ser usados; entretanto, o caracter % deve vir antes e seguir pattern (exceto quando se pesquisa o primeiro e último caracteres). Pattern é uma expressão da categoria de tipo de dados de cadeia de caracteres.</li><li>• Expression - É uma expressão, normalmente uma coluna que é pesquisada quanto ao padrão específico.</li></ul> |

Retorna uma cadeia de caracteres Unicode com os delimitadores adicionados para tornar a cadeia de caracteres de entrada um identificador delimitado válido do Microsoft SQL Server.

**QUOTENAME ( 'character\_string' [ , 'quote\_character' ] )**

- 'character\_string' - É uma cadeia de caracteres composta de dados de caracteres Unicode.
- 'quote\_character' - É uma cadeia de um caracter a ser usada como o delimitador. Pode ser uma aspa simples ( ' ), um parêntesis reto esquerdo ou direito ( [ ] ) ou aspas duplas ( " ). Se quote\_character não for especificado, serão usados parêntesis.

Substitui todas as ocorrências de um valor da cadeia de caracteres especificado por outro valor de cadeia de caracteres.

**REPLACE ( string\_expression , string\_pattern , string\_replacement )**

- string\_expression - É a expressão de cadeia de caracteres a ser pesquisada. string\_expression pode ser de um tipo de dados caracteres ou binário.
- string\_pattern - É a subcadeia de caracteres a ser localizada. O string\_pattern pode ser de um tipo de dados caracteres ou binário. O string\_pattern não pode ser uma cadeia de caracteres vazia ( "" ).
- Substituição de string - É a cadeia de caracteres de substituição. string\_replacement pode ser de um tipo de dados caracteres ou binário.

Repete um valor da cadeia de caracteres um número especificado de vezes.

**REPLICATE ( string\_expression , integer\_expression )**

Retorna o inverso de um valor da cadeia de caracteres.

**REVERSE ( string\_expression )**

Retorna a parte da direita de uma cadeia de caracteres com o número de caracteres especificado.

**RIGHT ( character\_expression , integer\_expression )**

- character\_expression - É uma expressão de dados de caracteres ou binários. character\_expression pode ser uma constante, variável ou coluna. character\_expression pode ser de qualquer tipo de dados, com exceção de **text** ou **ntext**, que possam ser implicitamente convertidos em **varchar** ou **nvarchar**.
- integer\_expression - Um inteiro positivo que especifica quantos caracteres de character\_expression serão retornados. Se integer\_expression for negativo, será retornado um erro. Se integer\_expression for do tipo **bigint** e contiver um valor grande, character\_expression deverá ter um tipo de dados grande, como **varchar(max)**.

Retorna uma cadeia de caracteres depois de truncar todos os espaços em branco à direita.

**RTRIM ( character\_expression )**

Retorna uma cadeia de caracteres de espaços repetidos.

**SPACE ( integer\_expression )**

Retorna dados de caracteres convertidos de dados numéricos.

**STR ( float\_expression [ , length [ , decimal ] ] )**

- float\_expression - É uma expressão de tipo de dados numérico aproximado (float) com um ponto decimal.
- Length - É o comprimento total. Isso inclui ponto decimal, sinal, dígitos e espaços. O padrão é 10.
- Decimal - É o número de casas à direita do ponto decimal. Decimal deve ser menor que ou igual a 16. Se decimal for maior que 16, o resultado será truncado com dezasseis casas à direita do ponto decimal.

A função STUFF insere uma cadeia de caracteres em outra cadeia de caracteres. Ela exclui um comprimento especificado de caracteres da primeira cadeia de caracteres na posição inicial e, em seguida, insere a segunda cadeia de caracteres na primeira, na posição inicial.

**STUFF ( character\_expression , start , length , character\_expression )**

- character\_expression - É uma expressão de dados de caracter. character\_expression pode ser uma constante, variável ou coluna de caracteres ou de dados binários.
- Start - É um valor de inteiro que especifica o local para iniciar a exclusão e a inserção. Se start ou length for negativo, uma cadeia de caracteres nula será retornada.
- Length - É um inteiro que especifica o número de caracteres a serem excluídos.

Retorna parte de uma expressão de caracteres, binária, de texto ou de imagem.

**SUBSTRING ( value\_expression ,start\_expression , length\_expression )**

- value\_expression - É uma expressão de **character**, **binary**, **text**, **ntext** ou **image**.
- start\_expression - É um inteiro ou uma expressão **bigint** que especifica onde os caracteres retornados devem iniciar.
- length\_expression - É um inteiro positivo ou uma expressão **bigint** que especifica quantos caracteres da value\_expression são retornados.

Retorna uma expressão de caracter com dados de caracter em minúsculas convertidos em maiúsculas.

**UPPER ( character\_expression )**

## Exemplo 54

USE Vendas

```
Select SalesOrderNumber
, LEN(SalesOrderNumber) 'LEN'
, Left (SalesOrderNumber,2) 'Left'
, Right (SalesOrderNumber,5) 'Right'
, SUBSTRING ( SalesOrderNumber ,3 , 2 ) 'SubString'
, Reverse(SalesOrderNumber) 'Reverse'
, Lower (Left(SalesOrderNumber,2)) 'Lower'
, Replace(SalesOrderNumber,'S','X') 'Replace'
, Replicate (SalesOrderNumber,2) 'Replicate'
, STUFF ( SalesOrderNumber , 3 , 3 , 'HHH') 'Stuff'
, PATINDEX ( '%1%',SalesOrderNumber ) 'PATINDEX'
From SalesLT.SalesOrderHeader
```

## Exemplo 55: Quantos caracteres tem o nome da caixa de correio?

```
Select EmailAddress, PATINDEX ( '%@%', EmailAddress)-1
from SalesLT.Customer
```

## Exemplo 56

```
Select '      texto com espaços' as Texto, LEN('      texto com
espaços') as Tamanho
Select ltrim('      texto com espaços') as Texto, LEN(ltrim('
texto com espaços')) as Tamanho
```

| Texto             | Tamanho |
|-------------------|---------|
| texto com espaços | 21      |

| Texto             | Tamanho |
|-------------------|---------|
| texto com espaços | 17      |

### 3.1.9 Funções de agregação

As funções de agregação atuam sobre uma coluna da tabela e devolvem um só valor. As cinco funções de agregação são:

**COUNT** – devolve o número de valores que estão de acordo com a condição;

**SUM** – devolve a soma dos valores da coluna;

**AVG** – devolve a média de todos os valores da coluna;

**MIN** – devolve o menor valor da coluna;

**MAX** – devolve o maior valor da coluna.

As funções de agregação podem ser usadas como expressões apenas no seguinte:

- A lista de seleção de uma instrução **SELECT** (uma subconsulta ou uma consulta externa).
- Uma cláusula **HAVING**.

Notas: *COUNT*, *MAX* e *MIN* são aplicáveis a todos os tipos de campos, enquanto que *SUM* e *AVG* só são aplicáveis a campos numéricos; Exceto para *COUNT(\*)* todas as funções ignoram *NULLs*. *COUNT(\*)* devolve o número de linhas presentes na tabela. Para eliminar duplicados deve ser adicionado á função o argumento *DISTINCT*.

#### Exemplo 57

```
Use Vendas
Select Max(ListPrice) 'Máximo'
From SalesLT.Product
```

|   | Máximo  |
|---|---------|
| 1 | 3578,27 |

#### Exemplo 58

```
Use Vendas
Select AVG(ListPrice) 'Média'
From SalesLT.Product
```

|   | Média    |
|---|----------|
| 1 | 744,5952 |

#### Exemplo 59

```
Use Vendas
Select Count(SalesOrderID) 'Contagem'
From SalesLT.SalesOrderHeader
```

|   | Contagem |
|---|----------|
| 1 | 32       |

**Exemplo 60:** Pretende-se listar os registos da tabelas de Vendas (SalesOrderheader) em que o valor total da venda (TotalDue) é inferior à média das encomendas realizadas.

```
Use Vendas
Select * from SalesLT.SalesOrderHeader
Where TotalDue < (Select AVG(TotalDue)
From SalesLT.SalesOrderHeader)
```

**Exemplo 61:** Utilize a base de dados “Vendas” e realize as seguintes consultas:

1. Qual a quantidade máxima de um produto(OrderQty) que foi vendida num registo de venda?
2. Qual a quantidade mínima de um produto(OrderQty) que foi vendida num registo de venda?
3. Qual a quantidade média de um produto(OrderQty) que foi vendida num registo de venda?
4. Quantos registos de vendas apresentam uma quantidade vendida por produto inferior à média de produtos vendidos por venda?
5. Qual o registo de vendas que apresenta o maior valor de facturação (TotalDue)?

Use Vendas

```
Select * from SalesLT.SalesOrderHeader
Where TotalDue = (Select Max(TotalDue)
From SalesLT.SalesOrderHeader)
```

6. Qual o nome do cliente (Customer) a que pertence a venda com maior valor de facturação (TotalDue) ?

```
Select * from SalesLT.Customer
Where CustomerID = (Select CustomerID
From SalesLT.SalesOrderHeader
Where TotalDue = (Select Max(TotalDue)
From SalesLT.SalesOrderHeader))
```

### 3.1.10 A cláusula *Group By*

As funções de agregação são muito úteis quando combinadas com a cláusula *Group By*. A cláusula *Group By* agrupa as linhas baseadas nos valores dos atributos especificados.

#### Exemplo 62

Use Exames

```
Select Área, Count(Nome) 'Nº de Docentes' From Docentes
Group By Área
```

|   | Área                  | Nº de Docentes |
|---|-----------------------|----------------|
| 1 | Ciências Empresariais | 31             |
| 2 | Ciências Humanas      | 17             |
| 3 | Design                | 19             |
| 4 | Engenharia            | 54             |

**Exemplo 63:** Utilize a base de dados “Exames”. Pretende-se saber quantas disciplinas estão atribuídas ou fazem parte da estrutura de cada um dos cursos.

Use Exames

```
Select IDCurso, Count(IDDisciplina) 'Nº de Disciplinas'
From CursosDisciplinas
Group By IDCurso
```

|   | IDCurso | Nº de Disciplinas |
|---|---------|-------------------|
| 1 | 1       | 46                |
| 2 | 2       | 37                |
| 3 | 3       | 44                |
| 4 | 4       | 33                |

*Ficamos sem saber qual o Nome do Curso que tem atribuído 37 Disciplinas!*

Use Exames

```
Select Cursos.Curso,
Count(CursosDisciplinas.IDDisciplina) 'Nº de Disciplinas'
From Cursos, CursosDisciplinas
Where Cursos.IDCurso = CursosDisciplinas.IDCurso
Group By Cursos.Curso
```

|   | Curso                       | Nº de Disciplinas |
|---|-----------------------------|-------------------|
| 1 | Assessoria de Administração | 46                |
| 2 | Contabilidade               | 37                |
| 3 | Contabilidade e Auditoria   | 44                |
| 4 | Design de Comunicação       | 33                |

**Exemplo 64:** Utilize a base de dados “Vendas”. Pretende-se saber quantos produtos existem por cada modelo de produto existente.

Resultado a obter:

|   | Name                   | Nº de Produtos |
|---|------------------------|----------------|
| 1 | All-Purpose Bike Stand | 1              |
| 2 | Bike Wash              | 1              |
| 3 | Cable Lock             | 1              |
| 4 | Chain                  | 1              |
| 5 | Classic Vest           | 3              |
| 6 | Cycling Cap            | 1              |

**Exemplo 65:** Na base de dados “Vendas” pretende-se saber quantos produtos existem por cada Categoria.

**Exemplo 66:** Na base de dados “Vendas”. Qual a Categoria de produtos que apresenta o maior número de sub-categorias?

Use Vendas

```
Select PC.ParentProductCategoryID as CategoryID,  
       (Select PP.Name  
        from SalesLT.ProductCategory as PP  
        where PP.ProductCategoryID = PC.ParentProductCategoryID)  
       as CategoryName, Count(PC.ProductCategoryID) as 'Contagem'  
from SalesLT.ProductCategory as PC  
Group by PC.ParentProductCategoryID
```

| CategoryID | CategoryName | Contagem |
|------------|--------------|----------|
| NULL       | NULL         | 4        |
| 1          | Bikes        | 3        |
| 2          | Components   | 14       |
| 3          | Clothing     | 8        |
| 4          | Accessories  | 12       |

### Regras para a cláusula Group By

As colunas selecionadas (no *SELECT*) têm que aparecer na cláusula *Group By*. Note-se que o inverso é válido, ou seja, uma coluna pode estar no *Group By* e não ser selecionada (i.e., não aparecer na parte do *Select*).

As condições *WHERE*, se existirem, são aplicadas antes do *Group By* e das funções de agregação serem calculadas.

#### 3.1.11 A cláusula Having

A cláusula *Having* é usada para fazer restrições ao nível dos grupos. É aplicada após a cláusula *Group By* e as funções de agregação serem calculadas.

**Exemplo 67:** Considerando o exemplo anterior. Quais os cursos que apresentam na sua estrutura mais que 40 disciplinas?

Use Exames

```
Select IDCurso, Count(IDDisciplina) 'Nº de Disciplinas' from  
CursosDisciplinas  
Group By IDCurso  
Having Count(IDDisciplina) > 40
```

**Exemplo 68:** Utilize a base de dados “Vendas”. Quantos registos de vendas (SalesOrderDetail) podemos contar em que foram vendidos produtos com um preço de venda superior a 1400, agrupados por vendas e em que a venda tem mais que três produtos encomendados independentemente da quantidade vendida?

```

Use Vendas
Select SalesOrderID, Count(SalesOrderDetailID) 'Contagem'
from SalesLT.SalesOrderDetail
Where SalesLT.SalesOrderDetail.UnitPrice > 1400
Group By SalesOrderID
Having Count(SalesOrderDetailID) >= 3

```

|   | SalesOrderID | Contagem |
|---|--------------|----------|
| 1 | 71782        | 7        |
| 2 | 71784        | 5        |
| 3 | 71796        | 5        |
| 4 | 71797        | 3        |
| 5 | 71858        | 3        |
| 6 | 71897        | 3        |
| 7 | 71898        | 7        |
| 8 | 71938        | 5        |

Notas finais: Não se podem usar funções de agregação na cláusula *Where*; A cláusula *Where* só pode ser aplicada a registos individuais; a cláusula *Having* só pode ser aplicada a grupos.

### 3.1.12 SELECT ... CASE

Dentro de uma instrução SELECT, a função de procura CASE permite que os valores a apresentar sejam substituídos com base em valores de comparação.

**Exemplo 69:** O exemplo seguinte permite decidir com base no campo “Size”

```

SELECT ProductID, Name, Size =
    CASE Size
        WHEN 'S' THEN 'Pequeno'
        WHEN 'M' THEN 'Médio'
        WHEN 'L' THEN 'Grande'
        ELSE Size
    END
FROM SalesLT.Product

```

| ProductID | Name                       | Size    |
|-----------|----------------------------|---------|
| 680       | HL Road Frame - Black, 58  | 58      |
| 706       | HL Road Frame - Red, 58    | 58      |
| 707       | Sport-100 Helmet, Red      | NULL    |
| 708       | Sport-100 Helmet, Black    | NULL    |
| 709       | Mountain Bike Socks, M     | Médio   |
| 710       | Mountain Bike Socks, L     | Grande  |
| 711       | Sport-100 Helmet, Blue     | NULL    |
| 712       | AWC Logo Cap               | NULL    |
| 713       | Long-Sleeve Logo Jersey, S | Pequeno |
| 714       | Long-Sleeve Logo Jersey, M | Médio   |



**Exemplo 70:** O exemplo seguinte permite com base no campo “*ListPrice*” realizar uma análise por intervalos

```
SELECT ProductID, Name, 'Nível' =  
CASE  
    WHEN ListPrice = 0 THEN 'Nível 0'  
    WHEN ListPrice < 50 THEN 'Nível 1'  
    WHEN ListPrice >= 50 and ListPrice < 250 THEN 'Nível 2'  
    WHEN ListPrice >= 250 and ListPrice < 1000 THEN 'Nível 3'  
    ELSE 'Nível 4'  
END  
FROM SalesLT.Product
```

| ProductID | Name                      | Nível   |
|-----------|---------------------------|---------|
| 680       | HL Road Frame - Black, 58 | Nível 4 |
| 706       | HL Road Frame - Red, 58   | Nível 4 |
| 707       | Sport-100 Helmet, Red     | Nível 1 |
| 708       | Sport-100 Helmet, Black   | Nível 1 |
| 709       | Mountain Bike Socks, M    | Nível 1 |
| 710       | Mountain Bike Socks, L    | Nível 1 |
| 711       | Sport-100 Helmet, Blue    | Nível 1 |

### 3.1.13 SQL – Join (variantes)

A ligação entre tabelas é frequentemente designada por *join*.

#### Cross join

O *produto cartesiano* de duas tabelas associa a cada linha da primeira tabela o conjunto das linhas da segunda tabela.

#### Exemplo 71

```
Use Exames  
Select * From Cursos, CursosDisciplinas  
Resultado - (8162 row(s) affected)
```

#### Equi-join

#### Exemplo 72

```
Use Exames  
Select * From Cursos, CursosDisciplinas  
Where Cursos.IDCurso = CursosDisciplinas.IDCurso  
Resultado - (583 row(s) affected)
```

A ligação entre as tabelas (via cláusula *Where*) é realizada através da igualdade entre duas colunas (*equi-join*). Na *equi-join* todas as colunas (de todas as tabelas) são apresentadas e a ligação entre as tabelas faz-se através de uma igualdade (nota: aparecem colunas com conteúdo igual.). Só os registos que satisfazem a condição de junção aparecem no resultado.

## Natural join

Na *junção natural* não são apresentadas colunas (com conteúdo) repetidas.

### Exemplo 73

```
Use Vendas
Select Name, Sum(OrderQty) as Total from SalesLT.Product,
SalesLT.SalesOrderDetail
Where SalesLT.Product.ProductID= SalesLT.SalesOrderDetail.ProductID
Group By Name
Order By Total DESC
```

## Inner join

Numa *inner join* só são apresentadas as linhas em que exista ligação entre as tabelas.

**Exemplo 74:** Antes de realizar o *INNER Join*, verifique quantas Categorias de produtos existem.

```
Use Vendas
Select * From SalesLT.Productcategory
Resultado - (41 row(s) affected)
```

**Exemplo 75:** Questão – Será que todas as categorias tem produtos registados, ou seja existem produtos em todas as categorias?

```
Use Vendas
Select Distinct SalesLT.ProductCategory.* From SalesLT.Product,
SalesLT.Productcategory
Where SalesLT.ProductCategory.ProductCategoryID =
SalesLT.Product.ProductCategoryID
Resultado - (37 row(s) affected)
```

Ou:

```
Use Vendas
Select Distinct SalesLT.ProductCategory.* From SalesLT.Product
INNER JOIN SalesLT.Productcategory
ON SalesLT.ProductCategory.ProductCategoryID =
SalesLT.Product.ProductCategoryID
```

**Exemplo 76:** Utilize a base de dados “Vendas”. Pretende-se associar os clientes com as respetivas vendas.

```
USE vendas
SELECT FirstName, SalesOrderNumber, OrderDate, TotalDue
FROM SalesLT.Customer AS S
JOIN SalesLT.SalesOrderHeader AS SO
ON S.CustomerID = SO.CustomerID
ORDER BY FirstName, OrderDate
```

**Exemplo 77:** Qual o resultado da seguinte instrução ?

```
Use Vendas
SELECT C.FirstName, C.MiddleName, C.LastName, C.CompanyName,
A.AddressLine1, A.AddressLine2, A.City, A.StateProvince,
A.CountryRegion, A.PostalCode
FROM SalesLT.Customer AS C
JOIN SalesLT.CustomerAddress AS CA ON C.CustomerID = CA.CustomerID
JOIN SalesLT.Address AS A ON CA.AddressID = A.AddressID
WHERE C.FirstName = N'Mary'
```

**Exemplo 78:** Decreve o resultado obtido com a seguinte instrução

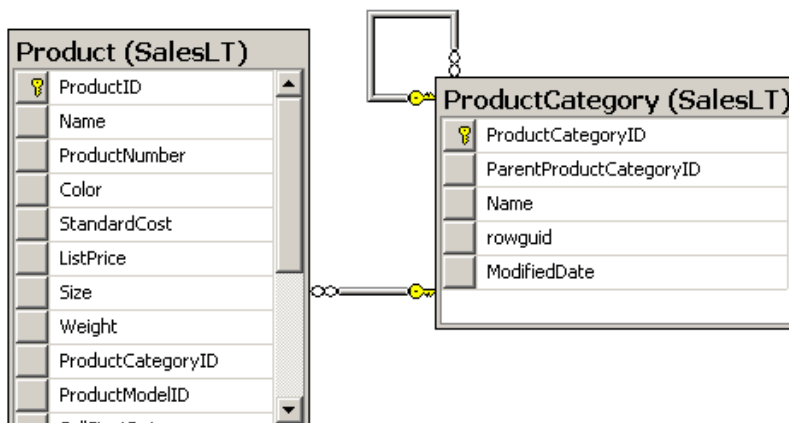
```
Use Vendas
SELECT C.LastName, Sum(SOH.TotalDue)
FROM SalesLT.Customer AS C
JOIN SalesLT.SalesOrderHeader AS SOH ON C.CustomerID =
SOH.CustomerID
WHERE FirstName = N'Walter'
GROUP BY C.LastName
```

## Outer join (em SQL Server)

A *outer join* permite obter a totalidade das linhas de uma tabela, mesmo quando não existe o correspondente valor na outra tabela. A *junção externa* pode ser realizada apenas à esquerda ou à direita.

- **Left join**

Quando a junção externa é realizada à esquerda, aparecem todas as linhas da tabela na esquerda e apenas as linhas correspondentes da tabela na direita.



```
Use Vendas
Select SalesLT.Product.Name, SalesLT.Product.Size,
SalesLT.Productcategory.Name From SalesLT.Product LEFT JOIN
SalesLT.Productcategory
ON SalesLT.ProductCategory.ProductCategoryID =
SalesLT.Product.ProductCategoryID
Resulado - (295 row(s) affected)
```

- **Right join**

Quando a junção externa é realizada à direita, aparecem todas as linhas da tabela na direita e apenas as linhas correspondentes da tabela na esquerda.

Use Vendas

```
Select SalesLT.Product.Name, SalesLT.Product.Size,  
SalesLT.Productcategory.Name From SalesLT.Product RIGHT JOIN  
SalesLT.Productcategory  
ON SalesLT.ProductCategory.ProductCategoryID =  
SalesLT.Product.ProductCategoryID
```

*Resultado - (299 row(s) affected)*

|     | Name              | Size | Name        |
|-----|-------------------|------|-------------|
| 295 | Women's Tights, S | S    | Tights      |
| 296 | NULL              | NULL | Clothing    |
| 297 | NULL              | NULL | Bikes       |
| 298 | NULL              | NULL | Accessories |
| 299 | NULL              | NULL | Components  |

- **Full join**

Combina as duas junções anteriores...

Use Vendas

```
Select SalesLT.Product.Name, SalesLT.Product.Size,  
SalesLT.Productcategory.Name  
From SalesLT.Product  
FULL JOIN SalesLT.Productcategory  
ON SalesLT.ProductCategory.ProductCategoryID =  
SalesLT.Product.ProductCategoryID
```

*Resultado - (299 row(s) affected)*

**Exemplo 79:** Utilize a base de dados “Vendas”. Pretende-se listar os produtos por modelo e categoria. Para incluir todos os produtos, mesmo os não categorizados pode-se utilizar o *full join*.

USE Vendas

```
SELECT PC.Name AS Category, PM.Name AS Model, P.Name AS Product  
FROM SalesLT.Product AS P  
JOIN SalesLT.ProductModel AS PM  
ON PM.ProductModelID = P.ProductModelID  
FULL JOIN SalesLT.ProductCategory AS PC  
ON PC.ProductCategoryID = P.ProductCategoryID  
ORDER BY PC.Name
```

### 3.1.14 Operações sobre conjuntos

#### Union

“SQL provides three set-manipulation constructs that extend the basic query form presented earlier. *Since the answer to a query is a multiset of rows*, it is natural to consider the use of operations such as union, intersection, and difference. SQL supports these operations under the names UNION, INTERSECT, and EXCEPT”. Ramakrishnan (2000), pp 129.

**Exemplo 8o:** Considere a seguinte query: *Encontre o nome dos modelos de produtos que apresentam produtos de cor azul ou cinzento.*

```
Use Vendas
Select PM.Name
From SalesLT.ProductModel PM, SalesLT.Product P
Where PM.ProductModelID = P.ProductModelID
And (P.Color = 'Blue' or P.Color = 'Silver')
Resultado - (62 row(s) affected)
```

Mas se a query for alterada para: *encontre o nome dos modelos de produtos que apresentam produtos de cor azul e de cor cinzenta.*

```
Use Vendas
Select PM.Name, P.Name, P.Color
From SalesLT.ProductModel PM, SalesLT.Product P
Where PM.ProductModelID = P.ProductModelID
And (P.Color = 'Blue' and P.Color = 'Silver')
A sintaxe da instrução está incorrecta pois o resultado obtido é (0 row(s) affected)
```

Utilizando a operação de união sobre dois conjuntos:

```
Use Vendas
Select PM.Name From SalesLT.ProductModel PM, SalesLT.Product P1
Where PM.ProductModelID = P1.ProductModelID
And P1.Color = 'Blue'
UNION
Select PM.Name From SalesLT.ProductModel PM, SalesLT.Product P2
Where PM.ProductModelID = P2.ProductModelID
And P2.Color = 'Silver'
```

Resultado - (20 row(s) affected). Ou seja existem 20 Modelos de produtos com produtos de cor azul e de cor cinzenta.

## Intersect

**Exemplo 81:** Pretende-se saber os exames agendados para a Época Normal (IDEpoca = 1) para a sala com IDSala = 17, mas apenas os que vão decorrer às 13.30h

```
Use Exames
Select * from ExamesSalas, Exames
Where ExamesSalas.IDExame = Exames.IDExame
And ExamesSalas.IDSala = 17 And Exames.IDEpoca = 1
INTERSECT
Select * from ExamesSalas, Exames
Where ExamesSalas.IDExame = Exames.IDExame
And Exames.Hora = '13:30'
```

## Except

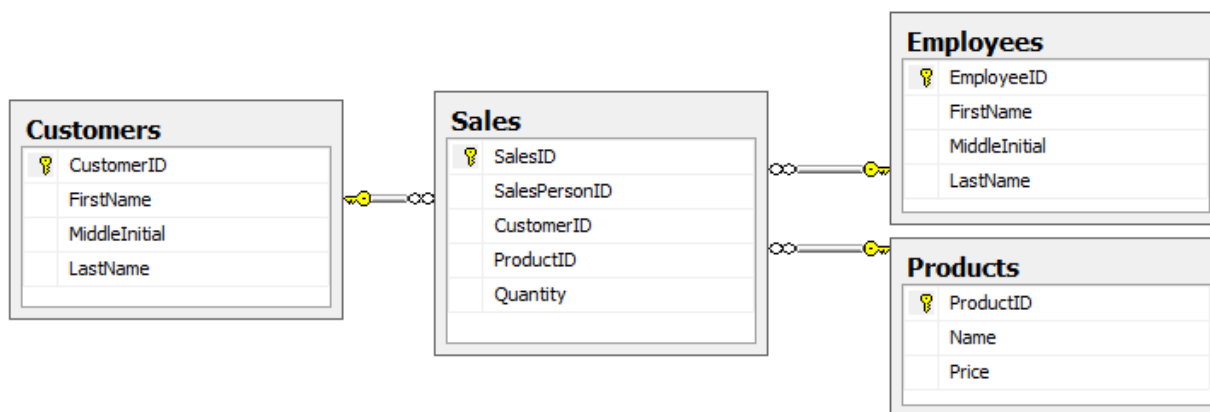
**Exemplo 82:** Pretende-se saber os exames agendados para a Época Normal (IDEpoca = 1) para a sala com IDSala = 17, EXCEPTO os que vão decorrer às 13.30h

```
Use Exames
Select * from ExamesSalas, Exames
Where ExamesSalas.IDExame = Exames.IDExame
And ExamesSalas.IDSala = 17 and Exames.IDEpoca = 1
EXCEPT
Select * from ExamesSalas, Exames
Where ExamesSalas.IDExame = Exames.IDExame
And Exames.Hora = '13:30'
```

### 3.1.15 Exercícios de SQL\_DML

#### Ex.DML.1

Pretende-se que desenvolva as instruções Structured Query Language (SQL) que permitam responder aos requisitos das questões apresentadas. De suporte ao desenvolvimento do trabalho, tem disponível no PAE uma base de dados com a denominação “SalesDB”. O diagrama de relações desta base de dados encontra-se na imagem seguinte.



#### Questões:

- #Q01 – Quantos produtos dispõe a empresa para venda?
- #Q02 – Qual a média dos preços dos produtos com preço superior a zero?
- #Q03 – Dos produtos existentes, quantos a empresa já vendeu?
- #Q04 – Quantos produtos foram adquiridos pelo cliente número 13700 e qual a quantidade total encomendada?
- #Q05 – Quantos clientes não usam MiddleInitial?
- #Q06 – Dos 1198 apelidos utilizados, qual o apelido (LastName) mais utilizado pelos clientes?
- #Q07 – Compare as vendas totais (Price\*Quantity) entre os produtos 10, 341 e 283.
- #Q08 – Qual o melhor cliente em termos de vendas totais (Price\*Quantity)?
- #Q09 – Qual o melhor produto (Name) em termos de vendas totais (Price\*Quantity)?
- #Q10 – Quantos produtos (variedade) é que o cliente número 880 nunca adquiriu?
- #Q11 – Que produtos (ProductID, Name) são comuns nas vendas aos clientes 17200 e ao 6000?
- #Q12 – Qual o cliente que comprou a maior variedade de produtos?
- #Q13 – O campo *Sales.SalesPersonID* indica o empregado associado a cada registo de venda. Apresente a percentagem de vendas realizada por cada empregado (baseando-se no montante total das vendas) de acordo com a imagem seguinte:

| Employee           | sum(Quantity*Price) | Percentagem |
|--------------------|---------------------|-------------|
| Morningstar Greene | 130041172089,61     | 6.96        |
| Abraham Bennet     | 126540241422,845    | 6.77        |
| Heather McBadden   | 122469584454,93     | 6.55        |
| Marjorie Green     | 117922472600,63     | 6.31        |
| Stearns MacFeather | 113322948968,88     | 6.06        |

#Q14 – Apresente um relatório de vendas para o cliente com o ID 6000 de acordo com a imagem seguinte:

| CustomerID | FirstName | LastName | Nº Vendas | €             | Nº Produtos |
|------------|-----------|----------|-----------|---------------|-------------|
| 6000       | Eduardo   | Walker   | 38790     | 5609367645,99 | 9           |

#Q15 – Apresente um relatório de vendas para o produto com o ID 15 de acordo com a imagem seguinte:

| Produto | Quantidade Vendida | Registos de Vendas | Número de Clientes |
|---------|--------------------|--------------------|--------------------|
| Decal 2 | 2108473            | 28417              | 5                  |

#Q16 – Quantos registos de vendas apresentam uma quantidade vendida superior à média de produtos vendidos por venda?

#Q17 – Qual é o melhor produto em termos de quantidade total vendida (Quantity)?

#Q18 – Em média quantos caracteres são utilizados no nome do produto?

#Q19 – Numa consulta à tabela Sales adicione um novo campo com a designação “UnitPrice”. Actualize este campo com base na informação disponível em *Products.Price* para o *Sales.ProductID* em causa.

#Q20 – Numa consulta à tabela Sales adicione novamente o “UnitPrice” e um novo campo com a designação “SubTotal”. Este campo deve ser calculado automaticamente através da fórmula (*Price \* Quantity*)\*1,23.

#Q21 – Numa consulta à tabela Products adicione um novo campo com a designação “IsDiscontinued”. Para os produtos com o valor de *Price* igual a zero, o valor do campo *IsDiscontinued* deve ser “Yes”, caso contrário deve ser “No”.

#Q22 – Numa consulta à tabela Customers adicione um novo campo com a designação “CustomerLevel”. Este campo deve apenas aceitar as opções “Level 1”, “Level 2” ou “Level 3”. Este campo deve ser atualizado de acordo com as vendas totais do *CustomerID* em causa, de acordo com os seguintes intervalos:

Level 1 – vendas entre 0 e 1.000.000

Level 2 – vendas entre 1.000.001 e 100.000.000

Level 3 – vendas > 100.000.000

#Q23 – Pretende-se uma consulta em que todos os campos da tabela *Employees* fiquem em Maiúsculas.

## Ex.DML.2

Considere os dados apresentados referentes a inquéritos relacionados com um processo de estudos de mercado para lançamento de um novo produto eletrónico. Os dados encontram-se disponíveis no ficheiro Excel “**Inqueritos1.xls**”.

Considere a seguinte estrutura de dados:

| Campos             | Descrição   |
|--------------------|---|
| Age                |   |
| MaritalStatus      | [0- Solteiro, 1- Casado]  |
| Income             |   |
| IncomeCategory     |   |
| CarCategory        |   |
| Education          | [1-12ºAno, 2-Bacharelato, 3-Licenciatura, 4-Mestrado, 5-Doutoramento] |
| Emplpy             |   |
| Retired            | [0-Não, 1-Sim]  |
| EmploymentCategory | [1-Técnico de Informática, 2-Professor, 3-Empresário]                 |



|                |                           |
|----------------|---------------------------|
| Gender         | [m-masculino, f-feminino] |
| Wireless       | [o-Não, 1-Sim]            |
| MultipleLines  | [o-Não, 1-Sim]            |
| VoiceMail      | [o-Não, 1-Sim]            |
| Pager          | [o-Não, 1-Sim]            |
| Internet       | [o-Não, 1-Sim]            |
| OwnTV          | [o-Não, 1-Sim]            |
| Own VCR        | [o-Não, 1-Sim]            |
| OwnCDplayer    | [o-Não, 1-Sim]            |
| OwnPDA         | [o-Não, 1-Sim]            |
| OwnComputer    | [o-Não, 1-Sim]            |
| OwnFax         | [o-Não, 1-Sim]            |
| ReadNewspapers | [o-Não, 1-Sim]            |

- Crie uma nova base de dados com a designação “Ex.DML.2”.
- Através do processo de importação, importe os dados do ficheiro de Excel para uma nova tabela de dados.
- Inicie uma nova *Query* que permita obter os dados solicitados nos seguintes quadros:

Quadro 1 - Análise global

|                        |  |
|------------------------|--|
| Nº total de respostas: |  |
| Média de idades:       |  |
| Idade mais nova:       |  |
| Idade mais velha:      |  |

Quadro 2 - Análise específica

|                  |                             |  |
|------------------|-----------------------------|--|
| <b>Solteiros</b> | Número de solteiros:        |  |
|                  | Média de idades:            |  |
| <b>Casados</b>   | Número de casados:          |  |
|                  | Média de idades:            |  |
| <b>Homens</b>    | Número de homens:           |  |
|                  | Média de idades:            |  |
|                  | Idade do homem mais novo:   |  |
|                  | Idade do homem mais velho:  |  |
| <b>Mulheres</b>  | Número de mulheres:         |  |
|                  | Média de idades:            |  |
|                  | Idade da mulher mais nova:  |  |
|                  | Idade da mulher mais velha: |  |

Quadro 3

|          | Solteiros | Casados | Total |
|----------|-----------|---------|-------|
| Homens   |           |         |       |
| Mulheres |           |         |       |
| Total    |           |         |       |

## 3.2 SQL – *Data Definition Language* (DDL)

Como já referido anteriormente, no núcleo de qualquer SGBD está o dicionário de dados, onde se encontram armazenados os *metadados* (descrições acerca dos dados armazenados) assim como toda a informação necessária à gestão da base de dados (quais os utilizadores da base de dados? Quais as suas permissões? etc.).

No caso do modelo relacional o dicionário de dados tem ainda a particularidade de estar armazenado em tabelas e, por isso, pode ser consultado, da mesma forma que os dados, utilizando uma linguagem como o SQL. Contudo enquanto os dados podem ser atualizados através de instruções SQL específicas para manipulação de dados, a atualização do dicionário de dados faz-se utilizando instruções específicas para definição de dados (DDL).

Em relação à definição de dados, uma das instruções fundamentais é a instrução de criação de tabelas. É com esta instrução que se define o esquema de cada tabela (quais os seus atributos, quais os respetivos domínios) e se especificam algumas das restrições de integridade que condicionam a sua manipulação.

A especificação do domínio de cada atributo corresponde a associar a cada atributo o tipo de dados que irá caracterizar as suas instâncias. Ex. SMALLINT, INTEGER, DATE, CHAR, DECIMAL, etc.

### 3.2.1 Create Database

Cria uma nova base de dados assim como os ficheiros usados para armazená-la.

Sintaxe;

```
CREATE DATABASE database_name7
[ ON8
    [ PRIMARY9 ] [ <filespec> [ ,...n ]
    [ , <filegroup> [ ,...n ] ]
[ LOG ON10 { <filespec> [ ,...n ] } ]
]
[ COLLATE collation_name ]
[ WITH <external_access_option> ]
]
```

#### Criar uma base de dados sem especificar ficheiros

O exemplo a seguir cria uma base de dados *BDTeste* e os ficheiros primário e de log de transações correspondentes. Como a instrução não tem nenhum item <filespec>, o ficheiro da base de dados primário é do tamanho do ficheiro primário da base de dados **modelo**<sup>11</sup>. No SQL Server 2005, o log de transações é definido como o maior destes valores: 512 KB ou

---

<sup>7</sup> Is the name of the new database. Database names must be unique within an instance of SQL Server and comply with the rules for identifiers.

<sup>8</sup> Specifies that the disk files used to store the data sections of the database, data files, are explicitly defined.

<sup>9</sup> Specifies that the associated <filespec> list defines the primary file. The first file specified in the <filespec> entry in the primary filegroup becomes the primary file. A database can have only one primary file.

<sup>10</sup> Specifies that the disk files used to store the database log, log files, are explicitly defined. LOG ON is followed by a comma-separated list of <filespec> items that define the log files. If LOG ON is not specified, one log file is automatically created that has a size that is 25 percent of the sum of the sizes of all the data files for the database or 512 KB, whichever is larger. LOG ON cannot be specified on a database snapshot.

<sup>11</sup> The SQL Server 2005 Database Engine uses a copy of the model database to initialize the database and its metadata.

25% do tamanho do ficheiro de dados primário, mas esse valor foi alterado nas versões mais recentes deste SGBD. Como MAXSIZE não é especificado, os ficheiros podem crescer até encher todo o espaço em disco disponível.

```
USE master;
CREATE DATABASE BDTeste;
-- Verificar o tamanho e o nome da Base de Dados
SELECT Name, Size, Size*1.0/128 AS [Size in Mbs]
FROM sys.master_files WHERE name = N'BDTeste';
```

|   | Name    | Size | Size in MBs |
|---|---------|------|-------------|
| 1 | BDTeste | 152  | 1.187500    |

## Criar uma base de dados que especifica os ficheiros de dados e de log de transações

O exemplo seguinte cria uma base de dados “BDTeste2”. Como a palavra-chave PRIMARY não é usada, o primeiro ficheiro (BDTeste2\_dat) torna-se o ficheiro primário.

```
USE master;
-- Obter o caminho das bases de dados do SQL Server
DECLARE @data_path nvarchar(256);
SET @data_path = (SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf',
LOWER(physical_name)) - 1)
FROM master.sys.master_files WHERE database_id = 1 AND file_id = 1);
-- CREATE DATABASE
EXECUTE ('CREATE DATABASE BDTeste2
ON
( NAME = BDTeste2_dat,
  FILENAME = '''+ @data_path + 'BDTeste2dat.mdf'',
  SIZE = 10MB, MAXSIZE = 50MB, FILEGROWTH = 5MB )
LOG ON
( NAME = BDTeste2_log,
  FILENAME = '''+ @data_path + 'BDTeste2log.ldf'',
  SIZE = 5MB, MAXSIZE = 25MB, FILEGROWTH = 5MB )'
);
-- Verificar o tamanho e o nome da Base de Dados
SELECT Name, Size, Size*1.0/128 AS [Size in Mbs]
FROM sys.master_files
WHERE name = N'BDTeste2_dat';
SELECT Name, Size, Size*1.0/128 AS [Size in Mbs]
FROM sys.master_files
WHERE name = N'BDTeste2_log';
```

|   | Name         | Size | Size in MBs |   | Name         | Size | Size in MBs |
|---|--------------|------|-------------|---|--------------|------|-------------|
| 1 | BDTeste2_dat | 1280 | 10.000000   | 1 | BDTeste2_log | 640  | 5.000000    |

### 3.2.2 CREATE TABLE

Sintaxe:

```
CREATE TABLE table_name
    ( { <column_definition> | <computed_column_definition> }
      [ <table_constraint> ] [ ,...n ] )
    [ ON { partition_scheme_name ( partition_column_name ) | filegroup
          | "default" } ]
    [ { TEXTIMAGE_ON { filegroup | "default" } } ]
    [ ; ]
```

+

```
<column_definition> ::=
column_name12 <data_type>
    [ COLLATE collation_name ]
    [ NULL | NOT NULL ] [
        [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed , increment ) ] [ NOT FOR REPLICATION ] ]
    ]
    [ ROWGUIDCOL ] [ <column_constraint> [ ...n ] ]
```

+

```
<column_constraint> ::=
[ CONSTRAINT constraint_name ]
{
    { PRIMARY KEY13 | UNIQUE14 }
        [ CLUSTERED | NONCLUSTERED ]
        [ WITH FILLFACTOR = fillfactor ]
        | WITH ( < index_option > [ , ...n ] ) ]
    [ ON { partition_scheme_name ( partition_column_name )
          | filegroup | "default" } ]
    | [ FOREIGN KEY ]
    REFERENCES [ schema_name. ] referenced_table_name [(ref_column)]
        [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
        [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
        [ NOT FOR REPLICATION ]
    | CHECK15 [ NOT FOR REPLICATION ] ( logical_expression ) }
```

+

```
<computed_column_definition> ::=
column_name AS computed_column_expression16
[ PERSISTED [ NOT NULL ] ]
[
    [ CONSTRAINT constraint_name ]
    { PRIMARY KEY | UNIQUE }
        [ CLUSTERED | NONCLUSTERED ]
        [
            WITH FILLFACTOR = fillfactor
        ]
    ]
```

<sup>12</sup> Is the name of a column in the table.

<sup>13</sup> Is a constraint that enforces entity integrity for a specified column or columns through a unique index. Only one PRIMARY KEY constraint can be created per table.

<sup>14</sup> Is a constraint that provides entity integrity for a specified column or columns through a unique index. A table can have multiple UNIQUE constraints.

<sup>15</sup> Is a constraint that enforces domain integrity by limiting the possible values that can be entered into a column or columns. CHECK constraints on computed columns must also be marked PERSISTED.

<sup>16</sup> Is an expression that defines the value of a computed column. A computed column is a virtual column that is not physically stored in the table, unless the column is marked PERSISTED.

```

        | WITH ( <index_option> [ , ...n ] )
    ]
| [ FOREIGN KEY ]
    REFERENCES referenced_table_name [ ( ref_column ) ]
    [ ON DELETE { NO ACTION | CASCADE } ]
    [ ON UPDATE { NO ACTION } ]
    [ NOT FOR REPLICATION ]
| CHECK [ NOT FOR REPLICATION ] ( logical_expression )
| ON { partition_scheme_name ( partition_column_name )
    | filegroup | "default" } ]
]

```

+

```

< table_constraint > ::= [ CONSTRAINT constraint_name ]
{
    { PRIMARY KEY | UNIQUE }
    [ CLUSTERED | NONCLUSTERED ] ( column [ ASC | DESC ] [ , ...n ] )
    [
        WITH FILLFACTOR = fillfactor
        | WITH ( <index_option> [ , ...n ] )
    ]
    [ ON { partition_scheme_name ( partition_column_name )
        | filegroup | "default" } ]
| FOREIGN KEY ( column [ , ...n ] )
    REFERENCES referenced_table_name [ ( ref_column [ , ...n ] ) ]
    [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
    [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
    [ NOT FOR REPLICATION ]
| CHECK [ NOT FOR REPLICATION ] ( logical_expression )
})

```

Tables are the basic structure where data is stored in the database. Many database tools allow you to create tables without writing SQL, but given that tables are the container of all the data, it is important to include the CREATE TABLE syntax in this paper.

Before we dive into the SQL syntax for **CREATE TABLE**, it is a good idea to understand what goes into a table. Tables are divided into rows and columns. Each row represents one piece of data, and each column can be thought of as representing a component of that piece of data. So, for example, if we have a table for recording customer information, then the columns may include information such as First Name, Last Name, Address, City, Country, Birth Date, and so on. As a result, when we specify a table, we include the column headers and the data types for that particular column.

So what are data types? Typically, data comes in a variety of forms. It could be an integer (such as 1), a real number (such as 0.55), a string (such as 'sql'), a date/time expression (such as '2008-JAN-25 03:22:22'), or even in binary format. When we specify a table, we need to specify the data type associated with each column (i.e., we will specify that 'First Name' is of type char (50) - meaning it is a string with 50 characters). One thing to note is that different relational databases allow for different data types, so it is wise to consult with a database-specific reference first.

So, if we are to create the “Clientes” table specified as above, we would type in

- a. First, we must create de database “MyDataBase”:

```
USE master;
IF DB_ID (N'MyDataBase') IS NOT NULL
/* Se a base de dados existir é eliminada */
DROP DATABASE MyDataBase;
/* Se a base de dados não existir é iniciada */
CREATE DATABASE MyDataBase;
```

b. Now we can create the table

```
CREATE TABLE Clientes
(PrimeiroNome nchar(50),
 UltimoNome nchar(50),
 Endereço nchar(50),
 Cidade nchar(50),
 Pais nchar(25),
 DataNascimento datetime)
-- Verificar a estrutura da tabela
EXEC sp_help Clientes
```

Sometimes, we want to provide a default value for each column. A default value is used when you do not specify a column's value when inserting data into the table. To specify a default value, add "Default [value]" after the data type declaration. In the above example, if we want to default column "Endereço" to "Desconhecido" and City to "Lisboa", we would type in

```
-- 1º eliminar a tabela "Clientes"
DROP TABLE Clientes
-- 2º criar uma nova tabela
CREATE TABLE Clientes
(PrimeiroNome nchar(50),
 UltimoNome nchar(50),
 Endereço nchar(50) default 'Desconhecido',
 Cidade nchar(50) default 'Lisboa',
 Pais nchar(25) default 'Portugal',
 DataNascimento datetime)
```

You can also limit the type of information a table / a column can hold. This is done through the CONSTRAINT keyword, which is discussed next.

### 3.2.2.1 CONSTRAINTS

You can place constraints to limit the type of data that can go into a table. Such constraints can be specified when the table is first created via the CREATE TABLE statement, or after the table is already created via the ALTER TABLE statement.

Common types of constraints include the following:

**NOT NULL, UNIQUE, CHECK, Primary Key, Foreign Key**

Each is described in detail below.

## NOT NULL

By default, a column can hold NULL. If you not want to allow NULL value in a column, you will want to place a constraint on this column specifying that NULL is now not an allowable value.

For example, in the following statement,

```
-- 1º eliminar a tabela "Clientes"
DROP TABLE Clientes
-- 2º criar uma nova tabela
CREATE TABLE Clientes
(IDCliente Integer NOT NULL,
 PrimeiroNome nchar(50) NOT NULL,
 UltimoNome nchar(50),
 Endereço nchar(50) default ' Desconhecido ',
 Cidade nchar(50) default 'Lisboa',
 Pais nchar(25) default 'Portugal',
 DataNascimento datetime)
```

Columns "IDCliente" and "PrimeiroNome" cannot include NULL, while "UltimoNome" can include NULL.

## UNIQUE

*Each UNIQUE constraint generates an index.* The UNIQUE constraint ensures that all values in a column are distinct.

For example, in the following statement,

```
-- 1º eliminar a tabela "Clientes"
DROP TABLE Clientes
-- 2º criar uma nova tabela
CREATE TABLE Clientes
(IDCliente Integer UNIQUE,
 PrimeiroNome nchar(50) NOT NULL,
 UltimoNome nchar(50),
 Endereço nchar(50) default ' Desconhecido ',
 Cidade nchar(50) default 'Lisboa',
 Pais nchar(25) default 'Portugal',
 DataNascimento datetime)
```

Column "IDCliente" cannot include duplicate values, while such constraint does not hold for columns "PrimeiroNome" and "UltimoNome".

Please note that a column that is specified as a primary key must also be unique. At the same time, a column that's unique may or may not be a primary key.

## CHECK

The CHECK constraint ensures that all values in a column satisfy certain conditions.

For example, in the following statement,

```
-- 1º eliminar a tabela "Clientes"
DROP TABLE Clientes
-- 2º criar uma nova tabela
CREATE TABLE Clientes
(IDCliente Integer CHECK (IDCliente > 0),
PrimeiroNome nchar(50) NOT NULL,
UltimoNome nchar(50),
Endereço nchar(50) default 'Desconhecido ',
CodigoPostal nchar(8)
CHECK (CodigoPostal LIKE
'[1-9][0-9][0-9][0-9][-][0-9][0-9][0-9]'),
Cidade nchar(50) default 'Lisboa',
Pais nchar(25) default 'Portugal',
DataNascimento datetime CHECK
(Year(DataNascimento) < year(getdate()))),
Sexo nchar(1) NOT NULL CHECK (Sexo IN ('M', 'F')))
```

### 3.2.2.2 Primary KEY

A primary key is used to uniquely identify each row in a table. It can either be part of the actual record itself, or it can be an artificial field (one that has nothing to do with the actual record). A primary key can consist of one or more fields on a table. When multiple fields are used as a primary key, they are called a composite key.

- A table can contain only one PRIMARY KEY constraint.
- Each PRIMARY KEY generates an index
- All columns defined within a PRIMARY KEY constraint must be defined as NOT NULL. If nullability is not specified, all columns participating in a PRIMARY KEY constraint have their nullability set to NOT NULL.

| Clientes |                |           |                                     |
|----------|----------------|-----------|-------------------------------------|
|          | Column Name    | Data Type | Allow Nulls                         |
| 🔑        | IDCliente      | int       | <input type="checkbox"/>            |
|          | PrimeiroNome   | nchar(50) | <input type="checkbox"/>            |
|          | UltimoNome     | nchar(50) | <input checked="" type="checkbox"/> |
|          | Endereço       | nchar(50) | <input checked="" type="checkbox"/> |
|          | CodigoPostal   | nchar(8)  | <input checked="" type="checkbox"/> |
|          | Cidade         | nchar(50) | <input checked="" type="checkbox"/> |
|          | Pais           | nchar(25) | <input checked="" type="checkbox"/> |
|          | DataNascimento | datetime  | <input checked="" type="checkbox"/> |
|          | Sexo           | nchar(1)  | <input type="checkbox"/>            |
|          |                |           | <input type="checkbox"/>            |

Primary keys can be specified either when the table is created (using CREATE TABLE) or by changing the existing table structure (using ALTER TABLE).



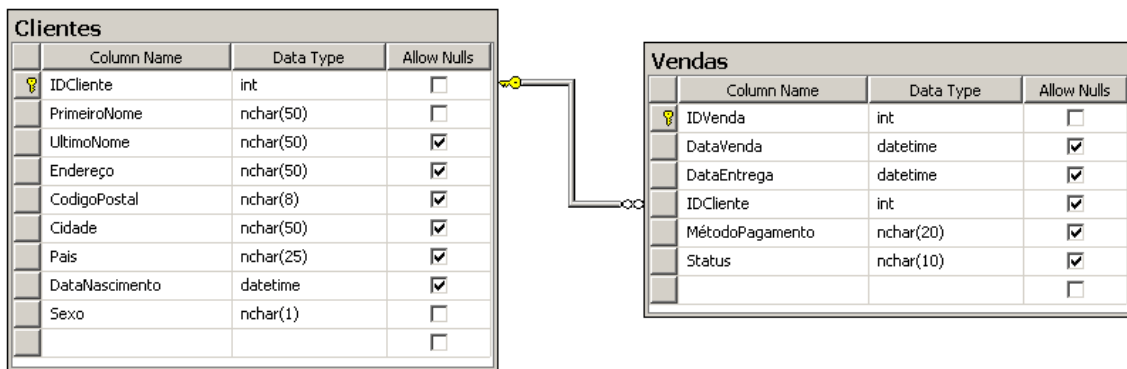
```
-- 1º eliminar a tabela "Clientes"
DROP TABLE Clientes
-- 2º criar uma nova tabela
CREATE TABLE Clientes
(IDCliente Integer PRIMARY KEY,
PrimeiroNome nchar(50) NOT NULL,
UltimoNome nchar(50),
Endereço nchar(50) default 'Unknown',
CodigoPostal nchar(8) CHECK
(CodigoPostal LIKE '[1-9][0-9][0-9][0-9][-][0-9][0-9][0-9]'),
Cidade nchar(50) default 'Lisboa',
Pais nchar(25) default 'Portugal',
DataNascimento datetime CHECK
(Year(DataNascimento)>1950 and
Year(DataNascimento)<year(getdate()))),
Sexo nchar(1) NOT NULL CHECK (Sexo IN ('M','F')))
```

### 3.2.2.3 Foreign KEY

A foreign key is a field (or fields) that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted.

- a. *When a value other than NULL is entered into the column of a FOREIGN KEY constraint, the value must exist in the referenced column; otherwise, a foreign key violation error message is returned.*
- b. *FOREIGN KEY constraints can reference another column in the same table, referred to as a self-reference. However, FOREIGN KEY constraints cannot be used to create a self-referencing or circular FOREIGN KEY constraint.*
- c. *The REFERENCES clause of a column-level FOREIGN KEY constraint can list only one reference column. This must have the same data type as the column on which the constraint is defined.*
- d. *The REFERENCES clause of a table-level FOREIGN KEY constraint must have the same number of reference columns as the number of columns in the constraint column list. The data type of each reference column also must be the same as the corresponding column in the column list.*
- e. *FOREIGN KEY constraints can reference only columns in PRIMARY KEY or UNIQUE constraints in the referenced table. FOREIGN KEY constraints cannot reference unique indexes.*

In the next example, the “IDCliente” column in the *Vendas* table is a foreign key pointing to the “IDCliente” column in the *Clientes* table.



```
CREATE TABLE Vendas
(IDVenda integer Primary KEY,
DataVenda DateTime Default Getdate(),
DataEntrega DateTime,
IDCliente integer,
MétodoPagamento nchar(20) CHECK (MétodoPagamento IN
('Numerário', 'Cheque', 'VISA')),
Status nchar(10) CHECK (Status IN ('Concluída', 'Retida')),
FOREIGN KEY (IDCliente) REFERENCES Clientes (IDCliente),
CONSTRAINT DataEntrega CHECK (DataEntrega >= DataVenda))
```

## Insert information into the tables “Clientes” and “Vendas”

```
USE MyDataBase
-- Registos de Clientes
Insert into Clientes Values (1, 'João', 'Costa', 'Rua de Cima', '7300-111', 'Portalegre', 'Portugal',
'1980-2-23', 'M')
Insert into Clientes Values (2, 'Pedro', 'Silva', 'Rua de Baixo', '7300-112', 'Portalegre', 'Portugal',
'1980-3-5', 'M')
Insert into Clientes Values (3, 'Maria', 'João', 'Avenida J', '7300-113', 'Portalegre', 'Portugal',
'1980-2-6', 'F')
Insert into Clientes Values (4, 'Maria', 'Pereira', 'Desconhecida', '2100-111', 'Lisboa', 'Portugal',
'1970-2-9', 'F')
Insert into Clientes Values (5, 'Carlos', 'Abreu', 'Desconhecida', '2100-121', 'Lisboa', 'Portugal',
'1961-12-6', 'M')
Insert into Clientes Values (6, 'Joaquim', 'Santos', 'Desconhecida', '7300-121', 'Avis', 'Portugal',
'1969-10-1', 'M')
Insert into Clientes Values (7, 'Joana', 'Martins', 'Rua da Pedra', '7300-131', 'Avis', 'Portugal',
'1972-4-4', 'F')
Insert into Clientes Values (8, 'Vicente', 'Louro', 'Desconhecida', '7300-141', 'Nisa', 'Portugal',
'1978-1-1', 'M')
Insert into Clientes Values (9, 'Marisa', 'Batalha', 'Rua do O', '2100-411', 'Lisboa', 'Portugal',
'1980-11-11', 'F')
Insert into Clientes Values (10, 'João', 'Rocha', 'Desconhecida', '7300-221', 'Nisa', 'Portugal',
'1969-10-1', 'M')
Insert into Clientes Values (11, 'João', 'Lopes', 'Pedra Basta', '7300-221', 'Portalegre', 'Portugal',
'1970-10-1', 'M')
Insert into Clientes Values (12, 'Fatima', 'Lopes', 'Rua de Belem', '2150-221', 'Lisboa', 'Portugal',
'1958-10-1', 'F')
Insert into Clientes Values (13, 'Anabela', 'Pereira', 'Vale de Aboim', '5200-221', 'Bragança', 'Portugal',
'1980-5-1', 'F')
Insert into Clientes Values (14, 'Luís', 'Miranda', 'Bairro dos Assentos', '7300-058', 'Portalegre',
'Portugal', '1978-5-1', 'M')
Insert into Clientes Values (15, 'Anabela', 'Sousa', 'Rua da República', '3200-221', 'Aveiro', 'Portugal',
'1980-5-1', 'M')
Insert into Clientes Values (16, 'Tiago', 'Lourenço', 'Praça do soldado', '3200-221', 'Aveiro', 'Portugal',
'1979-5-1', 'M')
Insert into Clientes Values (17, 'Susana', 'Silva', 'Bairro dos Assentos', '7300-789', 'Portalegre',
'Portugal', '1975-6-11', 'F')

-- Registos de Vendas
Insert Into Vendas Values (1201, '2008-11-1', '2008-11-1', 1, 'Numerário', 'Concluída')
Insert Into Vendas Values (1202, '2008-11-2', '2008-11-2', 1, 'Numerário', 'Concluída')
Insert Into Vendas Values (1203, '2008-11-2', '2008-11-2', 1, 'Numerário', 'Concluída')
Insert Into Vendas Values (1204, '2008-11-28', '2008-11-28', 5, 'Numerário', 'Concluída')
Insert Into Vendas Values (1205, '2008-11-28', '2008-11-28', 6, 'Cheque', 'Concluída')
Insert Into Vendas Values (1206, '2008-11-28', '2008-11-28', 7, 'Numerário', 'Concluída')
Insert Into Vendas Values (1207, '2008-11-28', '2008-11-28', 7, 'Numerário', 'Concluída')
Insert Into Vendas Values (1208, '2008-11-28', '2008-11-28', 5, 'Numerário', 'Concluída')
Insert Into Vendas Values (1209, '2008-11-28', '2008-11-28', 8, 'Numerário', 'Concluída')
Insert Into Vendas Values (1210, '2008-11-28', '2008-11-28', 8, 'VISA', 'Concluída')
Insert Into Vendas Values (1211, '2008-11-28', '2008-11-28', 9, 'Numerário', 'Concluída')
Insert Into Vendas Values (1212, '2008-1-28', '2008-1-28', 10, 'Numerário', 'Concluída')
Insert Into Vendas Values (1213, '2008-1-28', '2008-1-28', 10, 'Numerário', 'Concluída')
Insert Into Vendas Values (1214, '2008-11-28', '2008-11-28', 5, 'VISA', 'Retida')
Insert Into Vendas Values (1215, '2008-11-28', '2008-11-28', 1, 'Numerário', 'Concluída')
Insert Into Vendas Values (1216, '2008-11-28', '2008-11-28', 2, 'Numerário', 'Concluída')
Insert Into Vendas Values (1217, '2008-11-28', '2008-11-28', 3, 'VISA', 'Retida')
Insert Into Vendas Values (1218, '2008-11-28', '2008-11-28', 3, 'VISA', 'Concluída')
Insert Into Vendas Values (1219, '2008-11-28', '2008-11-28', 8, 'Numerário', 'Concluída')
Insert Into Vendas Values (1220, '2008-11-28', '2008-11-29', 1, 'Cheque', 'Concluída')
Insert Into Vendas Values (1221, '2008-11-28', '2008-11-29', 4, 'Numerário', 'Concluída')
Insert Into Vendas Values (1222, '2008-11-28', '2008-11-29', 4, 'VISA', 'Concluída')
Insert Into Vendas Values (1223, '2008-11-28', '2008-11-29', 5, 'Numerário', 'Concluída')
Insert Into Vendas Values (1224, '2008-12-2', '2008-12-2', 10, 'Numerário', 'Concluída')
Insert Into Vendas Values (1225, '2008-12-2', '2008-12-2', 11, 'VISA', 'Retida')
Insert Into Vendas Values (1226, '2008-12-2', '2008-12-2', 12, 'VISA', 'Concluída')
Insert Into Vendas Values (1227, '2008-12-2', '2008-12-2', 12, 'Numerário', 'Concluída')
Insert Into Vendas Values (1228, '2008-12-2', '2008-12-2', 13, 'Cheque', 'Concluída')
Insert Into Vendas Values (1229, '2008-12-2', '2008-12-2', 12, 'Numerário', 'Concluída')
Insert Into Vendas Values (1230, '2008-12-2', '2008-12-2', 12, 'VISA', 'Concluída')
Insert Into Vendas Values (1231, '2008-12-2', '2008-12-2', 1, 'Numerário', 'Concluída')
```

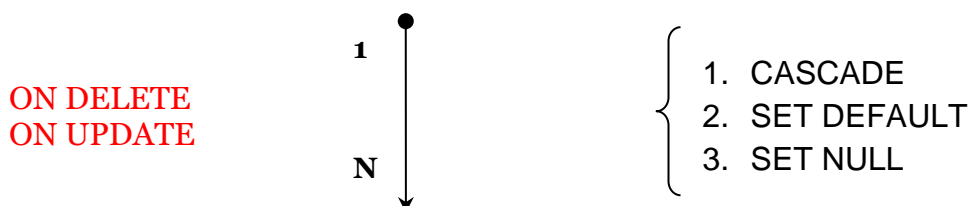
## Manutenção das restrições de Integridade Referencial

Qualquer operação de atualização da base de dados deve respeitar as restrições de integridade. Quando uma atualização viola uma restrição existente pode-se:

- Proibir a atualização (“*Rolling Back*”)
- Ativar ações corretivas (ON DELETE, ON UPDATE)

Considerando o exemplo desenvolvido na alínea anterior é necessário melhorar o modelo de dados para que a integridade dos dados não seja afetada.

|   | IDCliente | PrimeiroNome | UltimoNome | Endereço     | CodigoPostal | Cidade        | Pais        | DataNascimento    | Sexo |
|---|-----------|--------------|------------|--------------|--------------|---------------|-------------|-------------------|------|
| ▶ | 1         | João         | Costa      | Rua de Cima  | 7300-111     | Portalegre... | Portugal... | 23-02-1980 0:0... | M    |
|   | 2         | Pedro        | Silva      | Rua de Baixo | 7300-112     | Portalegre... | Portugal... | 05-03-1980 0:0... | M    |
|   | 3         | Maria        | João       | Avenida J    | 7300-113     | Portalegre... | Portugal... | 06-02-1980 0:0... | F    |
|   | 4         | Maria        | Pereira    | Desconhecida | 2100-111     | Lisboa        | Portugal... | 09-02-1970 0:0... | F    |
|   | 5         | Carlos       | Abreu      | Desconhecida | 2100-121     | Lisboa        | Portugal... | 06-12-1961 0:0... | M    |
|   | 6         | Joaquim      | Santos     | Desconhecida | 7300-121     | Avis          | Portugal... | 01-10-1969 0:0... | M    |
|   | 7         | Joana        | Martins    | Rua da Pedra | 7300-131     | Avis          | Portugal... | 04-04-1972 0:0... | F    |
|   | 8         | Vicente      | Louro      | Desconhecida | 7300-141     | Nisa          | Portugal... | 01-01-1978 0:0... | M    |
|   | 9         | Marisa       | Batalha    | Rua do O     | 2100-411     | Lisboa        | Portugal... | 11-11-1980 0:0... | F    |
|   | 10        | João         | Rocha      | Desconhecida | 7300-221     | Nisa          | Portugal... | 01-10-1969 0:0... | M    |
| * | NULL      | NULL         | NULL       | NULL         | NULL         | NULL          | NULL        | NULL              | NULL |

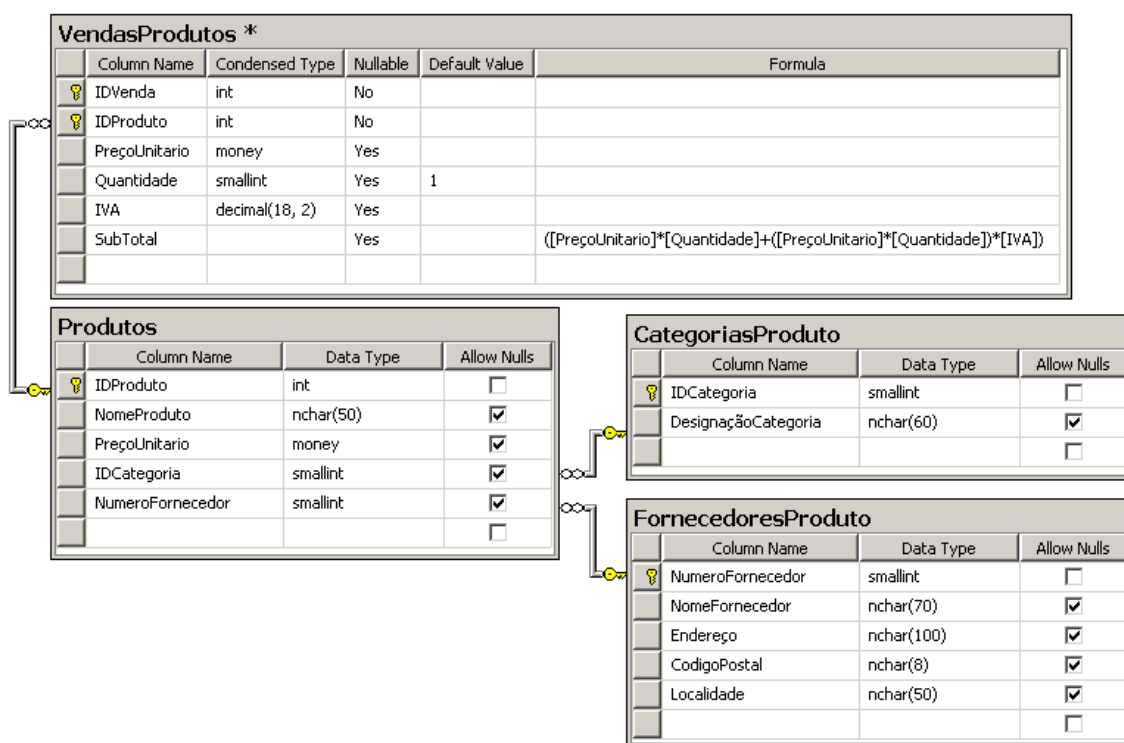


|   | IDVenda | DataVenda         | IDCliente | TotalVenda |
|---|---------|-------------------|-----------|------------|
| ▶ | 1201    | 28-11-2008 0:0... | 1         | 1200,0000  |
|   | 1202    | 28-11-2008 0:0... | 1         | 1500,0000  |
|   | 1203    | 28-11-2008 0:0... | 1         | 100,0000   |
|   | 1204    | 28-11-2008 0:0... | 5         | 1000,0000  |
|   | 1205    | 28-11-2008 0:0... | 6         | 100,0000   |
|   | 1206    | 28-11-2008 0:0... | 7         | 100,0000   |
|   | 1207    | 28-11-2008 0:0... | 7         | 150,0000   |
|   | 1208    | 28-11-2008 0:0... | 5         | 100,0000   |
|   | 1209    | 28-11-2008 0:0... | 8         | 150,0000   |
|   | 1210    | 28-11-2008 0:0... | 8         | 125,0000   |
|   | 1211    | 28-11-2008 0:0... | 9         | 1500,0000  |
|   | 1212    | 28-11-2008 0:0... | 10        | 1500,0000  |
|   | 1213    | 28-11-2008 0:0... | 10        | 1500,0000  |
|   | 1214    | 28-11-2008 0:0... | 5         | 100,0000   |
| * | NULL    | NULL              | NULL      | NULL       |

Neste caso optou-se por: sempre que é eliminado um registo na tabela “Clientes”, todos os registos dependentes deste na tabela “Vendas”, são também eliminados.

```
-- 1º eliminar a tabela "Vendas"
DROP TABLE Vendas;
-- 2º criar uma nova tabela
CREATE TABLE Vendas
(IDVenda integer Primary KEY,
DataVenda DateTime Default Getdate(),
DataEntrega DateTime,
IDCliente integer,
MétodoPagamento nchar(20) CHECK
(MétodoPagamento IN('Numerário', 'Cheque', 'VISA')),
Status nchar(10) CHECK (Status IN ('Concluída', 'Retida')),
FOREIGN KEY (IDCliente) REFERENCES Clientes (IDCliente)
On Delete Cascade,
CONSTRAINT DataEntrega CHECK (DataEntrega >= DataVenda))
```

Realize as alterações propostas no esquema seguinte à Base de Dados “MyDataBase”, promovendo configurações que contemplem a manutenção da integridade dos dados.



Pretende-se registar a seguinte informação:

MyDataBase\Tables\dbo.FornecedoresProduto

| NumeroFornecedor | NomeFornecedor | Endereço        | CodigoPostal | Localidade |
|------------------|----------------|-----------------|--------------|------------|
| 24               | CHIP7          | Rua da Sé       | 2100-234     | Lisboa     |
| 25               | HP Portugal    | Av. de Portugal | 2200-456     | Lisboa     |

MyDataBase\Tables\dbo.CategoriasProduto

| IDCategoria | DesignaçãoCategoria |
|-------------|---------------------|
| 101         | Computadores        |
| 102         | Impressoras         |
| 103         | Consumíveis         |

MyDataBase\Tables\dbo.Produtos

| IDProduto | NomeProduto        | PreçoUnitário | IDCategoria | NumeroFornecedor |
|-----------|--------------------|---------------|-------------|------------------|
| 134       | Notebook HPnx7400  | 789           | 101         | 25               |
| 135       | Impressora HP120   | 125           | 102         | 25               |
| 136       | Rato USB Microsoft | 15            | 103         | 24               |

```
INSERT Into CategoriasProduto (IDCategoria, DesignaçãoCategoria)
Values
(101, 'Computadores'),
(102, 'Impressoras'),
(103, 'Consumíveis')
```

- Na Encomenda 1201 foi encomendada a seguinte relação de produtos: 3 unidades do produto “Notebook HPnx7400”, com o preço unitário de 789€ % e IVA de 23%; 2 unidades do produto “Impressora HP120” com o preço unitário de 125€ e IVA de 23%; uma unidade do produto “Rato USB Microsoft” com o preço unitário de 15 euros e IVA de 23%.
- Na Encomenda 1202 foi encomendada a seguinte relação de produtos: 22 unidades do produto “Rato USB Microsoft”, com o preço unitário de 15 euros e IVA de 23%.
- Na Encomenda 1206 foi encomenda a seguinte relação de produtos: 20 unidades do produto “Notebook HPnx7400”, com o preço unitário de 789€ % e IVA de 23%;

### 3.2.3 Alter Table

Once a table is created in the database, there are many occasions where one may wish to change the structure of the table. Typical cases include the following: Add a column; Drop a column; Change a column name; Change the data type for a column.

Please note that the above is not an exhaustive list. There are other instances where ALTER TABLE is used to change the table structure, such as changing the primary key specification or adding a unique constraint to a column.

Sintaxe:

```
ALTER TABLE table_name
{ ALTER COLUMN17 column_name
  { [type_schema_name.] type_name [(precision [,scale]
    | max | xml_schema_collection ) ]
    [ NULL | NOT NULL ]
    [ COLLATE collation_name ]
    | {ADD | DROP } { ROWGUIDCOL | PERSISTED }
  }
  | [ WITH { CHECK | NOCHECK } ] ADD
  { <column_definition>|<computed_column_definition>
    | <table_constraint>
  } [ ,...n ]
| DROP18 { [CONSTRAINT ] constraint_name
  [WITH (<drop_clustered_constraint_option> [,...n ])]
  | COLUMN column_name
  } [ ,...n ]
  | [ WITH {CHECK|NOCHECK}] { CHECK | NOCHECK } CONSTRAINT
    { ALL | constraint_name [ ,...n ] }
  | { ENABLE | DISABLE } TRIGGER
    { ALL | trigger_name [ ,...n ] }
  | SWITCH [ PARTITION source_partition_number_expression ]
    TO [ schema_name. ] target_table
    [ PARTITION target_partition_number_expression ]
}
```

<sup>17</sup> Specifies that the named column is to be changed or altered. The altered column cannot be any one of the following:

- A column with a **timestamp** data type.
- The ROWGUIDCOL for the table.
- A computed column or used in a computed column.
- Used in an index, unless the column is a **varchar**, **nvarchar**, or **varbinary** data type, the data type is not changed, and the new size is equal to or larger than the old size.
- Used in statistics generated by the CREATE STATISTICS statement. First, remove the statistics using the DROP STATISTICS statement. Statistics that are automatically generated by the query optimizer are automatically dropped by ALTER COLUMN.
- Used in a PRIMARY KEY or [FOREIGN KEY] REFERENCES constraint.
- Used in a CHECK or UNIQUE constraint. However, changing the length of a variable-length column used in a CHECK or UNIQUE constraint is allowed.
- Associated with a default definition. However, the length, precision, or scale of a column can be changed if the data type is not changed.

<sup>18</sup> Specifies that *constraint\_name* or *column\_name* is removed from the table. Multiple columns and constraints can be listed.

Considere para os exemplos seguintes a base de dados *MyDataBase* iniciada.

**Exemplo 1:** É necessário adicionar um novo campo à tabela “Clientes”. Trata-se do campo para registar o Estado Civil do cliente, sendo que este apenas pode conter os seguintes valores: Solteiro, Casado, Viúvo, Divorciado ou União de Facto. Deve apresentar como valor por omissão a opção Solteiro.

```
ALTER table Clientes add EstadoCivil nchar(20) Default  
( 'Solteiro' ) CHECK (EstadoCivil IN  
( 'Solteiro', 'Casado', 'Viúvo', 'Divorciado', 'União de Facto' ))
```

**Exemplo 2:** É necessário alterar a designação do campo “PrimeiroNome” para “PrimeirosNomes”

```
EXEC sp_rename  
'Clientes.PrimeiroNome', 'PrimeirosNomes', 'COLUMN';  
Exec sp_help Clientes
```

**Exemplo 3:** Promova uma alteração ao campo “Endereço” da tabela. Pretende-se que este possa registar um máximo de 255 caracteres *unicode*.

```
ALTER table Clientes Alter Column Endereço nchar(255)
```

**Exemplo 4:** É necessário adicionar dois novos campos à tabela “FornecedoresProduto”. Um para registar o número de Telefone e outro para registar o endereço de correio eletrónico

```
ALTER table FornecedoresProduto add Telefone nchar(10)  
ALTER table FornecedoresProduto add eMail nchar(50)
```

**Exemplo 5:** Verificou-se que o campo “eMail” é desnecessário na tabela “FornecedoresProduto”. Pretende-se que o elimine da estrutura da tabela.

```
ALTER table FornecedoresProduto DROP Column eMail  
EXEC SP_HELP FornecedoresProduto
```

**Exemplo 6:** Pretende-se alterar a tabela *VendasProdutos* adicionando um novo campo - “Desconto” - o qual deve conter o valor de desconto a realizar por produto (ex: 2,1%). Coloque neste campo o valor zero por omissão. Adicione de seguida um campo para registar o valor monetário do Desconto, o qual depende da percentagem de desconto indicada no campo anterior sob o *SubTotal*.

**Exemplo 7:** Na tabela *VendasProdutos* o campo “Quantidade” deve apenas apresentar valores maiores ou iguais a 1 e menores que 100.

**Exemplo 8:** Na tabela *VendasProdutos* deve ser adicionado o campo “IDRegisto”, sendo que este deve ser incrementado em uma unidade sempre que for inserido um novo registo e deve constituir a chave primária da tabela .

### 3.2.4 Drop Table

Sometimes we may decide that we need to get rid of a table in the database for some reason. In fact, it would be problematic if we cannot do so because this could create a maintenance



nightmare for the DBA's. Fortunately, SQL allows us to do it, as we can use the DROP TABLE command. The syntax for DROP TABLE is

Sintaxe:

```
DROP TABLE19 [ database_name . [ schema_name ] . | schema_name . ]  
               table_name [ ,...n ] [ ; ]
```

So, if we wanted to drop the table called “Clientes” that we created in the CREATE TABLE section, we simply type

```
Use MyDataBase  
DROP TABLE Clientes
```

### 3.2.5 Truncate Table

Removes all rows from a table without logging the individual row deletions. TRUNCATE TABLE is similar to the DELETE statement with no WHERE clause. However, TRUNCATE TABLE is faster and uses fewer system and transaction log resources. The syntax for TRUNCATE TABLE is

Sintaxe:

```
TRUNCATE TABLE20  
    [ { database_name . [ schema_name ] . | schema_name . } ]  
    table_name  
[ ; ]
```

So, if we wanted to truncate the table called “Clientes” that we created in SQL CREATE TABLE, we simply type,

```
Use MyDataBase  
TRUNCATE TABLE Clientes
```

### 3.2.6 INSERT

INSERT appends new rows to a table. To replace data in a table, the DELETE or TRUNCATE TABLE statements must be used to clear existing data before loading new data by using INSERT. To modify column values in existing rows, use UPDATE. You can create a new table and load it with data in one step by using the INTO option of the SELECT statement.

---

<sup>19</sup> DROP TABLE cannot be used to drop a table that is referenced by a FOREIGN KEY constraint. The referencing FOREIGN KEY constraint or the referencing table must first be dropped. If both the referencing table and the table that holds the primary key are being dropped in the same DROP TABLE statement, the referencing table must be listed first.

<sup>20</sup> You cannot use TRUNCATE TABLE on tables that:

- Are referenced by a FOREIGN KEY constraint.
- Participate in an indexed view.
- Are published by using transactional replication or merge replication.

When you insert rows, the following rules apply:

- a. If an empty string ( ' ' ) is loaded into a column with a varchar or text data type, the default operation is to load a zero-length string.
- b. If an INSERT statement violates a constraint or rule, or if it has a value incompatible with the data type of the column, the statement fails and the Database Engine displays an error message.
- c. Inserting a null value into a text or image column does not create a valid text pointer, nor does it preallocate an 8-KB text page.
- d. If INSERT is loading multiple rows with SELECT or EXECUTE, any violation of a rule or constraint that occurs from the values being loaded causes the complete statement to be stopped, and no rows are loaded.
- e. When you insert values into tables in a remote instance of the Database Engine, and not all values for all columns are specified, you must identify the columns to which the specified values are to be inserted.

Sintaxe:

```
INSERT
    [ TOP21 ( expression ) [ PERCENT ] ]
    [ INTO]
    { <object22> | rowset_function_limited
      [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
    }
{
    [ ( column_list ) ]
    [ <OUTPUT Clause> ]
    { VALUES ( { DEFAULT23 | NULL | expression24 } [ ,...n ] )
      | derived_table25
      | execute_statement26
    }
}
| DEFAULT VALUES [ ; ]
```

---

<sup>21</sup> Specifies the number or percent of random rows that will be inserted.

<sup>22</sup> Is the name of the table or view that is to receive the data

<sup>23</sup> Forces the Database Engine to load the default value defined for a column.

<sup>24</sup> Is a constant, a variable, or an expression. The expression cannot contain a SELECT or EXECUTE statement.

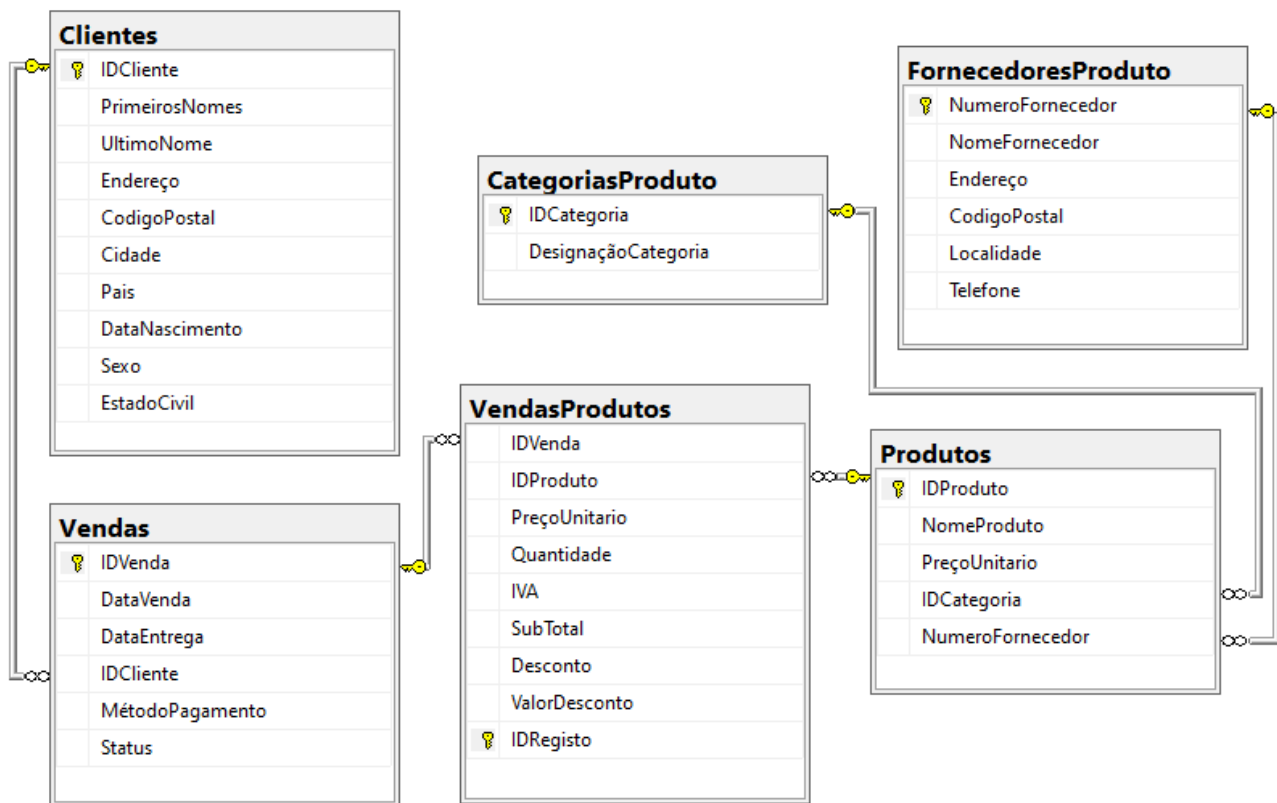
<sup>25</sup> Is any valid SELECT statement that returns rows of data to be loaded into the table. The SELECT statement cannot contain a common table expression (CTE).

<sup>26</sup> Is any valid EXECUTE statement that returns data with SELECT or READTEXT statements. The SELECT statement cannot contain a CTE.

If *execute\_statement* is used with INSERT, each result set must be compatible with the columns in the table or in *column\_list*.

*execute\_statement* can be used to execute stored procedures on the same server or a remote server. The procedure in the remote server is executed, and the result sets are returned to the local server and loaded into the table in the local server.

Considere do modelo de dados da *MyDataBase* desenvolvido até ao momento, para utilização da instrução *Insert*.



#### Exemplo 9: Inserção simples numa tabela

```

INSERT INTO FornecedoresProduto VALUES(26, 'ABC Link',
'Rua dos Franciscanos, 23, RC', '4700-099', 'Braga', '279500600')
INSERT INTO FornecedoresProduto VALUES(27, 'Só Hardware',
'Av dos Álamos, 2001, 8ºD', '3800-352', 'Aveiro', '265400850')
  
```

**Exemplo 10:** Inserção de registos com indicação dos atributos que vão receber os novos valores.

```

INSERT INTO Produtos(IDProduto, NomeProduto, PreçoUnitario,
IDCategoria, NumeroFornecedor)
VALUES(137, 'Notebook DELL', 899, 101, 24)
  
```

**Exemplo 11:** Operação de inserção equivalente à anterior, mas com a ordem dos atributos alterada.

```

INSERT INTO Produtos(NomeProduto, IDProduto, IDCategoria,
NumeroFornecedor, PreçoUnitario)
VALUES('Notebook ASUS', 138, 101, 24, 1250)
  
```

**Exemplo 12:** Inserção de dados numa tabela a partir de uma instrução SELECT

```
-- 1º criar uma nova tabela
CREATE TABLE BackupVendas
(
    IDVenda integer,
    DataVenda Date,
    IDCliente integer,
    TotalVenda Money,
    Primary Key (IDVenda))
-- 2º Copiar os registos
INSERT INTO BackupVendas
SELECT V.IDVenda, V.DataVenda, V.IDCliente,
SUM(VendasProdutos.SubTotal)
FROM Vendas V
JOIN VendasProdutos ON V.IDVenda = VendasProdutos.IDVenda
GROUP BY V.IDVenda, V.DataVenda, V.IDCliente
-- 3º Apresentar os registo da tabela BackupVendas copiados
Select * from BackupVendas
```

**Exemplo 13:** Inicie uma nova tabela com o nome “VendasGlobal” a qual deve registar a seguinte informação: Nome completo do cliente, Cidade, Nome do produto vendido, quantidade do produto vendido e Subtotal da venda. Utilize a instrução SELECT para reunir e registar a informação solicitada, na tabela iniciada.

**Exemplo 14:** atualize a tabela VendasProdutos com os registos seguintes. Copie os registos seguintes para a sua Query.

```
Insert Into VendasProdutos Values (1207, 134, 789, 3, '0.23', '0.05')
Insert Into VendasProdutos Values (1208, 135, 125, 5, '0.23', '0.025')
Insert Into VendasProdutos Values (1209, 135, 125, 5, '0.23', '0.025')
Insert Into VendasProdutos Values (1210, 136, 15, 55, '0.23', '0.001')
Insert Into VendasProdutos Values (1211, 136, 15, 55, '0.23', '0.001')
Insert Into VendasProdutos Values (1211, 134, 789, 3, '0.23', '0.001')
Insert Into VendasProdutos Values (1212, 134, 789, 3, '0.23', '0.05')
Insert Into VendasProdutos Values (1213, 134, 789, 3, '0.23', '0.05')
Insert Into VendasProdutos Values (1213, 136, 15, 3, '0.23', '0.05')
Insert Into VendasProdutos Values (1214, 134, 789, 3, '0.23', '0.05')
Insert Into VendasProdutos Values (1215, 134, 789, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1216, 134, 789, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1217, 134, 789, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1218, 134, 789, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1218, 135, 125, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1219, 134, 789, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1220, 134, 789, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1221, 135, 125, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1222, 135, 125, 2, '0.23', '0.05')
Insert Into VendasProdutos Values (1223, 135, 125, 11, '0.23', '0.05')
Insert Into VendasProdutos Values (1224, 135, 125, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1225, 135, 125, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1226, 135, 125, 10, '0.23', '0.05')
Insert Into VendasProdutos Values (1228, 135, 125, 1, '0.23', '0.05')
Insert Into VendasProdutos Values (1229, 134, 789, 5, '0.23', '0.05')
Insert Into VendasProdutos Values (1230, 134, 789, 5, '0.23', '0.05')
Insert Into VendasProdutos Values (1231, 134, 789, 5, '0.23', '0.05')
```

### 3.2.7 UPDATE

The UPDATE statement can change data values in single rows, groups of rows, or all the rows in a table or view. An UPDATE statement referencing a table or view can change the data in only one base table at a time.

Sintaxe:

#### UPDATE

```
[ TOP ( expression ) [ PERCENT ] ]
{ <object> | rowset_function_limited
  [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
}
SET27 { column_name = { expression | DEFAULT | NULL }
        | { udt_column_name.{ { property_name =expression
        | field_name = expression }
        | method_name28 ( argument [ ,...n ] )
        }}
        | column_name { .WRITE (expression ,@Offset ,@Length)}
        | @variable = expression
        | @variable = column = expression [ ,...n ]
        } [ ,...n ]
[ <OUTPUT Clause> ]
[ FROM { <table_source> } [ ,...n ] ]
[ WHERE { <search_condition>
        | { [ CURRENT OF
            { { [ GLOBAL ] cursor_name }
              | cursor_variable_name
            } ] }
        ]
]
[ OPTION ( <query_hint> [ ,...n ] ) ]
[ ; ]
```

The UPDATE statement has the following major clauses:

- a. SET  
Contains a comma-separated list of the columns to be updated and the new value for each column, in the form *column\_name = expression*. The value supplied by the expressions includes items such as constants, values selected from a column in another table or view, or values calculated by a complex expression.
- b. FROM  
Identifies the tables or views that supply the values for the expressions in the SET clause, and optional join conditions between the source tables or views.
- c. WHERE  
Specifies the search condition that defines the rows from the source tables and views that qualify to provide values to the expressions in the SET clause.

*If an update to a row violates a constraint or rule, violates the NULL setting for the column, or the new value is an incompatible data type, the statement is canceled, an error is returned, and no records are updated. When an UPDATE statement encounters an arithmetic error (overflow, divide by zero, or a domain error) during expression*

---

<sup>27</sup> Specifies the list of column or variable names to be updated.

<sup>28</sup> Is a nonstatic public mutator method of *udt\_column\_name* that takes one or more arguments.

*evaluation, the update is not performed. The rest of the batch is not executed, and an error message is returned.*

**Exemplo 15:** Modificação simples de dados. Crie na tabela “Clientes” um novo campo, para receber informação sobre a categoria do cliente, que pode ser (I) Individual, (EP) Empresa privada, (IP) Instituição Publica. Atribua a todos os registros da tabela a opção (I).

```
Use MyDataBase
Alter Table Clientes ADD CategoriaCliente nchar(2) CHECK
(CategoriaCliente IN ('I','EP','IP'))
go
UPDATE Clientes SET CategoriaCliente = 'I'
Select * from clientes
```

**Exemplo 16:** Modificação simples de dados através de uma expressão de cálculo. Pretende-se que o valor dos “PreçoUnitário” de venda de cada produto seja aumentado 1,23%.

```
UPDATE Produtos SET PreçoUnitario = PreçoUnitario*1.0123
Select * from Produtos
```

**Exemplo 17:** Modificação de dados utilizando a cláusula WHERE.

```
UPDATE Clientes SET EstadoCivil='Solteiro' Where IDCliente=5
Select * from Clientes
```

**Exemplo 18:** Modificação de dados numa tabela utilizando informação proveniente de outra tabela e utilizando a cláusula WHERE.

```
-- 1ª criar uma nova tabela
Create Table RelatorioVendas
( ANO smallint,
  MÊS tinyint,
  TotalVendas bigint,
  Primary Key (ANO,MÊS))
go
-- 2ª criar registos na tabela
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,1)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,2)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,3)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,4)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,5)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,6)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,7)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,8)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,9)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,10)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,11)
INSERT INTO RelatorioVendas (ANO,MÊS)VALUES(2008,12)
go
```

```
-- 3º actualizar o valor total de vendas para 2008
UPDATE RelatorioVendas SET TotalVendas =
    (Select Sum(SubTotal)
     From Vendas V
     JOIN VendasProdutos VP ON V.IDVenda = VP.IDVenda
     Where year(DataVenda) = ANO and month(DataVenda) = MÊS
     Group By Year(DataVenda), month(DataVenda))
```

**Exemplo 19:** Na base de dados “MyDataBase” pretende-se que na tabela “Vendas” altere o ano da data da venda (DataVenda) de 2008 para 2007.

```
Update Vendas Set DataVenda =
    REPLACE(DataVenda, Year(DataVenda), '2007')
Select * from vendas
```

Volte a repor o valor do ano da data da venda em 2008.

**Exemplo 20:** Na base de dados “MyDataBase” pretende-se que na tabela Clientes os registos que apresentam o campo Endereço = “Desconhecida” seja substituída por “Actualizar na próxima venda!”

```
Update Clientes Set Endereço = replace(Endereço,
    'Desconhecida', 'Actualizar na próxima venda!')
```

**Exemplo 21:** Na base de dados “MyDataBase” pretende-se alterar o registo do cliente “João Rocha” para “João Rochas”.

|    | IDCliente | PrimeirosNomes | UltimoNome |
|----|-----------|----------------|------------|
| 4  | 4         | Maria          | Pereira    |
| 5  | 5         | Carlos         | Abreu      |
| 6  | 6         | Joaquim        | Santos     |
| 7  | 7         | Joana          | Martins    |
| 8  | 8         | Vicente        | Louro      |
| 9  | 9         | Marisa         | Batalha    |
| 10 | 10        | João           | Rocha      |
| 11 | 11        | João           | Lopes      |
| 12 | 12        | Estima         | Lopes      |

**Exemplo 22:** Na base de dados “MyDataBase” pretende-se alterar o IDCliente número 8 para 80. Atenção que este cliente pode apresentar registos de Vendas.

|    | IDCliente | PrimeirosNomes | UltimoNome |
|----|-----------|----------------|------------|
| 4  | 4         | Maria          | Pereira    |
| 5  | 5         | Carlos         | Abreu      |
| 6  | 6         | Joaquim        | Santos     |
| 7  | 7         | Joana          | Martins    |
| 8  | 8         | Vicente        | Louro      |
| 9  | 9         | Marisa         | Batalha    |
| 10 | 10        | João           | Rocha      |



**Exemplo 23:** Na base de dados “MyDataBase” pretende-se que adicione um novo fornecedor de produtos com o NumeroFornecedor 99.

|   | NumeroFornecedor | NomeFornecedor | Endereço                     | CodigoPostal | Localidade | Telefone  |
|---|------------------|----------------|------------------------------|--------------|------------|-----------|
| 1 | 24               | CHIP7          | Rua da Sé                    | 2100-234     | Lisboa     | NULL      |
| 2 | 25               | HP Portugal    | Av. de Portugal              | 2200-456     | Lisboa     | NULL      |
| 3 | 26               | ABC Link       | Rua dos Franciscanos, 23, RC | 4700-099     | Braga      | 279500600 |
| 4 | 27               | Só Hardware    | Av dos Álamos, 2001, 8ºD     | 3800-352     | Averio     | 265400850 |
| 5 | 99               | LinkSys        | Av. dos Bombeiros            | 2100-547     | Lisboa     | 215478741 |

De seguida pretende-se que os produtos fornecidos pelo NumeroFornecedor 24, passem a ser todos eles fornecidos pelo NumeroFornecedor 99.

|   | IDProduto | NomeProduto        | PreçoUnitario | IDCategoria | NumeroFornecedor |
|---|-----------|--------------------|---------------|-------------|------------------|
| 1 | 134       | Notebook HPnx7400  | 798,7047      | 101         | 25               |
| 2 | 135       | Impressora HP120   | 126,5375      | 102         | 25               |
| 3 | 136       | Rato USB Microsoft | 15,1845       | 103         | 24               |
| 4 | 137       | Notebook DELL      | 910,0577      | 101         | 24               |
| 5 | 138       | Notebook ASUS      | 1265,375      | 101         | 24               |

**Exemplo 24:** Na base de dados “MyDataBase” pretende-se que altere o Endereço e o Código Postal do Cliente com o IDCliente 7 com os seguinte valores: Vale do Paraíso nº 56, 7300-120.

**Exemplo 25:** Na base de dados “MyDataBase” pretende-se que na tabela Produtos altere o IDProduto 134 para 139. Deve considerar que não se devem perder as vendas já realizadas até ao momento para o referido produto.

**Exemplo 26:** REVISÃO. Qual o total de unidades vendidas por cada categoria de produto?

### 3.2.8 DELETE

The DELETE statement removes one or more rows in a table or view. Any table that has all rows removed remains in the database. The DELETE statement deletes only rows from the table; the table must be removed from the database by using the DROP TABLE statement. The DELETE statement may fail if it violates a trigger or tries to remove a row referenced by data in another table with a FOREIGN KEY constraint. If the DELETE removes multiple rows, and any one of the removed rows violates a trigger or constraint, the statement is canceled, an error is returned, and no rows are removed.

When a DELETE statement encounters an arithmetic error (overflow, divide by zero, or a domain error) occurring during expression evaluation, the Database Engine handles these errors as if SET ARITHABORT is set ON. The rest of the batch is canceled, and an error message is returned.



Sintaxe:

```
DELETE
[ TOP ( expression ) [ PERCENT ] ]
[ FROM ]
{ <object> | rowset_function_limited
  [ WITH ( <table_hint_limited> [ ...n ] ) ]
}
[ <OUTPUT Clause> ]
[ FROM <table_source> [ ,...n ] ]
[ WHERE { <search_condition>
          | { [ CURRENT OF
              { { [ GLOBAL ] cursor_name }
                | cursor_variable_name
              } ] } ]
        ]
[ OPTION ( <Query Hint> [ ,...n ] ) ]
[ ; ]
```

**Exemplo 27:** Na base de dados “MyDataBase” pretende-se eliminar todos os registos da tabela Clientes que pertençam à cidade de “Avis”.

**Exemplo 28:** Na base de dados “MyDataBase” pretende-se eliminar todos os registos da tabela Clientes, da cidade de Nisa mas apenas os que apresentam um valor total de Vendas inferior a 5000€.

**Exemplo 29:** Na base de dados “MyDataBase” pretende-se eliminar o registo do cliente 'Carlos', 'Abreu'.

|   | IDCliente | PrimeirosNomes | UltimoNome | Endereço     | CodigoPostal | Cid |
|---|-----------|----------------|------------|--------------|--------------|-----|
| 4 | 4         | Maria          | Pereira    | Desconhec... | 2100-111     | Lis |
| 5 | 5         | Carlos         | Abreu      | Desconhec... | 2100-121     | Lis |
| 6 | 6         | Joaquim        | Santos     | Desconhec... | 7300-121     | Av  |
| 7 | 7         | Joana          | Martins    | Rua da Pe... | 7300-131     | Av  |

**Exemplo 30:** Na base de dados “MyDataBase” quantos registos têm as tabelas Clientes e Vendas.

**Exemplo 31:** Na base de dados “MyDataBase” pretende-se listar as Vendas realizadas assim como os Produtos que constam nos registos de Vendas, agrupadas por Cliente.

### 3.3 SQL VIEWS/Consultas

Uma consulta (*view*) é uma forma de visualizar os dados de uma tabela ou um conjunto de tabelas relacionadas entre si. Uma consulta pode incidir sobre todos os campos de uma tabela, ou um conjunto de tabelas, ou apenas em relação a alguns campos; pode apresentar todos os registos ou efetuar uma filtragem de modo a apresentar apenas os dados que correspondem a determinada condição.

As consultas facilitam a seleção de dados com o propósito de eliminar redundâncias e retirar os dados em excesso. Assume um formato semelhante ao de uma tabela. Os dados selecionados numa consulta podem ser alterados, apagados ou tratados como se pertencessem a uma tabela. O SQL Server permite formular diferentes tipos de consultas, utilizando diversos métodos e procedimentos:

- a. Consultas de seleção – são as *Views*/consultas mais usadas pela maioria dos utilizadores, com o objetivo de selecionar um conjunto de campos, de uma ou várias tabelas, assim como filtrar um conjunto de registos e criar campos calculados.
- b. Consulta de referência cruzada: com uma CONSULTA DE REFERÊNCIA CRUZADA, podemos sumariar grandes quantidades de informação num formato de fácil leitura. Os dados obtidos surgem da intersecção entre dois campos e são apresentados em linhas e colunas, como uma folha de dados. *Se tivermos uma tabela de produtos e outra tabela de encomendas, podemos visualizar o total de produtos encomendados por ano para um artigo determinado.*
- c. Consulta de ação: permitem realizar alterações aos registos de uma ou mais tabelas. Usa-se uma CONSULTA DE ACÇÃO para criar uma nova tabela, apagar, juntar ou fazer alterações de registos.

#### Razões para utilizar ou aplicar uma *view*/consulta:

- a. Selecionar campos - Podemos incluir numa consulta apenas um conjunto limitado de campos de uma tabela. Numa tabela de Pacientes, a consulta pode mostrar apenas o nome e a morada e ocultar os restantes campos.
- b. Selecionar registos - Podemos especificar um critério na consulta que limite a apresentação dos registos de uma tabela. Por exemplo, podemos apenas selecionar os Pacientes de uma certa cidade.
- c. Ordenar registos - Podemos visualizar os registos de uma tabela com uma certa ordem. Por exemplo, podemos ordenar os nomes de Pacientes por ordem alfabética.
- d. Fazer questões acerca de dados em várias tabelas – Podemos usar uma consulta para responder a uma questão sobre os dados de uma ou mais tabelas e apresentar os resultados numa só folha de dados. Podemos fazer questões sobre os dados de outras bases de dados.
- e. Realizar cálculos - Podemos criar novos campos que contenham resultados de um cálculo, chamados campos calculados. Por exemplo, para encontrar o custo total, podemos criar um campo que some os custos dos tratamentos de cada paciente.
- f. Usar uma consulta como um tipo de dados para formulários, relatórios ou outras consultas - Podemos criar uma consulta de seleção e usá-la como base para a execução de um formulário ou relatório. Usando uma consulta, podemos incluir dados de mais do que uma tabela e selecionar um critério para mostrar apenas um limite de dados.

- g. Fazer alterações de dados em tabelas - Podemos alterar, eliminar ou ligar um grupo de registos, usando CONSULTAS DE ACÇÃO. Podemos criar uma nova tabela que inclua os registos de uma tabela existente ou um grupo de tabelas.
- h. Para personalizar os dados - As views permitem que diferentes utilizadores vejam os dados de forma diferente, mesmo quando eles estiverem a aceder aos mesmos dados e ao mesmo tempo. Isso é especialmente útil quando os utilizadores que têm muitos interesses e níveis de habilidade diferentes partilham a mesma base de dados. Por exemplo, uma view pode ser criada para recuperar apenas os dados para clientes com quem um gerente de conta lida. A view pode determinar quais os dados a serem recuperados com base no ID de login do gerente de conta que usa a view.
- i. Para exportar e importar dados - As Views podem ser utilizadas para exportar dados para outras aplicações como o Microsoft Excel para analisar os dados com recurso a gráficos. O utilitário **bcp** permite importar/exportar dados definidos pela view para um ficheiro externo.
- j. Para combinar dados armazenados em vários servidores - Uma View pode ser construída com base em dados de fontes heterogéneas, como servidores remotos. Por exemplo, para combinar dados de diferentes servidores remotos, cada um dos quais armazena dados para uma região diferente de uma organização, pode ser criada uma view/consulta distribuída que recupere dados de cada fonte de dados e depois crie uma view com base nessas consultas distribuídas.

Ao segmentar dados por múltiplos servidores, as views/consultas que acedem apenas a uma fração de dados podem ser executadas mais rapidamente, porque há menos dados para verificar. Se as tabelas estiverem localizadas em servidores diferentes, ou em um computador que usa multiprocessadores, cada tabela envolvida na consulta também pode ser verificada em paralelo. Isso pode melhorar o desempenho de consulta. Além disso, tarefas de manutenção, como reconstruir índices ou fazer backup de uma tabela, podem ser executadas mais rapidamente.

Ao realizar-se uma consulta através de uma *View*, o mecanismo de base de dados verifica se todos os objetos da base de dados referenciados em algum lugar da instrução existem, se são válidos no contexto da instrução e se as instruções de modificação de dados *não violam nenhuma regra de integridade de dados*. Uma verificação que falhe retorna uma mensagem de erro.

Quando uma view é criada, as informações sobre ela são armazenadas nas seguintes views do catálogo: *sys.views*, *sys.columns* e *sys.sql\_expression\_dependencies*. O texto da instrução *CREATE VIEW* é armazenado na exibição do catálogo *sys.sql\_modules*.

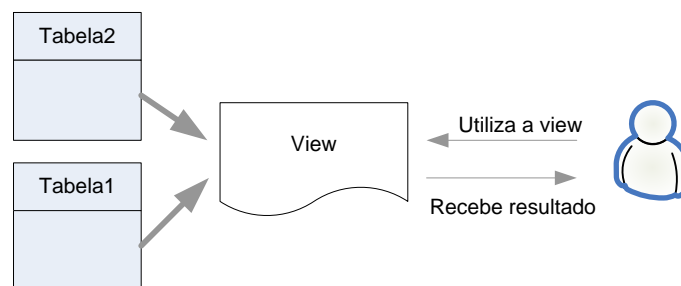
Sintaxe:

```
USE NomeBaseDados
CREATE/ALTER VIEW [ schema_name . ] view_name [ (column [ ,...n ] )
]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement
[ WITH CHECK OPTION29 ] [ ; ]

<view_attribute> ::=
{
    [ ENCRYPTION ]
    [ SCHEMABINDING ]
    [ VIEW_METADATA ] }
```

### 3.3.1 Criar e gerir Views

Utilize a base de dados *MyDataBase*. Pretende-se consultar regularmente o número total de produtos, existente por categoria. Crie uma consulta/view para este efeito.

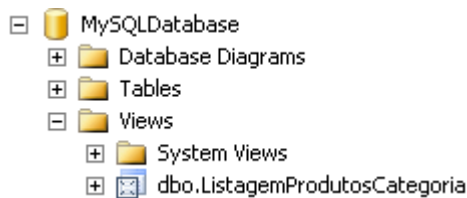


#### Exemplo 32

Pretende-se que na base de dados *MyDataBase* construa uma view/consulta para obter o número total de produtos existentes por categoria.

```
USE MyDataBase
-- Criar a View
go
CREATE VIEW ListagemProdutosCategoria AS
    SELECT C.DesignaçãoCategoria, COUNT(P.IDProduto) 'Total'
    FROM CategoriasProduto C
    JOIN Produtos P on C.IDCategoria = P.IDCategoria
    Group By C.DesignaçãoCategoria
GO
-- Utilizar/Listar a View
SELECT * FROM ListagemProdutosCategoria
```

<sup>29</sup> Forces all data modification statements executed against the view to follow the criteria set within *select\_statement*. When a row is modified through a view, the WITH CHECK OPTION makes sure the data remains visible through the view after the modification is committed. CHECK OPTION cannot be specified if TOP is used anywhere in *select\_statement*.



|   | DesignaçãoCategoria | Total |
|---|---------------------|-------|
| 1 | Computadores        | 3     |
| 2 | Consumíveis         | 1     |
| 3 | Impressoras         | 1     |

Ou

```
Select * from sys.views
```

Ou

```
Select * from sys.objects
Where Type_Desc = 'View'
```

-- Listar as instruções da View

```
EXEC sp_helptext ListagemProdutosCategoria
```

```

-- Criar a View
CREATE VIEW ListagemProdutosCategoria AS
SELECT C.DesignaçãoCategoria, COUNT(P.IDProduto) 'Total'
FROM CategoriasProduto C
JOIN Produtos P on C.IDCategoria = P.IDCategoria
Group By C.DesignaçãoCategoria

```

Aplicação da opção WITH ENCRYPTION. Altere a *view* criada no exemplo anterior e utilize a opção *with encryption* na criação da mesma.

-- Criar a View

GO

```
ALTER VIEW ListagemProdutosCategoria
```

```
WITH ENCRYPTION AS
```

```

SELECT C.DesignaçãoCategoria, COUNT(P.IDProduto) 'Total'
FROM CategoriasProduto C
JOIN Produtos P on C.IDCategoria = P.IDCategoria
Group By C.DesignaçãoCategoria

```

GO

-- Listar as instruções da View

```
EXEC sp_helptext ListagemProdutosCategoria
```

```

The text for object 'ListagemProdutosCategoria' is encrypted.

```

### Exemplo 33

Pretende-se que na base de dados *MyDataBase* construa uma view/consulta para obter o volume total de vendas por cliente. De seguida pretende-se apenas as vendas para os clientes da “Cidade” de Lisboa. Ou seja, é interrogada a View.

```
Use MyDataBase
-- Criar a VIEW
GO
Create View RelatorioClientes AS
    Select C.IDCliente, PrimeirosNomes, UltimoNome, Cidade,
           sum(Subtotal) 'Total'
    From Clientes C
    JOIN Vendas V ON C.IDCliente = V.IDCliente
    JOIN VendasProdutos VP ON V.IDVenda = VP.IDVenda
    Group By C.IDCliente, PrimeirosNomes, UltimoNome, Cidade
GO
-- Utilizar a VIEW
Select * from RelatorioClientes Where Cidade = 'Lisboa'
```

|   | IDCliente | PrimeirosNomes | UltimoNome | Cidade | Total    |
|---|-----------|----------------|------------|--------|----------|
| 1 | 4         | Maria          | Pereira    | Lisboa | 461.25   |
| 2 | 9         | Marisa         | Batalha    | Lisboa | 3926.16  |
| 3 | 12        | Fatima         | Lopes      | Lisboa | 11242.20 |

```
Select * From RelatorioClientes Where Total > 500
Select Cidade, sum(Total) Total From RelatorioClientes
Group By Cidade
```

### Exemplo 34

A *view/consulta* seguinte permita apresentar o total de vendas, o total de descontos efetuados e o peso destes sobre o total de vendas, tudo agrupado pelo Mês da data da venda. Foi atribuído o nome “RelatorioFinanceiro” à view/consulta.

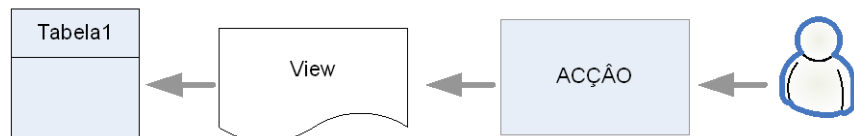
```
Use MyDataBase
-- Criar a VIEW
go
CREATE VIEW RelatorioFinanceiro AS
    SELECT Year(V.DataVenda) 'Ano', Month(V.DataVenda) 'Mês '
    , SUM(VendasProdutos.SubTotal) 'Total Vendas(€)',
    SUM(VendasProdutos.ValorDesconto) 'Total Descontos(€)',
    SUM(VendasProdutos.ValorDesconto)/SUM(VendasProdutos.SubTotal)
    '%' FROM Vendas V INNER JOIN VendasProdutos ON V.IDVenda =
    VendasProdutos.IDVenda
    Group By Year(V.DataVenda), Month(V.DataVenda)
go
```

|   | Ano  | Mês | Total Vendas(€) | Total Descontos(€) | %        |
|---|------|-----|-----------------|--------------------|----------|
| 1 | 2008 | 1   | 5878.17         | 293.91             | 0.050000 |
| 2 | 2008 | 11  | 43482.96        | 743.70             | 0.017103 |
| 3 | 2008 | 12  | 16555.80        | 827.81             | 0.050001 |

### 3.3.2 Views de ação<sup>30</sup>

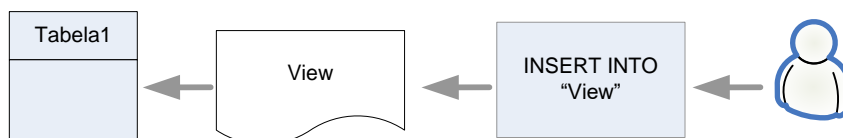
As consultas de ação são aquelas que não devolvem nenhum registo (resultado), tendo por objetivo adicionar, eliminar ou modificar registos de uma ou mais tabelas.

É possível modificar os dados de uma tabela base subjacente através de uma *view*, contanto que as seguintes condições sejam verdadeiras:



- Todas as modificações, inclusive as instruções UPDATE, INSERT e DELETE, devem referenciar colunas de apenas uma tabela base, no entanto a view pode consultar várias tabelas base.
- As colunas a serem modificadas na *view* devem referenciar diretamente os dados subjacentes das colunas da tabela. As colunas não podem ser derivadas de qualquer outro modo, como pelo seguinte:
  - Uma função de agregação: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR e VARP.
  - A coluna não pode ser calculada através de uma expressão que utiliza outras colunas. As colunas formadas com o uso dos operadores de conjunto UNION, UNION ALL, CROSSJOIN, EXCEPT e INTERSECT resultam numa coluna calculada e também não são atualizáveis.
- As colunas modificadas não são afetadas pelas cláusulas GROUP BY, HAVING ou DISTINCT.
- TOP não é usado em nenhum lugar na “*select\_statement*” da *view* junto com a cláusula WITH CHECK OPTION.

**INSERT INTO** – adiciona um registo numa tabela. É conhecida como uma consulta de dados adicionados. Esta consulta pode ser de dois tipos: Inserir um único registo ou inserir numa tabela os registos contidos em outra tabela.



```
INSERT INTO "View" (campo1, campo2, ..., campoN)
VALUES (valor1, valor2, ..., valorN)
```

Esta consulta regista no campo1 o valor1, no campo2 e valor2 e assim, sucessivamente.

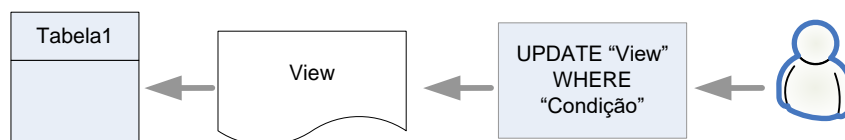
<sup>30</sup> Sintaxes apresentadas e desenvolvidas no ponto “3.2.SQL Data Definition Language (DDL)”

**Exemplo 35:** Pretende-se que na base de dados MyDataBase construa uma view/consulta para adicionar registos à tabela “CategoriasProduto”.

```
USE MyDataBase
-- Verificar quantas categorias existem
Select * From CategoriasProduto
-- Criar a View
GO
CREATE VIEW GerirCategorias AS
    Select IDCategoria, DesignaçãoCategoria
    From CategoriasProduto
GO
-- Registrar valores na View
Insert Into GerirCategorias
    values (104, 'Fotografia Digital'),
           (105, 'GPS'),
           (106, 'Equipamentos de Rede')
GO
-- Verificar quantas categorias existem depois da View
Select * from GerirCategorias
Select * from CategoriasProduto
```

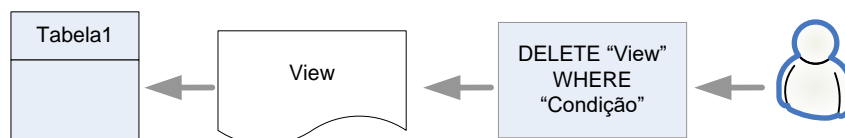
**UPDATE** - Cria uma consulta de atualização que altera os valores dos campos de uma tabela, baseando-se em um critério específico.

UPDATE não gera nenhum resultado. Se numa consulta de atualização suprimimos a cláusula WHERE todos os registos da tabela assinalada serão atualizados.



```
Update GerirCategorias SET DesignaçãoCategoria = 'Livros'
Where IDcategoria = 104
```

**DELETE** - Cria uma consulta de eliminação que elimina os registos de uma tabela listada na cláusula FROM e que satisfaça a cláusula WHERE. Esta consulta elimina os registos completos, não é possível eliminar o conteúdo de algum campo em concreto.

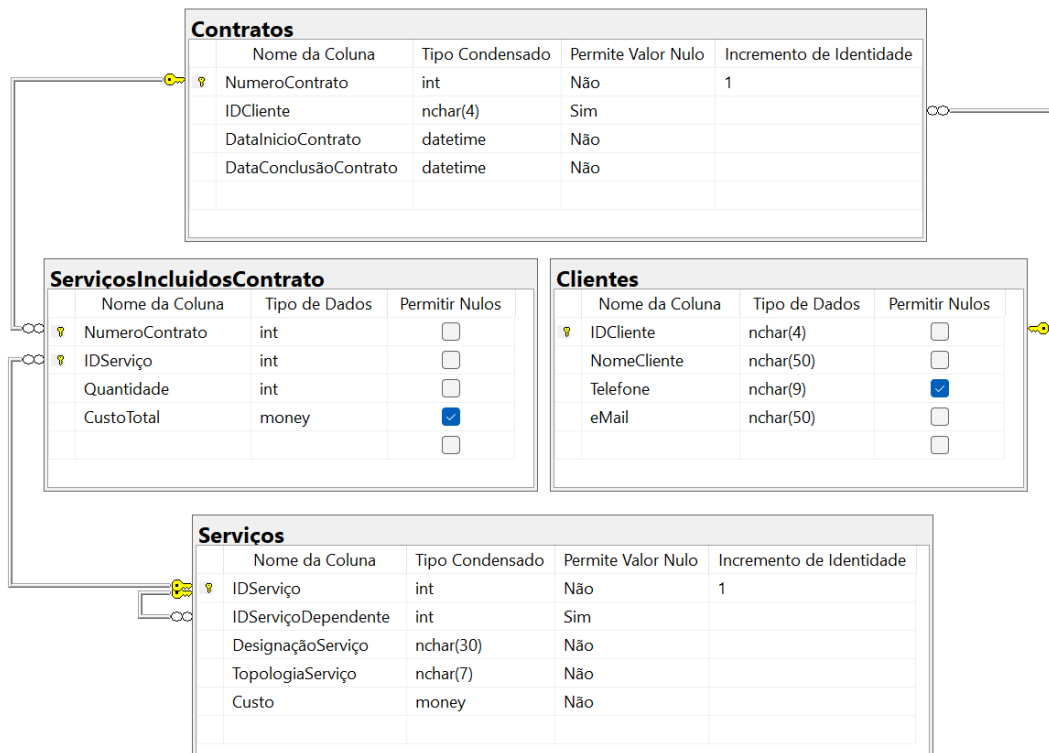


```
Delete From GerirCategorias where IDCategoria = 104
```



### 3.4 Exercícios práticos de SQL-DDL

#### Ex.DDL.1



Considere o modelo de dados apresentado na imagem anterior relacionado com a gestão de informação de contratos de prestação de serviços a clientes. Num único contrato podem ser incluídos mais que um serviço. Desenvolva as tarefas necessárias para implementar os seguintes requisitos:

Crie uma nova base de dados com o nome “**Ex.DDL.1**”: Crie as tabelas, campos, tipo de campos e relações apresentadas no modelo de dados.

Considere as seguintes restrições na definição da tabela “Serviços”: O campo *TopologiaServiço* deve apenas aceitar as opções: Diário, Semanal, Mensal, Anual; Considere que um serviço pode ter outros serviços relacionados ou dependentes.

Considere as seguintes restrições na definição da tabela “Contratos”: O valor do campo *DataInicioContrato* deve apresentar por omissão a data do sistema;

Insira os seguintes dados nas respetivas tabelas:

#### Contratos

| NumeroContrato | IDCliente | DataInicioContrato      | DataConclusãoContrato   |
|----------------|-----------|-------------------------|-------------------------|
| 1              | AA13      | 2010-01-01 00:00:00.000 | 2010-12-31 00:00:00.000 |
| 2              | AA14      | 2010-01-01 00:00:00.000 | 2010-12-31 00:00:00.000 |
| 3              | AA15      | 2010-01-01 00:00:00.000 | 2010-12-31 00:00:00.000 |
| 4              | AA13      | 2010-01-01 00:00:00.000 | 2010-12-31 00:00:00.000 |

#### Clientes

| IDCliente | NomeCliente             | Telefone  | eMail |
|-----------|-------------------------|-----------|-------|
| AA13      | Município de Portalegre | 245300211 |       |
| AA14      | CIMAA                   | 245300247 |       |
| AA15      | ISTU                    | 245457845 |       |

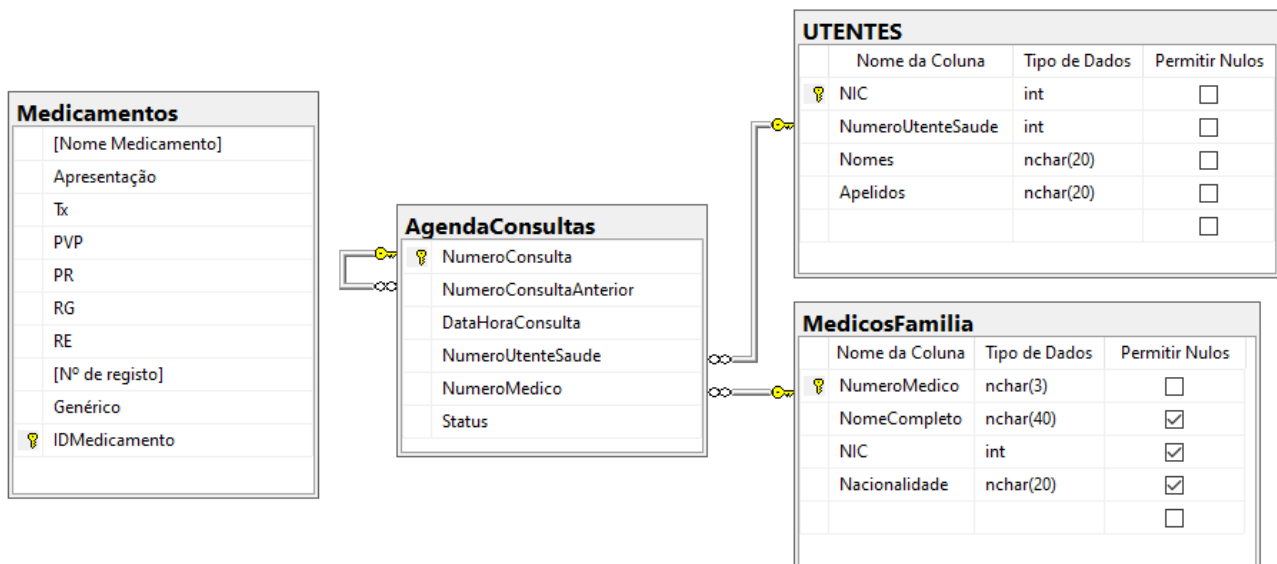
## Serviços

| IDServiço | IDServiçoDependente | DesignaçãoServiço          | TopologiaServiço | Custo    |
|-----------|---------------------|----------------------------|------------------|----------|
| 1         | NULL                | Manutenção Bases de dados  | Mensal           | 10000,00 |
| 2         | NULL                | Desenvolvimento Aplicações | Semanal          | 7500,00  |
| 3         | NULL                | Manutenção de Firewall     | Anual            | 1500,00  |
| 4         | 2                   | Teste de Aplicações        | Semanal          | 2500,00  |

## Serviços Incluídos Contrato

| NumeroContrato | IDServiço | Quantidade | CustoTotal |
|----------------|-----------|------------|------------|
| 1              | 1         | 10         | NULL       |
| 1              | 3         | 1          | NULL       |
| 2              | 2         | 25         | NULL       |
| 3              | 3         | 1          | NULL       |
| 4              | 3         | 12         | NULL       |

## Ex.DDL.2



Considere o modelo de dados apresentado na imagem anterior relacionado com a gestão de informação de consultas médicas de um Centro de Saúde para Utentes do Serviço Nacional de Saúde. Desenvolva no ficheiro as instruções SQL necessárias para implementar os seguintes requisitos:

- Crie uma nova base de dados com o nome “Ex.DDL.2”
- Crie as tabelas, campos, tipo de campos e relações apresentadas no esquema do modelo de dados
- Considere as seguintes restrições na definição da tabela “AgendaConsultas”: O campo NumeroConsulta deve ser incrementado a cada novo registo; O campo Status deve apenas aceitar as opções Agendada, Anulada e Realizada, devendo apresentar por omissão a opção Agendada.
- Considere como informação

### UTENTES

| NIC      | NumeroUtenteSaude | Nomes      | Apelidos |
|----------|-------------------|------------|----------|
| 12345678 | 24364718          | José Silva | Lopes    |
| 87654321 | 24364787          | Maria João | Costa    |

## MedicosFamilia

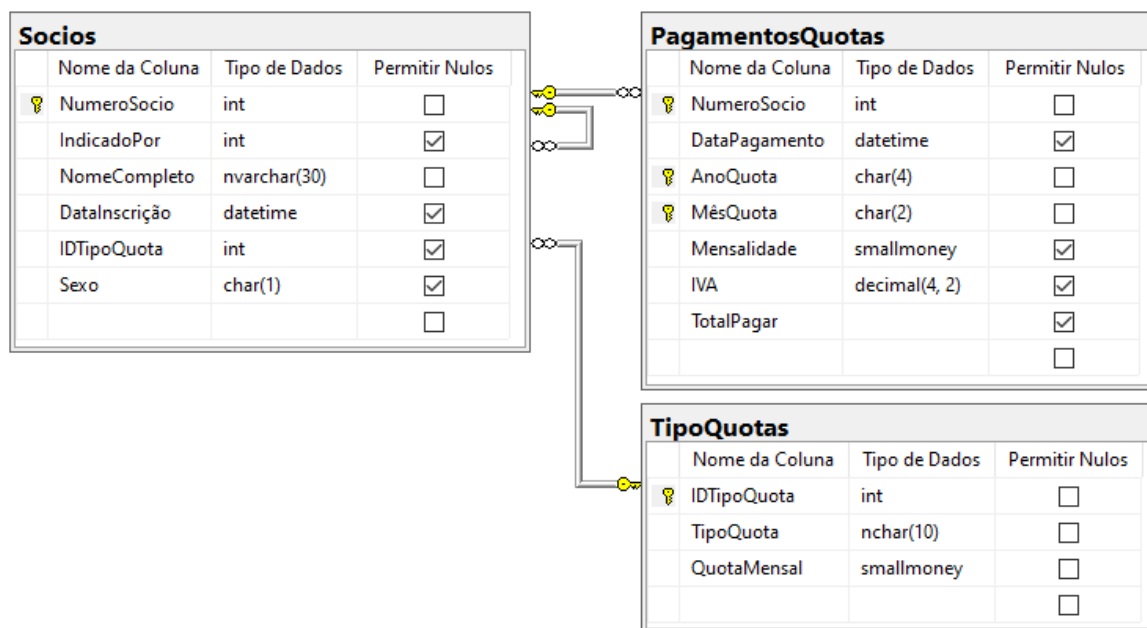
| NumeroMedico | NomeCompleto           | NIC      | Nacionalidade |
|--------------|------------------------|----------|---------------|
| M01          | João Seabra Pereira    | 25371829 | Portugal      |
| M02          | Francisco Manuel Santo | 98654317 | Portugal      |

## AgendaConsultas

| NumeroConsulta | NumeroConsultaAnterior | DataHoraConsulta        | NumeroUtenteSaude | NumeroMedico | Status    |
|----------------|------------------------|-------------------------|-------------------|--------------|-----------|
| 1              | NULL                   | 2011-01-27 10:00:00.000 | 24364718          | M01          | Realizada |
| 2              | NULL                   | 2011-01-27 10:00:00.000 | 24364787          | M02          | Realizada |
| 3              | 1                      | 2011-03-23 11:00:00.000 | 24364718          | M01          | Agendada  |
| 4              | 2                      | 2011-03-24 12:00:00.000 | 24364787          | M02          | Agendada  |

- Recorrendo apenas à instrução UPDATE TABLE, realize a seguinte alteração aos dados da base de dados: Altere o Número do Médico M01 para M10;
- Desenvolva o modelo de dados de forma a implementar o seguinte requisito: pretende-se disponibilizar aos Médicos o registo e controlo da prescrição de medicamentos aos seus utentes em cada consulta. Contudo este deve sempre indicar o problema detetado e o medicamento prescrito em cada caso. Utilize o ficheiro Medicamentos.xls disponível como base da tabela Medicamentos.

## Ex.DDL.3



Considere o Modelo de Dados apresentado na imagem anterior, desenvolva no ficheiro iniciado as instruções SQL necessárias para implementar os seguintes requisitos:

- Crie uma nova base de dados com o nome “Ex.DDL.3”
- Crie as tabelas, campos, tipo de campos e relações apresentadas no modelo de dados
- Considere as seguintes restrições na definição da tabela “Sócios”:
  - O campo NumeroSocio deve ser incrementado a cada novo registo e iniciar em 1000;
  - O campo Sexo deve apenas contemplar as opções M ou F.
  - O campo IndicadoPor tem por objectivo indicar se o novo Sócio foi indicado por um outro Sócio já existente.

- O campo DataInscrição deve apresentar a data do sistema como valor default.
- d. Considere as seguintes restrições na definição da tabela “PagamentosQuotas”:
- O campo DataPagamento deve apresentar a data do sistema como valor default.
  - O campo AnoQuota deve apenas aceitar 4 dígitos numéricos;
  - O campo MêsQuota deve apenas aceitar 2 dígitos numéricos;
  - O campo TotalPagar é calculado pela soma da Mensalidade mais a Mensalidade multiplicada pelo valor do IVA;
- e. Registre nas tabelas a seguinte informação:

#### *TipoQuotas*

| IDTipoQuota | TipoQuota | QuotaMensal |
|-------------|-----------|-------------|
| 1           | Juvenil   | 50,00       |
| 2           | Adulto    | 70,00       |
| 3           | Senior    | 50,00       |

#### *Socios*

| NumeroSocio | IndicadoPor | NomeCompleto      | DataInscrição           | IDTipoQuota | Sexo |
|-------------|-------------|-------------------|-------------------------|-------------|------|
| 1000        | NULL        | João Paulo Costa  | 2010-01-01 00:00:00.000 | 1           | M    |
| 1001        | NULL        | Maria João Abreu  | 2010-01-01 00:00:00.000 | 2           | F    |
| 1002        | NULL        | Pedro José Santos | 2010-01-01 00:00:00.000 | 3           | M    |

#### *PagamentosQuotas*

| NumeroSocio | DataPagamento           | AnoQuota | MêsQuota |
|-------------|-------------------------|----------|----------|
| 1000        | 2010-01-19 10:45:51.060 | 2010     | 01       |
| 1000        | 2010-01-19 10:45:51.060 | 2010     | 02       |
| 1001        | 2010-01-19 10:45:51.060 | 2010     | 01       |
| 1001        | 2010-01-19 10:45:51.060 | 2010     | 02       |
| 1002        | 2010-01-19 10:45:51.060 | 2010     | 01       |

- f. Recorrendo apenas à instrução UPDATE TABLE, realize as seguintes alterações aos dados da base de dados:
- Os sócios 1000 e 1001 foram indicados pelo sócio 1002.
  - Altere o valor da QuotaMensal do TipoQuota “Senior” para 40,00€
  - Altere na tabela TipoQuotas o valor do IDTipoQuota 3 para 13
- g. Recorrendo apenas à instrução ALTER TABLE, realize a seguinte alteração à base de dados: adicione um novo campo à tabela *Socios* designado “EstadoInscrição” no qual podem apenas ser contempladas as opções “Activa” “Inativa”. Todos os registos existentes devem ser atualizados com o campo Socios.EstadoInscrição = Activa.

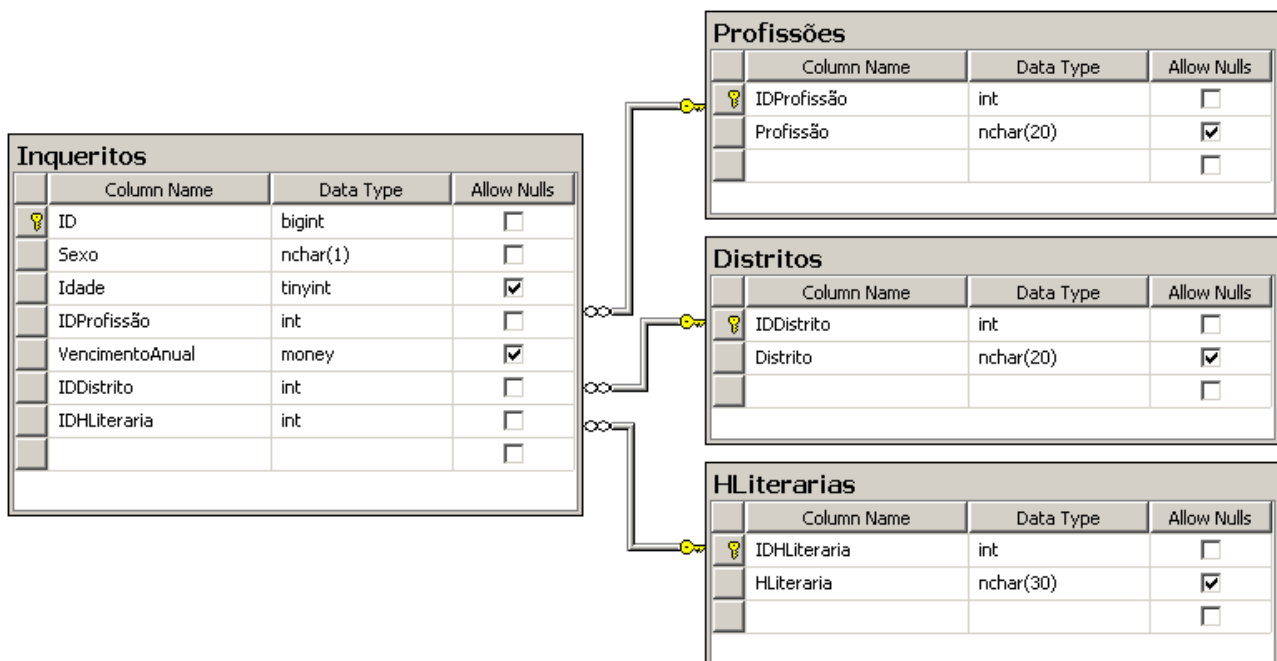
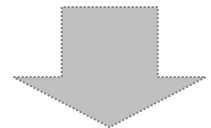
## Ex.DDL.4

Considere os dados apresentados referentes a inquéritos relacionados com um processo de estudos de mercado. Os dados encontram-se disponíveis no ficheiro Excel “Inqueritos2.xls”.

- Crie uma nova base de dados com a designação “Ex.DDL.4”
- Através do processo de importação, importe os dados do ficheiro de Excel para uma nova tabela de dados.
- Inicie uma nova *Query* para promover as alterações necessárias:

- Altere o nome da tabela importada para “Inqueritos”.
- Promova as alterações à estrutura da tabela importada de acordo com o esquema apresentado.
- Elimine a redundância existente nos campos: *Profissão*, *Distrito* e *Formação*. Devem ser criadas novas tabelas com as opções existentes e que devem derivar da consulta à tabela “Inqueritos”. Não deve ser eliminada nenhuma informação/registo.

|  | Column Name     | Data Type     | Allow Nulls                         |
|--|-----------------|---------------|-------------------------------------|
|  | ID              | float         | <input checked="" type="checkbox"/> |
|  | Sexo            | nvarchar(255) | <input checked="" type="checkbox"/> |
|  | Idade           | float         | <input checked="" type="checkbox"/> |
|  | Profissão       | nvarchar(255) | <input checked="" type="checkbox"/> |
|  | VencimentoAnual | money         | <input checked="" type="checkbox"/> |
|  | Distrito        | nvarchar(255) | <input checked="" type="checkbox"/> |
|  | Formação        | nvarchar(255) | <input checked="" type="checkbox"/> |
|  | F8              | float         | <input checked="" type="checkbox"/> |
|  | F9              | float         | <input checked="" type="checkbox"/> |
|  | F10             | float         | <input checked="" type="checkbox"/> |
|  |                 |               | <input type="checkbox"/>            |



Inicie uma nova Query que permita obter os dados solicitados nos seguintes quadros:

Quadro 1 - Análise global

|                        |          |   |                 |                     |                    |
|------------------------|----------|---|-----------------|---------------------|--------------------|
| Nº total de respostas: |          |   |                 |                     |                    |
|                        | Contagem | % | Média de Idades | Idade do mais velho | Idade do mais Novo |
| Homens                 |          |   |                 |                     |                    |
| Mulheres               |          |   |                 |                     |                    |
| Total                  |          |   |                 |                     |                    |

Quadro 2

|             |      |   |       |   |
|-------------|------|---|-------|---|
|             | Sexo |   |       |   |
| Profissão   | f    | m | Total | % |
| Agricultura |      |   |       |   |
| Indústria   |      |   |       |   |
| Serviços    |      |   |       |   |
| Total       |      |   |       |   |
| %           |      |   |       |   |

Quadro 3 - Pretende-se analisar a média do Vencimento Anual, de acordo com a estrutura da tabela seguinte.

|      |              | Distrito |       |        |            |       |
|------|--------------|----------|-------|--------|------------|-------|
| Sexo | Habilitações | Beja     | Évora | Guarda | Portalegre | Viseu |
| f    | 12º Ano      |          |       |        |            |       |
|      | Bacharelato  |          |       |        |            |       |
|      | Doutoramento |          |       |        |            |       |
|      | Licenciatura |          |       |        |            |       |
|      | Mestrado     |          |       |        |            |       |
| m    | 12º Ano      |          |       |        |            |       |
|      | Bacharelato  |          |       |        |            |       |
|      | Doutoramento |          |       |        |            |       |
|      | Licenciatura |          |       |        |            |       |
|      | Mestrado     |          |       |        |            |       |

## 4. Linguagem T- SQL

### 4.1 Variáveis

Em T-SQL as variáveis são declaradas através da instrução *DECLARE*. Estas podem ser locais ou globais. As variáveis globais, variáveis do sistema não podem ser criadas pelo utilizador. Estas distinguem-se das locais pois são iniciadas por dois símbolos @.

Sintaxe:

```
DECLARE
  { {@local_variable31 [AS] data_type32 }
    | {@cursor_variable_name33 CURSOR }
    | {@table_variable_name34 < table_type_definition> } } [ ,...n ]
< table_type_definition35 > ::=
  TABLE ( { < column_definition > | < table_constraint > } [ ,... ] )
< column_definition > ::=
  column_name { data_type | AS computed_column_expression36 }
    [ COLLATE37 collation_name ]
    [ [ DEFAULT38 constant_expression39 ] | IDENTITY40 [(seed,increment)] ]
    [ ROWGUIDCOL41 ]
    [ < column_constraint > ]
< column_constraint > ::=
  { [ NULL | NOT NULL ]
    | [ PRIMARY KEY | UNIQUE ]
    | CHECK ( logical_expression ) }
< table_constraint > ::=
  { { PRIMARY KEY | UNIQUE } ( column_name [ ,... ] )
    | CHECK ( search_condition ) }
```

---

<sup>31</sup> Is the name of a variable. Variable names must begin with an at (@) sign

<sup>32</sup> A variable cannot be of **text**, **ntext**, or **image** data type

<sup>33</sup> Is the name of a cursor variable. Cursor variable names must begin with an at (@) sign and conform to the rules for identifiers

<sup>34</sup> Is the name of a variable of type **table**. Variable names must begin with an at (@) sign and conform to the rules for identifiers

<sup>35</sup> Defines the **table** data type. The table declaration includes column definitions, names, data types, and constraints. The only constraint types allowed are PRIMARY KEY, UNIQUE, NULL, and CHECK.

<sup>36</sup> Is an expression defining the value of a computed column

<sup>37</sup> Specifies the collation for the column. *collation\_name* can be either a Windows collation name or an SQL collation name, and is applicable only for columns of the **char**, **varchar**, **text**, **nchar**, **nvarchar**, and **ntext** data types

<sup>38</sup> Specifies the value provided for the column when a value is not explicitly supplied during an insert

<sup>39</sup> Is a constant, NULL, or a system function used as the default value for the column.

<sup>40</sup> Indicates that the new column is an identity column.

<sup>41</sup> Indicates that the new column is a row global unique identifier column. Only one **uniqueidentifier** column per table can be designated as the ROWGUIDCOL column.

## Exemplo 1

```
-- Exemplo de atribuição de valores
DECLARE @A INT,@A1 NVARCHAR(25);
SELECT @A as A, @A1 as A1;
SET @A = 1;
SET @A1 = 'Valor';
SELECT @A as 'A', @A1 'A1';
```

| Resultados |      | Mensagens |
|------------|------|-----------|
|            | A    | A1        |
| 1          | NULL | NULL      |

|   | A | A1    |
|---|---|-------|
| 1 | 1 | Valor |

Sintaxe:

```
PRINT42 msg_str43 | @local_variable44 | string_expr45
```

## Exemplo 2

```
USE ProDados
Declare @TempID nvarchar(10)
SET @TempID = 'BOLID'
Select * From Customers Where CustomerID=@TempID
```

```
Declare @TempID nvarchar(10), @CompanyName nvarchar(50)
SET @TempID = 'BOLID'
Select @CompanyName = CompanyName
      From Customers Where CustomerID = @TempID
Print @CompanyName
```

```
-- Calcular o total das vendas, guardar o valor obtido numa
variável e apresentar (output) o valor guardado na variável
Declare @TotalVendas Decimal(10,2)
Select @TotalVendas = sum(UnitPrice*Quantity) from [OrderDetails]
Print @TotalVendas
-- OU
Declare @TotalVendas Decimal(10,2)
SET @TotalVendas =
      (Select sum(UnitPrice*Quantity) from [OrderDetails])
Print @TotalVendas
```

---

<sup>42</sup> Returns a user-defined message to the client

<sup>43</sup> Is a character string or Unicode string constant

<sup>44</sup> Is a variable of any valid character data type. *@local\_variable* must be **char** or **varchar**, or it must be able to be implicitly converted to those data types

<sup>45</sup> Is an expression that returns a string. Can include concatenated literal values, functions, and variables



### Exemplo 3

```
USE ProDados
DECLARE @find varchar(30);
SET @find = 'M%';
SELECT LastName, FirstName FROM Employees
WHERE FirstName LIKE @find;
```

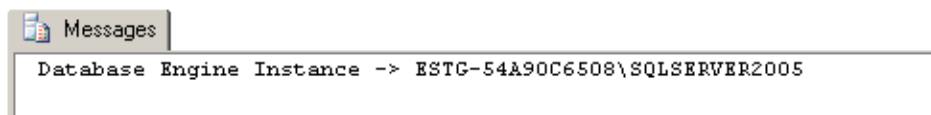
| Resultados |          | Mensagens |  |
|------------|----------|-----------|--|
|            | LastName | FirstName |  |
| 1          | Peacock  | Margaret  |  |
| 2          | Suyama   | Michael   |  |

### Exemplo 4 – Funções do sistema (variáveis globais)

|                        |  |
|------------------------|--|
| Select @@Language;     | Returns the name of the language currently being used  |
| Select @@ServerName;   | Returns the name of the local server running SQL Server.   |
| Select @@SPID;         | Returns the session ID of the current user process   |
| Select @@RowCount;     | Returns the number of rows affected by the last statement  |
| Select @@ServiceName;  | Returns the name of the registry key under which SQL Server is running   |
| Select @@TextSize;     | Returns the current value of the TEXTSIZE option of the SET statement  |
| Select @@Total_Errors; | Returns the number of disk write errors encountered by SQL Server since last started                                 |
| Select @@Version;      | Returns version, processor architecture, build date, and operating system for the current installation of SQL Server |

### Exemplo 5

```
-- Para obter o "Server Name" do Database Engine
DECLARE @Msg NVARCHAR(300);
SELECT @Msg = N'Database Engine Instance -> ' + RTRIM(@@SERVERNAME)
PRINT @Msg;
```



Para apoio a determinados processos ou mesmo como medida de segurança, pode ser necessário recorrer a uma ou mais tabelas temporárias de forma a suportar a sua realização. Uma vez concluído o processo a tabela é eliminada.

Existem dois tipos de **tabelas temporárias: locais e globais**. As locais são apenas visíveis para o utilizador que as criou, enquanto que as globais são visíveis para todos os utilizadores da base de dados. São similares a tabelas permanentes. Contudo ao contrário destas são armazenadas na TempDB<sup>46</sup> e são apagadas de forma automática quando já não são utilizadas. As tabelas temporárias locais devem conter o carácter “#” como o primeiro carácter no nome da tabela, enquanto que as tabelas temporárias globais devem apresentar dois caracteres “##” no nome da tabela.

<sup>46</sup> The tempdb system database is a global resource available to all users connected to the instance of SQL Server, and holds all temporary tables and temporary stored procedures. tempdb is re-created every time SQL Server is started so that the system always starts with a clean copy of the database.

## Exemplo 6

```
-- Criar uma tabela temporária local
CREATE TABLE #ProductTemp
    (ProductID INT PRIMARY KEY, Name nvarchar(30));

SELECT Name FROM TempDB.dbo.SysObjects
```



|   | Name         |
|---|--------------|
| 1 | #ProductTemp |

```
-- Inserção de valores na tabela temporária local
INSERT INTO #ProductTemp Values (1, 'A'), (2, 'B'), (3, 'C')

Select * from #ProductTemp
```

## Exemplo 7

Pretende-se criar uma tabela temporária para guardar temporariamente as vendas totais (UnitPrice\*Quantity) por produto.

```
Create Table #VendasTotaisProdutos
    (ProductID int primary key,
     ProductName nvarchar(50),
     TotalVendas decimal(10,2))

INSERT INTO #VendasTotaisProdutos
Select P.ProductID, ProductName, sum(OD.UnitPrice*Quantity)
From Products P
JOIN [OrderDetails] OD ON P.ProductID = OD.ProductID
Group BY P.ProductID, ProductName
Order By P.ProductID
```

```
Select * from #VendasTotaisProdutos
```

## Exemplo 8

```
-- Variável do tipo Tabela
DECLARE @TabelaBackup TABLE (C1 INT PRIMARY KEY, C2 INT NOT NULL);

INSERT INTO @TabelaBackup (C1, C2) VALUES (1, 101);
INSERT INTO @TabelaBackup (C1, C2) VALUES (2, 102);

SELECT * FROM @TabelaBackup;
```

## Exemplo 9

```
Declare @dt datetime
SET @dt = (Select getdate())
Print @dt
Print 'Data de hoje --> ' + cast(@dt AS nvarchar(30))
Print 'Data de hoje --> ' + convert(nvarchar(30),@dt)
--

Declare @a as decimal(18,2)
SET @a = (14*78425125154)/45
Print @a
Print 'Valor de a --> ' + convert(nvarchar(20),@a)
```

Sintaxe das funções CAST e CONVERT:

```
CAST47 ( expression AS data_type [ (length) ] )
CONVERT ( data_type [ (length) ] , expression [ ,style] )
```

## 4.2 Mecanismos de controlo da linguagem T-SQL

A linguagem T-SQL possui como mecanismos de controlo: BEGIN e END, BREAK, CONTINUE, GOTO, IF e ELSE, RETURN, WAITFOR e WHILE.

Sintaxe:

```
BEGIN
    { sql_statement | statement_block }
END
```

```
IF48 Boolean_expression49
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

```
WHILE Boolean_expression
    { sql_statement | statement_block }
    [ BREAK50 ]
    { sql_statement | statement_block }
    [ CONTINUE51 ]
    { sql_statement | statement_block }
```

---

<sup>47</sup> Explicitly converts an expression of one data type to another. CAST and CONVERT provide similar functionality.

<sup>48</sup> An IF...ELSE construct can be used in batches, in stored procedures, and in ad hoc queries. When this construct is used in a stored procedure, it is frequently used to test for the existence of some parameter.

<sup>49</sup> Is an expression that returns TRUE or FALSE. If the Boolean expression contains a SELECT statement, the SELECT statement must be enclosed in parentheses.

<sup>50</sup> Causes an exit from the innermost WHILE loop.

<sup>51</sup> Causes the WHILE loop to restart, ignoring any statements after the CONTINUE keyword

Os erros podem ser processados usando blocos de instruções TRY e CATCH semelhante aos utilizados em outras linguagens como Java e C++. Existem dois blocos, o bloco TRY e o bloco CATCH. Quando um erro é detetado no interior do bloco TRY o controlo é passado para o bloco CATCH onde o erro pode ser processado. O bloco TRY começa com a instrução BEGIN TRY e termina com END TRY. Um bloco CATCH começa com BEGIN CATCH e termina com END CATCH.

Sintaxe:

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    { sql_statement | statement_block }
END CATCH [ ; ]
```

```
RETURN52 [ integer_expression53 ]
```

### Exemplo 10

```
Declare @A Int, @B Int
Set @A = 1
Set @B = 10
While @A <= @B
    Begin
        Print 'A - ' + Convert(Varchar(10), @A);
        Set @A = @A+1
    End
```

### Exemplo 11

```
USE ProDados;
Declare @p int, @UnitsInStock int, @ReorderLevel int
SET @p = 76
Select @UnitsInStock = UnitsInStock, @ReorderLevel = ReorderLevel
From Products where ProductID = @p

IF @UnitsInStock < @ReorderLevel
    Print 'Necessário encomendar!'
ELSE
    Begin
        Print 'Stock ok!'
        Print '-----'
    End
```

---

<sup>52</sup> Exits unconditionally from a query or procedure. RETURN is immediate and complete and can be used at any point to exit from a procedure, batch, or statement block. Statements that follow RETURN are not executed.

<sup>53</sup> Is the integer value that is returned. Stored procedures can return an integer value to a calling procedure or an application.

## Exemplo 12

Utilize variáveis e uma estrutura de controlo para desenvolver o processo apresentado na imagem seguinte.



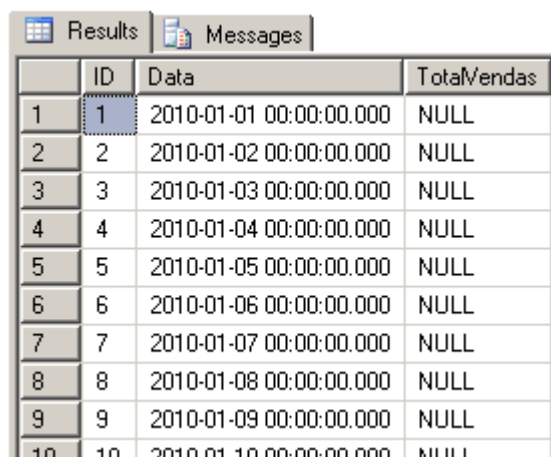
```
Mensagens
Início da Contagem
Data introduzida - Dec 20 2009
Contador - Data      - Dia
-----
0          - 20 2009 - Sunday
1          - 21 2009 - Monday
2          - 22 2009 - Tuesday
3          - 23 2009 - Wednesday
4          - 24 2009 - Thursday
5          - 25 2009 - Friday
6          - 26 2009 - Saturday
7          - 27 2009 - Sunday
-----
Fim da Contagem - 8
```

## Exemplo 13

Crie uma variável com a seguinte estrutura:

```
DECLARE @RelatorioDiario AS Table (
    ID int Identity(1,1) Primary Key,
    Data DateTime,
    TotalVendas bigint
)
```

Utilize uma estrutura de controlo e as variáveis necessárias para preencher a variável anterior de forma automática, com um registo por cada dia do ano atual, de acordo com a imagem seguinte:



|    | ID | Data                    | TotalVendas |
|----|----|-------------------------|-------------|
| 1  | 1  | 2010-01-01 00:00:00.000 | NULL        |
| 2  | 2  | 2010-01-02 00:00:00.000 | NULL        |
| 3  | 3  | 2010-01-03 00:00:00.000 | NULL        |
| 4  | 4  | 2010-01-04 00:00:00.000 | NULL        |
| 5  | 5  | 2010-01-05 00:00:00.000 | NULL        |
| 6  | 6  | 2010-01-06 00:00:00.000 | NULL        |
| 7  | 7  | 2010-01-07 00:00:00.000 | NULL        |
| 8  | 8  | 2010-01-08 00:00:00.000 | NULL        |
| 9  | 9  | 2010-01-09 00:00:00.000 | NULL        |
| 10 | 10 | 2010-01-10 00:00:00.000 | NULL        |

## 4.3 Stored Procedures

O desenvolvimento de aplicações para interface com bases de dados, implica que todas as regras de negócio da aplicação devam residir do lado do cliente, tornando a base de dados um simples depósito de texto organizado lógica e fisicamente. SGDBs Cliente/Servidor permitem que estas regras “desçam”, se não totalmente, em grande parte, para o Servidor. Para isso existem blocos modulares de código, recompilados e armazenados no servidor como parte da Base de Dados, conhecidos como *stored procedures e triggers*<sup>54</sup>. Cada um destes blocos pode conter instruções de controlo de fluxo, comandos DML<sup>55</sup> ou simples consultas. Entre as vantagens da sua utilização, que muitas vezes se confundem com as vantagens da própria arquitetura cliente/servidor podemos citar:

- a. Centralização: qualquer mudança numa regra de negócio da aplicação implicava a atualização de dezenas, às vezes centenas de aplicações espalhadas pela empresa, pela cidade ou pelo mundo. Ao invés disso, basta executar um script no servidor e os utilizadores nem sequer ficam a saber que mudanças ocorreram;
- b. Segurança: a vida do administrador da Bases de Dados fica mais fácil, ao invés de ter que quebrar a cabeça com diretivas de segurança para diferentes níveis de privilégios sobre os diferentes objetos de uma base de dados, pode definir a segurança a partir de *stored procedures*;
- c. Tráfego de rede: um *stored procedure* ou um *trigger* são executados no servidor, sendo que o cliente se limita a executá-los, implícita ou explicitamente, com eventuais parâmetros, e pode receber de volta algum retorno.

### 4.3.1 Tipos de Stored Procedures

O SQL Server possui pelo menos 4 tipos de procedimentos: (1) *locais*, são os procedimentos de utilizador; (2) *temporários*, existentes apenas durante uma sessão; de (3) *sistema*, criados pelo Servidor durante a instalação com funções administrativas e (4) *estendidos*, chamados a partir de uma DLL.

- a. *Extended Stored procedure* - são DLL's criadas em alguma linguagem de programação, que podem ser executadas dinamicamente pelo SQL Server. Normalmente são desenvolvidas para executar tarefas que o SQL Server não pode fazer. Como convenção, possuem o prefixo `XP_` no nome.
- b. *Temporários* - Procedimentos armazenados na base de dados *TempDB* (Outros objetos com exceção de *views* ou *functions*, podem ser criados de forma temporária). Existem dois tipos de procedimentos temporários:

**Locais:** visíveis apenas na sessão aberta, ou seja, para o utilizador que o criou. Este tipo de procedimento é excluído automaticamente com o fim da sessão. Para criar um procedimento temporário local deve ser adicionado o prefixo `#` ao nome do mesmo.

Ex: `CREATE PROCEDURE #AnalisarVendas(...)`

---

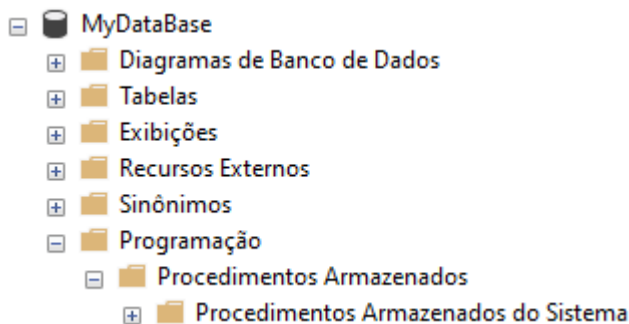
<sup>54</sup> Um trigger permite que uma determinada sequência de comandos SQL seja accionada quando um determinado evento ocorre

<sup>55</sup> Data Manipulation Language

**Globais:** Visíveis para todas as sessões. Este tipo de procedimento também é excluído automaticamente com o fim da sessão em que foi criado, a não ser que esteja a ser utilizado por alguma outra sessão no momento da exclusão, neste caso, a exclusão ocorrerá ao fim do comando T-SQL que lhe fez referência. Para criar um procedimento temporário global, deve ser adicionado o prefixo ## ao nome do mesmo.

Ex: **CREATE PROCEDURE ##AnalisarVendas (...)**

- c. Procedimentos de sistema (*System Stored Procedures*) – Constitui uma ferramenta importantíssima do SQL Server. Estes procedimentos são criados durante a instalação do SQL Server e residem na base de dados *Master*. A sua função é facilitar a realização de rotinas administrativas, bem como obter informações sobre o sistema e/ou seus objetos. Um procedimento de sistema tem por convenção “sp\_” como prefixo, e é executado como um procedimento de utilizador.



**Exemplo 14** - *sp\_helpdb*: obtém informações sobre uma base de dados no servidor

**EXEC sp\_helpdb** MyDataBase

|   | name       | db_size  | owner | dbid | created    | status  | compatibility_level |
|---|------------|----------|-------|------|------------|---|---------------------|
| 1 | MyDataBase | 16.00 MB | sa    | 23   | Dec 1 2021 | Status=ONLINE, Updateability=READ_WRITE, UserAcc... | 150                 |

|   | name           | fileid | filename   | filegroup | size    | maxsize       | growth   | usage     |
|---|----------------|--------|--|-----------|---------|---------------|----------|-----------|
| 1 | MyDataBase     | 1      | C:\Program Files\Microsoft SQL Server\MSSQL15.MSS... | PRIMARY   | 8192 KB | Unlimited     | 65536 KB | data only |
| 2 | MyDataBase_log | 2      | C:\Program Files\Microsoft SQL Server\MSSQL15.MSS... | NULL      | 8192 KB | 2147483648 KB | 65536 KB | log only  |

**Exemplo 15** - *sp\_help*: obtém informações sobre um objeto de uma base de dados, como por exemplo, uma tabela.

**USE** MyDataBase

**EXEC sp\_help** Produtos

|   | Name     | Owner | Type      | Created_datetime        |
|---|----------|-------|-----------|-------------------------|
| 1 | Produtos | dbo   | usertable | 2021-12-01 13:07:33.640 |

|   | Column_name      | Type     | Computed | Length | Prec | Scale | Nullable | TrimTrailingBlanks | FixedLenNullInSource | Collation            |
|---|------------------|----------|----------|--------|------|-------|----------|--------------------|----------------------|----------------------|
| 1 | IDProduto        | int      | no       | 4      | 10   | 0     | no       | (n/a)              | (n/a)                | NULL                 |
| 2 | NomeProduto      | nchar    | no       | 100    |      |       | yes      | (n/a)              | (n/a)                | Latin1_General_CI_AS |
| 3 | PreçoUnitario    | money    | no       | 8      | 19   | 4     | yes      | (n/a)              | (n/a)                | NULL                 |
| 4 | IDCategoria      | smallint | no       | 2      | 5    | 0     | yes      | (n/a)              | (n/a)                | NULL                 |
| 5 | NumeroFornecedor | smallint | no       | 2      | 5    | 0     | yes      | (n/a)              | (n/a)                | NULL                 |



### Exemplo 16 - *sp\_columns*: obtém informações sobre colunas de uma tabela

USE MyDataBase

EXEC sp\_Columns Produtos

|   | TABLE_QUALIFIER | TABLE_OWNER | TABLE_NAME | COLUMN_NAME      | DATA_TYPE | TYPE_NAME | PRECISION | LENGTH | SCALE |
|---|-----------------|-------------|------------|------------------|-----------|-----------|-----------|--------|-------|
| 1 | MyDataBase      | dbo         | Produtos   | IDProduto        | 4         | int       | 10        | 4      | 0     |
| 2 | MyDataBase      | dbo         | Produtos   | NomeProduto      | -8        | nchar     | 50        | 100    | NULL  |
| 3 | MyDataBase      | dbo         | Produtos   | PreçoUnitario    | 3         | money     | 19        | 21     | 4     |
| 4 | MyDataBase      | dbo         | Produtos   | IDCategoria      | 5         | smallint  | 5         | 2      | 0     |
| 5 | MyDataBase      | dbo         | Produtos   | NumeroFornecedor | 5         | smallint  | 5         | 2      | 0     |

- d. Procedimentos de Utilizador - procedimentos utilizados como uma extensão da aplicação no servidor. Normalmente este é um tipo de procedimento muito importante no desenvolvimento de software.

Sintaxe:

```
CREATE {PROC|PROCEDURE}[schema_name.]procedure_name56[;number57]  
    [ { @parameter58 [ type_schema_name. ] data_type59 }  
      [ VARYING60 ] [ = default61 ] [ [ OUTPUT62 ] [ ,...n ]  
[ WITH <procedure_option> [ ,...n ]  
[ FOR REPLICATION63 ]  
AS { <sql_statement> [;][ ...n ] | <method_specifier> }[;]  
<procedure_option> ::=  
    [ ENCRYPTION64 ][ RECOMPILE65 ][ EXECUTE_AS_Clause66 ]  
<sql_statement67> ::=  
{ [ BEGIN ] statements [ END ] }  
<method_specifier> ::= EXTERNAL NAME68  
assembly_name.class_name.method_name
```

<sup>56</sup> Procedure names must comply with the rules for identifiers and must be unique within the schema. We strongly recommend that you not use the prefix **sp\_** in the procedure name.

<sup>57</sup> Is an optional integer that is used to group procedures of the same name.

<sup>58</sup> Is a parameter in the procedure. One or more parameters can be declared. The value of each declared parameter must be supplied by the user when the procedure is called, unless a default for the parameter is defined or the value is set to equal another parameter. A stored procedure can have a maximum of 2,100 parameters.

<sup>59</sup> Is the data type of the parameter and the schema to which it belongs. All data types, except the **table** data type, can be used as a parameter for a Transact-SQL stored procedure.

<sup>60</sup> Specifies the result set supported as an output parameter. This parameter is dynamically constructed by the stored procedure and its contents may vary. Applies only to **cursor** parameters.

<sup>61</sup> Is a default value for the parameter. If a *default* value is defined, the procedure can be executed without specifying a value for that parameter. The default must be a constant or it can be NULL.

<sup>62</sup> Indicates that the parameter is an output parameter. The value of this option can be returned to the calling EXECUTE statement. Use OUTPUT parameters to return values to the caller of the procedure. **text**, **ntext**, and **image** parameters cannot be used as OUTPUT parameters, unless the procedure is a CLR procedure.

<sup>63</sup> Specifies that stored procedures that are created for replication cannot be executed on the Subscriber. A stored procedure created with the FOR REPLICATION option is used as a stored procedure filter and is executed only during replication. Parameters cannot be declared if FOR REPLICATION is specified.

<sup>64</sup> Indicates that SQL Server will convert the original text of the CREATE PROCEDURE statement to an obfuscated format.

<sup>65</sup> Indicates that the Database Engine does not cache a plan for this procedure and the procedure is compiled at run time. This option cannot be used when FOR REPLICATION is specified.

<sup>66</sup> Specifies the security context under which to execute the stored procedure

<sup>67</sup> Is one or more Transact-SQL statements to be included in the procedure.

<sup>68</sup> Specifies the method of a .NET Framework assembly for a CLR stored procedure to reference.



## Exemplo 17

Pretende-se desenvolver um procedimento para juntar num único campo: *Título de cortesia* + *Nome* + *Apelido* dos funcionários da tabela *Employees* da base de dados ProDados.

Criar o procedimento

```
USE ProDados
GO
CREATE PROC pr_NomeCompleto as
Select TitleOfCourtesy + ' ' + FirstName + ' ' + LastName 'Full
Name' From Employees
```

Executar o procedimento - A execução pode ser feita pela simples chamada do procedimento, quando esta for o primeiro comando de um bloco, ou através do método execute (ou simplesmente exec):

```
Exec pr_NomeCompleto
```



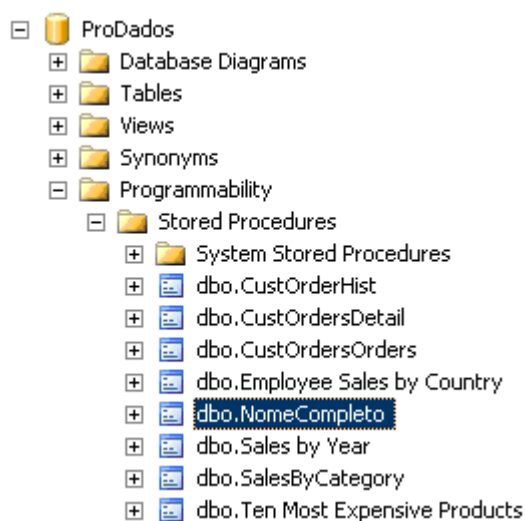
|   | Full Name             |
|---|-----------------------|
| 1 | Ms. Nancy Davolio     |
| 2 | Dr. Andrew Fuller     |
| 3 | Ms. Janet Leverling   |
| 4 | Mrs. Margaret Peacock |
| 5 | Mr. Steven Buchanan   |
| 6 | Mr. Michael Suyama    |
| 7 | Mr. Robert King       |
| 8 | Ms. Laura Callahan    |
| 9 | Ms. Anne Dodsworth    |

Os procedimentos do tipo utilizador ficam guardados na pasta Programmability\Stored Procedure da Base de Dados.

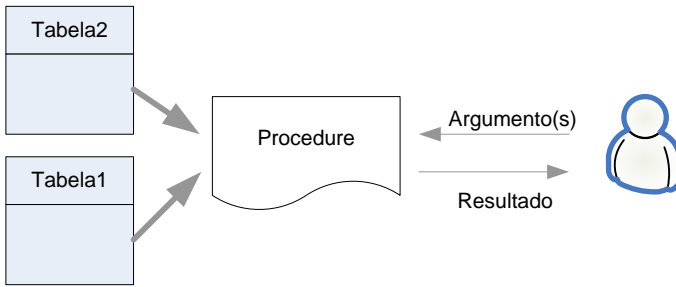
De forma a evitar erros, é útil verificar antes da criação do procedimento, se o mesmo existe na base de dados. Para isso verifica-se se existe alguma entrada/registo na tabela *sysobjects* com nome “NomeCompleto” e tipo P (procedimento). Em caso afirmativo o procedimento é excluído antes da sua criação:

```
USE ProDados
IF Exists (Select [Name]
From sysobjects Where [Name] =
'pr_NomeCompleto' and Type = 'P')
Drop Proc pr_NomeCompleto
```

```
GO
CREATE PROC pr_NomeCompleto as
Select TitleOfCourtesy + ' ' + FirstName + ' ' + LastName 'Full
Name' From Employees
GO
Exec pr_NomeCompleto
```



### Exemplo 18



Pretende-se alterar o procedimento de modo a condicionar (filtrar) a listagem dos empregados com dois parâmetros: *Pais* e *TitleOfCourtesy*.

```
USE ProDados
GO
ALTER PROC pr_NomeCompleto @Pais nchar(30), @TitleOfCourtesy
nchar(25) AS
Select TitleOfCourtesy + ' ' + FirstName + ' ' + LastName 'Full
Name' From Employees
Where Country = @Pais and TitleOfCourtesy = @TitleOfCourtesy
```

Podemos executar o procedimento de duas formas: passando os argumentos, na mesma ordem, separados por vírgula:

```
Exec pr_NomeCompleto UK, 'MR.'
```

Ou citando o nome do parâmetro seguido do valor:

```
Exec pr_NomeCompleto @Pais = 'UK', @TitleOfCourtesy = 'MR.'
```

### 4.3.2 Parâmetro Default

Se tentarmos executar o procedimento anterior sem indicar o valor dos parâmetros, teremos um erro, pois estes não podem ser omitidos, a não ser que possuam um valor *Default*. Um valor *default* pode ser indicado logo após a declaração do parâmetro.

### Exemplo 19

```
USE ProDados
GO
ALTER PROC pr_NomeCompleto @Pais nchar(30) = 'USA',
@TitleOfCourtesy nchar(25) = 'MR.' AS
Select TitleOfCourtesy + ' ' + FirstName + ' ' + LastName 'Full
Name' From Employees
Where Country = @Pais and TitleOfCourtesy = @TitleOfCourtesy
```

Se executarmos o procedimento apenas com um dos parâmetros, o valor *default* dos outros parâmetros será utilizado no procedimento:

```
Exec pr_NomeCompleto @Pais = 'UK'
Exec pr_NomeCompleto @TitleOfCourtesy = 'Ms.'
```

### 4.3.3 Argumento *With Encryption*

A tabela de sistema *syscomments* possui o texto de todos os procedimentos e triggers (e de outros objetos). Cada base de dados possui sua própria tabela *syscomments*. Através desta tabela podemos recuperar o texto usado na criação do objeto. Caso exista a necessidade de ocultar o texto do procedimento ou trigger, pode-se usar o argumento *With Encryption* no momento da criação do objeto. Isto encripta o texto do objeto, tornando-o ilegível. Nem mesmo o administrador da base de dados consegue recuperar o texto de um objecto encriptado. Para encriptar o procedimento deve ser utilizado o argumento *with encryption* após o nome do procedimento:

#### Exemplo 20

```
USE ProDados
IF EXISTS (SELECT [Name] FROM sysobjects WHERE [Name] =
'pr_NomeCompleto' and Type = 'P')
Drop PROC pr_NomeCompleto
GO

Create Procedure pr_NomeCompleto @Pais nchar(30) = 'UK'
With Encryption AS
Select TitleOfCourtesy + ' ' + FirstName + ' ' + LastName 'Full
Name' From Employees
Where Country = @Pais
GO
Exec pr_NomeCompleto
Exec pr_NomeCompleto @Pais = 'USA'
```

### 4.3.4 Execução Automática

Um procedimento pode ser configurado para execução automática sempre que o SQL Server for iniciado. Tal procedimento não pode ter parâmetros de entrada, deve residir na base de dados Master e ser criado por um membro do grupo/perfil sysadmin.

Sintaxe:

```
sp_procoption69 [ @ProcName = ] 'procedure'70
, [ @OptionName = ] 'option'71
, [ @OptionValue = ] 'value'72
```

Para iniciar ou cancelar a execução automática, deve-se usar o procedimento de sistema *sp\_procoption*, que recebe três argumentos: Nome do procedimento, opção que é sempre startup e valor. O código abaixo configura um suposto procedimento de nome Proc\_teste para inicialização automática:

---

<sup>69</sup> Sets stored procedure for autoexecution. A stored procedure that is set to autoexecution runs every time an instance of SQL Server is started.

<sup>70</sup> Is the name of the procedure for which to set an option. *procedure* is nvarchar(776), with no default.

<sup>71</sup> Is the name of the option to set. The only value for *option* is startup.

<sup>72</sup> Is whether to set the option on (true or on) or off (false or off). *value* is varchar(12), with no default.

```
EXEC sp_procoption 'teste', startup, true
```

Para desabilitar a execução automática, basta alterar o último argumento para False:

```
EXEC sp_procoption 'teste', startup, False
```

### 4.3.5 Obter o código fonte de um procedimento

É possível obter o código fonte (definição) de um procedimento, desde que o mesmo não esteja encriptado, através do procedimento *sp\_helptext*. Este procedimento recebe como argumento o nome do procedimento:

```
USE ProDados
```

```
EXEC sp_helptext pr_NomeCompleto
```

### 4.3.6 Retorno de dados, parâmetros OUTPUT e RETURN

No SQL Server um procedimento pode devolver valores ao cliente de três formas diferentes:

1. Um ou mais conjuntos de resultados provenientes de consultas realizadas no início do procedimento;
2. Parâmetro OUTPUT: um parâmetro de saída (OUTPUT) pode devolver um valor ao utilizador, além do que pode ainda ser utilizado para passar um valor para o procedimento, como um parâmetro de entrada.
3. Return: return é um parâmetro especial, utilizado para devolver um número inteiro ao utilizador. Normalmente é utilizado para tratamento de erros ou para verificar determinadas condições na execução do procedimento.

#### Exemplo 21 (c/ OUTPUT)

```
Use ProDados
```

```
GO
```

```
CREATE PROC pr_Testes_OUTPUT @Valor1 int, @Valor2 int OUTPUT AS
```

```
Print '-- Inicio PROC --'
```

```
    Print @Valor1
```

```
    Print @Valor2
```

```
    Set @Valor2 = @Valor1 * @Valor2
```

```
Print '-- Fim PROC --'
```

O procedimento *pr\_Testes\_OUTPUT* recebe dois parâmetros, um de entrada o outro de saída. Ao lermos o parâmetro de saída (*@Valor2*) após a execução do procedimento, ele deve ter a multiplicação dos valores dos dois parâmetros.

Observe no exemplo a execução do procedimento e a leitura do parâmetro de saída:

```
Declare @A int
```

```
SET @A = 4
```

```
Exec pr_Testes_OUTPUT 2, @A OUTPUT
```

```
Print @A
```

### Exemplo 22 (c/ RETURN)

```
Create PROC pr_Teste_RETURN @Valor1 int, @Valor2 int AS
    Print @Valor1
    Print @Valor2
    Set @Valor2 = @Valor1 * @Valor2
    Return @Valor2

Declare @A int
Exec @A = pr_Teste_RETURN 2, 5
Print @A
```

## 4.3.7 Aplicação prática de procedimentos

### Exemplo 23

Pretende-se criar um procedimento que realize uma contagem dos produtos existentes por categoria do produto. Guarde o procedimento com o nome “pr\_ConsultaCategorias”

```
USE ProDados;
GO
CREATE Proc pr_ConsultaCategorias @CN nvarchar(30)=null AS
    IF @CN is null
    Begin
        Select C.CategoryID, CategoryName,
            count(P.ProductID) 'Total Produtos'
        From Categories C
        JOIN Products P ON C.CategoryID=P.categoryID
        Group By C.CategoryID, CategoryName
    END
    Else
    Begin
        IF Exists (Select CategoryName From categories
            Where CategoryName = @CN)
        Begin
            Select C.CategoryID, CategoryName,
                count(P.ProductID) 'Total Produtos'
            From Categories C
            JOIN Products P ON C.CategoryID=P.categoryID
            Where CategoryName = @CN
            Group By C.CategoryID, CategoryName
        End
        Else
        Print 'Categoria não existe'
    End
GO

EXEC pr_ConsultaCategorias 'Seafood';
```

| Results |            | Messages     |                |
|---------|------------|--------------|----------------|
|         | CategoryID | CategoryName | Total Produtos |
| 1       | 8          | Seafood      | 12             |

### Exemplo 24

```
-- Devolução de parametro
-- Parametro Output
USE ProDados;
GO
CREATE PROC pr_GetProductName @ProductID CHAR(10), @ProductName
VARCHAR(25) OUTPUT
AS SELECT @ProductName = P.ProductName FROM dbo.Products P
WHERE ProductID = @ProductID;
GO

USE ProDados;
DECLARE @ProdName VARCHAR(25);
EXEC pr_GetProductName '58', @ProdName OUTPUT;
PRINT @ProdName;
```

| Messages |                        |
|----------|------------------------|
|          | Escargots de Bourgogne |

### Exemplo 25

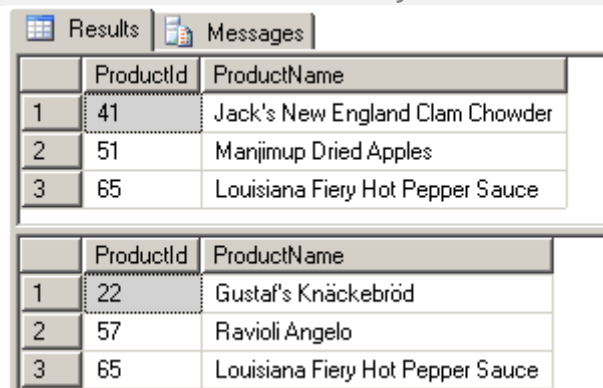
```
-- Retornar o montante total de vendas para um customerID
Create Proc pr_MT @CustomerID nchar(6), @MT money OUTPUT,
@Name nchar(30) OUTPUT
AS
Print @CustomerID
Select @Name=CompanyName, @MT = sum(Quantity * UnitPrice)
From Customers C
JOIN Orders O ON C.CustomerID = O.CustomerID
JOIN [Order Details] OD ON O.OrderID = OD.OrderID
Where C.CustomerID=@CustomerID
Group By CompanyName
GO

Declare @M money, @N nchar(30)
EXEC pr_MT 'BOLID', @M OUTPUT, @N OUTPUT
Print 'Montante total de vendas : ' + ltrim(convert(nchar(15),@M))
Print 'Cliente : ' + @N
```

## Exemplo 26

```
-- Criação do procedimento "pr_ProdutosClienteData".
-- Lista dos produtos encomendados pelo Cliente Y no dia X
USE ProDados
GO
CREATE PROCEDURE pr_ProdutosClienteData @CodCli NCHAR(5), @DataEnc
DateTime
AS SELECT OD.ProductId, P.ProductName
FROM Orders O
INNER JOIN [Order Details] OD ON O.OrderId = OD.OrderId
INNER JOIN Products P ON OD.ProductID = P.ProductID
WHERE O.CustomerId = @CodCli AND O.OrderDate = @DataEnc
GO

-- Execução do procedimento "ProdutosClienteData"
EXECUTE pr_ProdutosClienteData 'HANAR', '1996-07-08'
EXECUTE pr_ProdutosClienteData 'VICTE', '1996-07-08'
```



The screenshot shows the SQL Server Enterprise Manager interface with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying two separate query result sets. Each result set is a table with two columns: 'ProductId' and 'ProductName'. The first result set, corresponding to the first EXECUTE statement, contains three rows: (41, 'Jack's New England Clam Chowder'), (51, 'Manjimup Dried Apples'), and (65, 'Louisiana Fiery Hot Pepper Sauce'). The second result set, corresponding to the second EXECUTE statement, contains three rows: (22, 'Gustaf's Knäckebröd'), (57, 'Ravioli Angelo'), and (65, 'Louisiana Fiery Hot Pepper Sauce').

|   | ProductId | ProductName                      |
|---|-----------|----------------------------------|
| 1 | 41        | Jack's New England Clam Chowder  |
| 2 | 51        | Manjimup Dried Apples            |
| 3 | 65        | Louisiana Fiery Hot Pepper Sauce |

|   | ProductId | ProductName                      |
|---|-----------|----------------------------------|
| 1 | 22        | Gustaf's Knäckebröd              |
| 2 | 57        | Ravioli Angelo                   |
| 3 | 65        | Louisiana Fiery Hot Pepper Sauce |

```
-- Alteração do procedimento "pr_ProdutosClienteData"
USE ProDados
GO
ALTER PROCEDURE pr_ProdutosClienteData @CodCli NCHAR(5), @DataEnc1
DateTime, @DataEnc2 DateTime
AS SELECT OD.ProductId, P.ProductName
FROM Orders O
INNER JOIN [Order Details] OD ON O.OrderId = OD.OrderId
INNER JOIN Products P ON OD.ProductID = P.ProductID
WHERE O.CustomerId = @CodCli AND O.OrderDate
BETWEEN @DataEnc1 AND @DataEnc2
GO

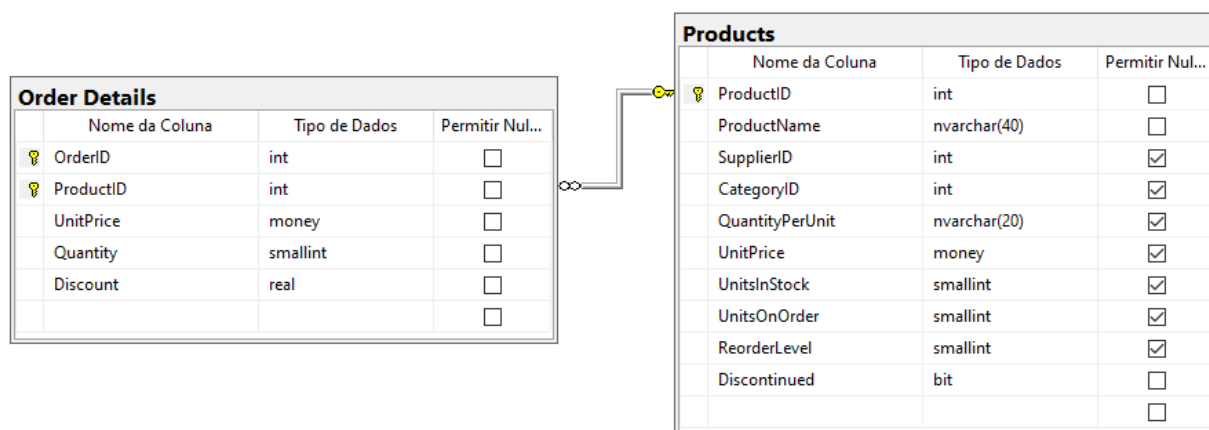
EXECUTE pr_ProdutosClienteData 'VINET', '1996-07-04', '1996-09-02'
```

|   | ProductId | ProductName                   |
|---|-----------|-------------------------------|
| 1 | 11        | Queso Cabrales                |
| 2 | 42        | Singaporean Hokkien Fried Mee |
| 3 | 72        | Mozzarella di Giovanni        |
| 4 | 71        | Flotemysost                   |
| 5 | 72        | Mozzarella di Giovanni        |
| 6 | 56        | Gnocchi di nonna Alice        |

## Exemplo 27

Crie um procedimento denominado “pr\_BackupDados”, que permita copiar os dados das tabelas “Products” e “Orders” para duas tabelas temporárias globais “##ProductsTemp” e “##OrdersTemp”.

## Exemplo 28



Considere a estrutura parcial de relações entre tabelas da base de dados ProDados

Pretende-se que construa um procedimento (pr\_CompararProdutos), associado à base de dados ProDados, que permita comparar para dois produtos, a quantidade vendida e o montante total de vendas. O procedimento deve aceitar dois argumentos de entrada, neste caso o ProductID de cada produto, devendo antes de listar a informação pretendida verificar se estes existem na base de dados. O procedimento deve indicar se ambos os produtos existem, e caso algum não exista, o procedimento não deve continuar.

Considere os resultados da execução do procedimento, apresentados de seguida, para apoio à sua construção:

**EXEC** pr\_CompararProdutos 5, 25

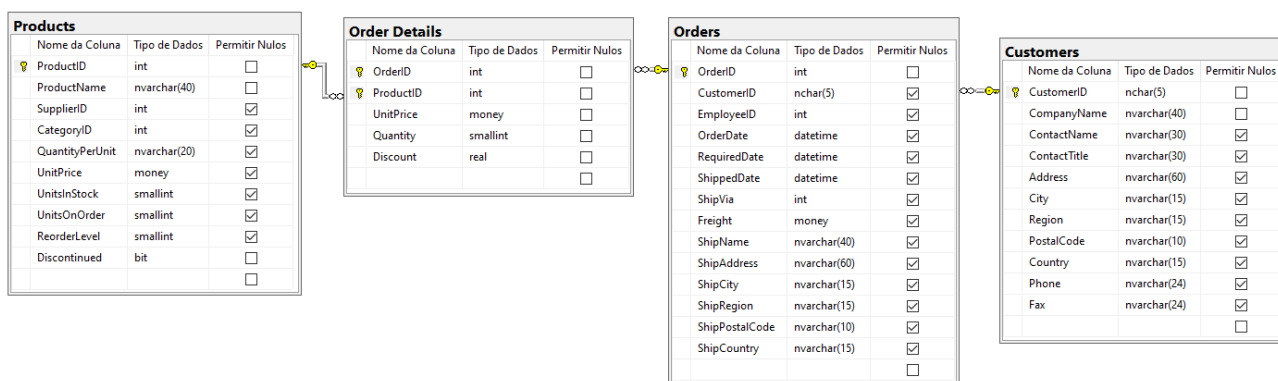
### Mensagens

Produto 1: 5 Existe  
Produto 2: 25 Existe

| ProductName             | Quantidade Vendida | Montante Total |
|-------------------------|--------------------|----------------|
| Chef Anton's Gumbo Mix  | 298                | 5801.15        |
| NuNuCa Nuß-Nougat-Creme | 318                | 4051.60        |



## Exemplo 29



Estrutura parcial de relações entre tabelas da base de dados ProDados

Pretende-se que construa um procedimento (pr\_TopClientes), associado à base de dados ProDados, que permita obter os melhores clientes de um determinado produto, considerando como critério o total das unidades compradas pelo cliente. O procedimento deve aceitar como argumentos de entrada o ProductID do produto e quantos clientes se pretende listar (Ex: listar os cinco melhores clientes – TOP 5). Utilize o exemplo seguinte com resultados do procedimento, como base de conhecimento para a sua construção.

**EXEC pr\_TopClientes 2,3**

Mensagens

Produto - Chang

Quantidade total vendida - 1057

| CompanyName        | Country | Quantidade Total | %         |
|--------------------|---------|------------------|-----------|
| Save-a-lot Markets | USA     | 203              | 0.1920530 |
| QUICK-Stop         | Germany | 121              | 0.1144749 |
| Folk och få HB     | Sweden  | 65               | 0.0614948 |

## 4.4 Triggers

Podemos criar *triggers* DML ou DDL<sup>73</sup>. Um *Trigger* é muito semelhante a um procedimento que é automaticamente ativado quando ocorre determinado evento na base de dados. *Triggers* DML são executados quando um utilizador INSERE, ALTERA ou ELIMINA registos numa tabela ou consulta. *Triggers* DDL<sup>74</sup> são executados como resposta a variadíssimos eventos de *Data Definition Language* (DDL), nomeadamente instruções CREATE, ALTER, e DROP realizados na base de dados ou no servidor.

Triggers DML são normalmente utilizados para reforçar as restrições existentes assim como a integridade dos dados na base de dados.

Este capítulo vai focar-se sobre os *triggers* DML. Este tipo de *trigger* pode ser ativado imediatamente antes (BEFORE) ou imediatamente depois (AFTER) de cada evento. Ou seja estes podem ser classificados como:

1. AFTER Triggers, os quais são ativados depois do evento (Insert, Delete, Update) realizado sobre um campo ou uma tabela. AFTER triggers nunca são ativados se uma restrição for violada; ou seja estes não podem ser usados para evitar violação de restrições;
2. INSTEAD OF Triggers, são ativados em vez do processamento do evento (Insert, Delete, Update) sobre o campo ou sobre a tabela e antes da verificação das restrições (constraints). Se existirem AFTER Triggers na tabela, eles são ativados depois dos INSTEAD OF Triggers. Se uma restrição for violada o INSTEAD OF Trigger não será finalizado.

Cada tabela ou view pode ter um INSTEAD OF Trigger associado a cada evento (UPDATE, DELETE, and INSERT). Por outro lado uma tabela pode ter vários AFTER triggers por cada evento.

---

<sup>73</sup> DML and DDL triggers can be created in the SQL Server 2005 Database Engine directly from Transact-SQL statements or from methods of assemblies that are created in the Microsoft .NET Framework common language runtime (CLR) and uploaded to an instance of SQL Server. SQL Server allows for creating multiple triggers for any specific statement.

<sup>74</sup> DDL triggers can fire in response to a Transact-SQL event processed in the current database, or on the current server. The scope of the trigger depends on the event. For example, a DDL trigger created to fire in response to a CREATE TABLE event will do so whenever a CREATE TABLE event occurs in the database. A DDL trigger created to fire in response to a CREATE LOGIN event will do so whenever a CREATE LOGIN event occurs in the server. DDL triggers, like regular triggers, fire stored procedures in response to an event. However, unlike DML triggers, they do not fire in response to UPDATE, INSERT, or DELETE statements on a table or view. Instead, they fire in response to a variety of Data Definition Language (DDL) statements. These statements are primarily statements that start with CREATE, ALTER, and DROP. DDL triggers can be used for administrative tasks such as auditing and regulating database operations. Use DDL triggers when you want to do the following:

- You want to prevent certain changes to your database schema.
- You want something to occur in the database in response to a change in your database schema.
- You want to record changes or events in the database schema.

A tabela seguinte compara o funcionamento dos triggers do tipo AFTER e INSTEAD OF.

| Function   | AFTER trigger  | INSTEAD OF trigger  |
|--|--|---|
| Applicability  | Tables   | Tables and views  |
| Quantity per table or view   | Multiple per triggering action (UPDATE, DELETE, and INSERT)  | One per triggering action (UPDATE, DELETE, and INSERT)  |
| Cascading references   | No restrictions apply  | INSTEAD OF UPDATE and DELETE triggers are not allowed on tables that are targets of cascaded referential integrity constraints. |
| Execution  | After:<br>Constraint processing<br>Declarative referential actions<br>inserted and deleted tables<br>creation<br>The triggering action | Before:<br>Constraint processing<br>In place of: The triggering action<br>After: inserted and deleted tables<br>creation        |
| Order of execution   | First and last execution may be specified  | Not applicable  |
| varchar(max), nvarchar(max), and varbinary(max) column references in inserted and deleted tables | Allowed  | Allowed   |
| text, ntext, and image column references in inserted and deleted tables                          | Not allowed  | Allowed   |

**Tabelas INSERTED<sup>75</sup> e DELETED<sup>76</sup>** - As tabelas *inserted* e *deleted* são criadas pelo SQL Server durante a execução de um Trigger, e contêm, respetivamente, os dados inseridos ou excluídos da tabela. Estão residentes apenas em memória temporária, e os dados não podem ser alterados.

**Verificar qual o campo que foi alterado** - Num trigger ativado durante uma atualização, é interessante saber se um determinado campo foi alterado, ou ainda, quais os campos que foram alterados. O SQL Server dispõe de duas funções com este fim:

- IF UPDATE*<sup>77</sup> (nome\_da\_coluna) para saber se determinada coluna sofreu alteração, e
- COLUMNS\_UPDATED()*<sup>78</sup> para saber quais as colunas que foram alteradas

<sup>75</sup> The **inserted** table stores copies of the affected rows during INSERT and UPDATE statements. During an insert or update transaction, new rows are added at the same time to both the **inserted** table and the trigger table. The rows in the **inserted** table are copies of the new rows in the trigger table.

<sup>76</sup> The **deleted** table stores copies of the affected rows during DELETE and UPDATE statements. During the execution of a DELETE or UPDATE statement, rows are deleted from the trigger table and transferred to the **deleted** table. The **deleted** table and the trigger table ordinarily have no rows in common.

<sup>77</sup> Returns a Boolean value that indicates whether an INSERT or UPDATE attempt was made on a specified column of a table or view. UPDATE() is used anywhere inside the body of a Transact-SQL INSERT or UPDATE trigger to test whether the trigger should execute certain actions.

<sup>78</sup> Returns a varbinary bit pattern that indicates the columns in a table or view that were inserted or updated. COLUMNS\_UPDATED is used anywhere inside the body of a Transact-SQL INSERT or UPDATE trigger to test whether the trigger should execute certain actions.

**Ordem de execução** - Podemos ter mais de um Trigger After associado a um mesmo evento de uma mesma tabela. É possível determinar qual será o trigger a ser activado em primeiro (first) ou em último (last). Os demais (ordem none) serão todos executados entre o primeiro e o último sem uma ordem específica. Para determinar a ordem de um trigger usamos a System Procedure *sp\_settriggerorder*, que recebe três argumentos: *Nome do trigger*, *ordem (first, last ou none)* e *evento (update, insert ou delete)*. Trigger do tipo *Instead Of* não podem ter uma ordem de execução definida.

**Número de registos afetados** - Podemos determinar o número de registos afetados por um trigger através da variável global @@ROWCOUNT.

Sintaxe (DML Trigger):

```
CREATE TRIGGER [ schema_name . ]trigger_name79
ON { table80 | view }
[WITH <dml_trigger_option81> [ ,...n ] ]
{FOR | AFTER82 | INSTEAD OF83 }
{[ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[WITH APPEND84 ]
[NOT FOR REPLICATION85 ]
AS { sql_statement86 [ ; ] [ ...n ] | EXTERNAL NAME <method
specifier [ ; ] > }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS87 Clause ]
```

As seguintes instruções Transact-SQL **não podem ser utilizadas** em triggers DML: *ALTER DATABASE*, *CREATE DATABASE*, *DROP DATABASE*, *LOAD DATABASE*, *LOAD LOG*, *RECONFIGURE*, *RESTORE DATABASE*, *RESTORE LOG*.

---

<sup>79</sup> Is the name of the trigger. A *trigger\_name* must comply with the rules for identifiers, with the exception that *trigger\_name* cannot start with # or ##.

<sup>80</sup> Is the table or view on which the DML trigger is executed

<sup>81</sup> Encrypts the text of the CREATE TRIGGER statement. Using WITH ENCRYPTION prevents the trigger from being published as part of SQL Server replication

<sup>82</sup> Specifies that the DML trigger is fired only when all operations specified in the triggering SQL statement have executed successfully. AFTER is the default when FOR is the only keyword specified. AFTER triggers cannot be defined on views.

<sup>83</sup> Specifies that the DML trigger is executed *instead of* the triggering SQL statement, therefore, overriding the actions of the triggering statements. INSTEAD OF cannot be specified for DDL triggers.

<sup>84</sup> Specifies that an additional trigger of an existing type should be added

<sup>85</sup> Indicates that the trigger should not be executed when a replication agent modifies the table that is involved in the trigger

<sup>86</sup> Is the trigger conditions and actions. Trigger conditions specify additional criteria that determine whether the tried DML or DDL statements cause the trigger actions to be performed.

<sup>87</sup> Specifies the security context under which the trigger is executed. Enables you to control which user account the instance of SQL Server uses to validate permissions on any database objects that are referenced by the trigger.

### Exemplo 30

```
-- TRIGGER para compreensão dos eventos associados
USE ProDados
go
Create Trigger tr_Eventos ON Products AFTER Insert, Update, Delete
AS
    Declare @ID as varchar(50)
    IF (Select ProductID From INSERTED) <> ''
        and (Select ProductID From DELETED) <> ''
    Begin
        Print 'Update -----'
        SET @ID = (Select ProductID From INSERTED)
        Print 'ID Produto Alterado -> ' + @ID
    End
    ELSE
    Begin
        IF (Select ProductID From INSERTED) <> ''
        Begin
            Print 'Insert -----'
            SET @ID = (Select ProductID From INSERTED)
            Print 'ID Produto Inserido -> ' + @ID
        End
        ELSE
        Begin
            Print 'Delete -----'
            SET @ID = (Select ProductID From DELETED)
            Print 'ID Produto Eliminado -> ' + @ID
        End
    End
End
```

Instruções para testar do Trigger:

```
Insert Into Products (ProductName,SupplierID,CategoryID)
Values('Producto Nacional',10,1)
--
Update Products Set ProductName = 'Bacalhau Nacional' Where
ProductID = 80
--
Delete From Products Where ProductID = 80
```

### Exemplo 31

Embora o SQL Server ofereça ótimos mecanismos de auditoria, podemos utilizar *triggers* para rastrear qualquer tipo de alteração na base de dados e criar o nosso próprio sistema de auditoria.

Suponha que o diretor de uma empresa quer saber que utilizadores têm realizado alterações aos preços dos produtos na base de dados da empresa. É preciso ainda saber quando e de onde (estação) esta alteração foi feita. Vamos utilizar como cenário a Base de Dados ProDados, tabela *Products*. Esta tabela possui um campo *UnitPrice*, onde esta armazenado o preço atual do produto. Pretende-se assim criar um *trigger After Update*, que será

chamado de “AuditaPreçoProduto”. É também necessário uma tabela para armazenar o processo de auditoria, sendo que o *trigger* deve verificar se esta tabela existe, e em caso negativo, criá-la.

USE ProDados

Create Table Auditoria

```
(IDResgisto Int Identity(0,1) NOT NULL Primary Key,  
Data datetime NOT NULL Default getdate(),  
Utilizador nchar(30) NOT NULL Default suser_name(),  
EstaçãoTrabalho nchar(30) Default host_name(),  
IDProduto int NOT NULL,  
PreçoUnitárioNovo money NOT NULL,  
PreçoUnitárioAntigo money NOT NULL)
```

Os campos *Utilizador*, *EstaçãoTrabalho* e *Data* apresentam como valor *default*, respetivamente, as funções *suser\_name()*, *host\_name()* e *getdate()*. *Suser\_name* obtém o utilizador da secção iniciada, *Host\_name()* o nome da estação de trabalho e *GetDate()* a data e hora corrente. Deste modo estes valores podem ser omitidos no momento de criar um novo registo na tabela *Auditoria*.

Em seguida temos que obter das tabelas *Inserted* e *Deleted* o novo e antigo preço do produto, respetivamente, e ainda o IDProduto respectivo, para sabermos qual foi o produto alterado, para isso é necessário declarar três variáveis locais: @ID, @PrecoNovo e @PrecoAntigo:

```
DECLARE @ID INT, @PrecoNovo MONEY, @PrecoAntigo MONEY
```

Em seguida atribuímos a estas variáveis os valores das tabelas *Inserted* e *Deleted*:

```
SELECT @ID = (SELECT ProductID FROM Deleted), @PrecoNovo =(SELECT  
UnitPrice FROM Inserted), @PrecoAntigo = (SELECT UnitPrice FROM  
Deleted)
```

Finalmente inserimos o registo na tabela de auditoria

```
INSERT INTO Auditoria  
(IDProduto,PreçoUnitárioNovo,PreçoUnitárioAntigo)  
VALUES (@ID,@PrecoNovo,@PrecoAntigo)
```

Um último detalhe: o trigger vai ser ativado sempre que a tabela *Products* for alterada, mas só nos interessa auditar registos que tenham o campo *UnitPrice* alterado, para isso, utilizamos a função *if update* para realizar esta verificação, e só então executar a auditoria:

```
CREATE TRIGGER tr_AuditaPreçoProduto ON Products AFTER UPDATE AS  
IF UPDATE(UnitPrice)  
BEGIN  
SET NOCOUNT ON88  
DECLARE @ID INT, @PrecoNovo MONEY, @PrecoAntigo MONEY  
SELECT @ID = (SELECT ProductID FROM deleted),  
@PrecoNovo = (SELECT UnitPrice FROM inserted), @PrecoAntigo
```

---

<sup>88</sup> Stops the message that shows the number of rows affected by a Transact-SQL statement from being returned as part of the results.

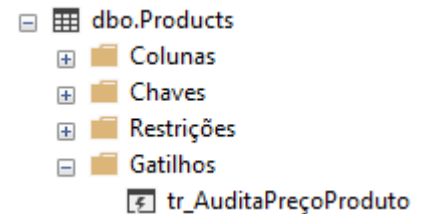
```

= (Select UnitPrice FROM deleted)

INSERT INTO Auditoria
(IDProduto,PreçoUnitárioNovo,PreçoUnitárioAntigo)
VALUES (@ID,@PrecoNovo,@PrecoAntigo)
END

```

O Trigger criado fica associado à tabela *Products*, como representado na imagem.



Para testar o seu funcionamento altere alguns preços de produtos e depois leia o conteúdo da tabela Auditoria.

```

update Products set UnitPrice = UnitPrice + 1 where ProductID = 1
update Products set UnitPrice = UnitPrice + 1 where ProductID = 2
update Products set UnitPrice = UnitPrice + 1 where ProductID = 3

Select * from Auditoria

```

Se o triggers não apresentar erros, este deve ser activado sempre que existir alguma alteração na tabela.campo *Products.UnitPrice*, registrando o evento na tabela Auditoria:

|   | IDResgisto | Data                    | Utilizador | EstaçãoTrabalho | IDProduto | PreçoUnitárioNovo | PreçoUnitárioAntigo |
|---|------------|-------------------------|------------|-----------------|-----------|-------------------|---------------------|
| 1 | 0          | 2009-12-02 16:06:18.123 | dbo        | ESTG-54A90C6508 | 1         | 18,10             | 18,00               |
| 2 | 1          | 2009-12-02 16:06:21.780 | dbo        | ESTG-54A90C6508 | 2         | 19,02             | 19,00               |
| 3 | 2          | 2009-12-02 16:06:25.810 | dbo        | ESTG-54A90C6508 | 3         | 9,20              | 10,00               |

Para listar os triggers associados a uma tabela assim como o seu tipo e funcionalidade:

```
EXEC sp_helptrigger 'Products'
```

|   | trigger_name       | trigger_owner | isupdate | isdelete | isinsert | isafter | isinsteadof | trigger_schema |
|---|--------------------|---------------|----------|----------|----------|---------|-------------|----------------|
| 1 | AuditaPreçoProduto | dbo           | 1        | 0        | 0        | 1       | 0           | dbo            |

Para listar o conteúdo/instruções do trigger:

```
EXEC sp_helptext 'tr_AuditaPreçoProduto'
```

Para listar os objetos relacionados e dependentes da tabela *Products*, tais como: *views*, *check constraints*, *procedures* e *triggers*.

```
EXEC sp_depends @objname = N'Products'
```

### Exemplo 32

Pretende-se auditar a criação de novos Clientes (Customers), Produtos (Products) e Fornecedores (Suppliers) da base de dados ProDados. Atribua o nome “tr\_AuditarNovosRegistos” ao trigger.

É necessário uma tabela para armazenar o processo de auditoria, registando a atividade (*INSERT*) nas tabelas indicadas.

```
-- Tabela de apoio ao processo/TRIGGER
```

```
USE ProDados
```

```
GO
```

```
CREATE TABLE AuditarTabelas
```

```
(
```

```
    IDRegisto INT PRIMARY KEY IDENTITY (1,1),
```

```
    DataReg DATETIME NOT NULL DEFAULT GETDATE(),
```

```
    Operação VARCHAR(10) NOT NULL,
```

```
    Registo VARCHAR(50) NOT NULL,
```

```
    Tabela VARCHAR(20) NOT NULL
```

```
)
```

```
-- TRIGGER para a tabela Customers
```

```
GO
```

```
CREATE TRIGGER tr_AuditaNovosRegisto ON Customers
```

```
FOR INSERT
```

```
AS
```

```
    DECLARE @Codigo VARCHAR(50)
```

```
    SELECT @Codigo = INSERTED.CustomerID FROM INSERTED
```

```
    INSERT INTO AuditarTabelas
```

```
        VALUES (GETDATE(), 'INSERT', @Codigo, 'Customers')
```

```
-- Teste do TRIGGER para a tabela Customers
```

```
INSERT INTO Customers (CustomerId, CompanyName)
```

```
    VALUES ('XPT01', 'Nome do XPT01')
```

```
INSERT INTO Customers (CustomerId, CompanyName)
```

```
    VALUES ('XPT02', 'Nome do XPT02')
```

```
INSERT INTO Customers (CustomerId, CompanyName)
```

```
    VALUES ('XPT03', 'Nome do XPT03')
```

```
INSERT INTO Customers (CustomerId, CompanyName)
```

```
    VALUES ('XPT04', 'Nome do XPT04')
```

```
GO
```

```
SELECT * FROM AuditarTabelas
```

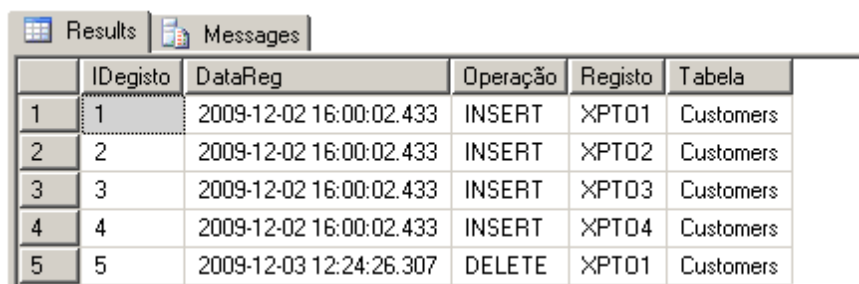
| Results |           | Messages                |          |         |           |
|---------|-----------|-------------------------|----------|---------|-----------|
|         | IDRegisto | DataReg                 | Operação | Registo | Tabela    |
| 1       | 1         | 2009-12-02 16:00:02.433 | INSERT   | XPT01   | Customers |
| 2       | 2         | 2009-12-02 16:00:02.433 | INSERT   | XPT02   | Customers |
| 3       | 3         | 2009-12-02 16:00:02.433 | INSERT   | XPT03   | Customers |
| 4       | 4         | 2009-12-02 16:00:02.433 | INSERT   | XPT04   | Customers |



### Exemplo 33

Pretende-se auditar a eliminação de registos de Clientes (Customers), Produtos (Products) e Fornecedores (Suppliers) da base de dados Prodados. Atribua o nome “tr\_AuditarRegistosEliminados” ao trigger.

```
-- Criação do Trigger
USE Prodados
GO
CREATE TRIGGER tr_AuditarRegistosEliminados ON Customers
FOR DELETE
AS
    DECLARE @Codigo AS VARCHAR(50)
    SELECT @Codigo = DELETED.CustomerID FROM DELETED
    INSERT INTO AuditarTabelas
        VALUES (GETDATE(), 'DELETE', @Codigo, 'Customers')
GO
-- Teste do Trigger
DELETE Customers WHERE CustomerID LIKE 'XPT0%'
GO
-- Verificação do resultado do Trigger
SELECT * FROM AuditarTabelas
```



|   | IDegisto | DataReg                 | Operação | Registo | Tabela    |
|---|----------|-------------------------|----------|---------|-----------|
| 1 | 1        | 2009-12-02 16:00:02.433 | INSERT   | XPT01   | Customers |
| 2 | 2        | 2009-12-02 16:00:02.433 | INSERT   | XPT02   | Customers |
| 3 | 3        | 2009-12-02 16:00:02.433 | INSERT   | XPT03   | Customers |
| 4 | 4        | 2009-12-02 16:00:02.433 | INSERT   | XPT04   | Customers |
| 5 | 5        | 2009-12-03 12:24:26.307 | DELETE   | XPT01   | Customers |

### Exemplo 34

O exemplo seguinte reúne num só trigger os exemplos anteriores. Ou seja, em vez de termos um trigger para o evento insert e outro para o evento delete, temos um único trigger para os dois eventos.

```
CREATE TRIGGER tr_AuditarRegistos ON Customers
After INSERT, DELETE as
Declare @Codigo as varchar(50)
IF (Select CustomerID from inserted) <> ''
    BEGIN
        Set @codigo = (Select CustomerID from inserted)
        INSERT INTO AuditarTabelas VALUES
            (getdate(), 'INSERT', @codigo, 'Customers')
    END
ELSE
    BEGIN
        Set @Codigo = (Select CustomerID from deleted)
```

```

        INSERT INTO AuditarTabelas VALUES
        (getdate(), 'DELETE', @codigo, 'Customers')
    END
go
-- Teste do Trigger
INSERT INTO Customers (CustomerID, CompanyName)
VALUES ('xpto1', 'nome do xpto1')

DELETE FROM Customers where customerid = 'xpto1'
Select * from AuditarTabelas

```

### Exemplo 35

```

USE ProDados
GO
Create TRIGGER tr_AuditarCategoriasProdutos ON Categories
    AFTER Insert, Update
    AS
    SET NoCount ON
    IF Update(CategoryName)
    SELECT 'Foi alterada a categoria ' + Inserted.CategoryName
    FROM Inserted
go
-- Teste do Trigger
UPDATE Categories SET CategoryName = 'Produce A'
WHERE CategoryName = 'Produce'

```

| Results |                                    | Messages |
|---------|------------------------------------|----------|
|         | (No column name)                   |          |
| 1       | Foi alterada a categoria Produce A |          |

```

INSERT categories(CategoryName) VALUES ('Papeleria')

```

| Results |                                    | Messages |
|---------|------------------------------------|----------|
|         | (No column name)                   |          |
| 1       | Foi alterada a categoria Papeleria |          |

### Exemplo 36

Pretende-se que o campo “LastName” da tabela Employees seja protegido, impedido que seja alterado (UPDATE). Ou seja uma vez registado não pode ser alterado.

```

USE ProDados
go
CREATE TRIGGER tr_Employees_LastName on Employees FOR UPDATE AS
    IF UPDATE(lastname)

```

```

BEGIN
    Print 'Não pode alterar este campo!'
    ROLLBACK TRAN89
    RETURN
END
GO

```

```

exec sp_helptext tr_Employees_LastName
exec sp_depends @objname =N'Employees'

```

```

UPDATE Employees SET LASTNAME = 'Davolio1' Where EmployeeID =1
Select * from Employees

```

### Exemplo 37

```

USE ProDados
-- Tabela de apoio ao exercicio.
CREATE TABLE Tabela_Exemplo
(
    ID      int IDENTITY(1,1) Primary Key,
    Custo   money NOT NULL,
    IVA     decimal(18,2) NOT NULL,
    Total   AS (Custo+(Custo * IVA))
)
GO

-- Registrar um novo registo na tabela de forma correcta.
INSERT INTO Tabela_Exemplo (Custo, IVA) VALUES (125, 0.21)
Select * from Tabela_Exemplo

-- Registrar um novo registo na tabela de forma NÃO correcta.
-- Dá um erro e o registo não é guardado na tabela
INSERT INTO Tabela_Exemplo (Custo, IVA, Total)
VALUES (125, 0.21, 167)
go

-- Criar uma View de acção
Create VIEW View_Tabela_Exemplo AS
    Select ID, Custo, IVA, Total From Tabela_Exemplo

```

---

<sup>89</sup> Rolls back an explicit or implicit transaction to the beginning of the transaction, or to a savepoint inside the transaction. A transaction cannot be rolled back after a COMMIT TRANSACTION statement is executed. If a ROLLBACK TRANSACTION is issued in a trigger:

- All data modifications made to that point in the current transaction are rolled back, including any made by the trigger.
- The trigger continues executing any remaining statements after the ROLLBACK statement. If any of these statements modify data, the modifications are not rolled back. No nested triggers are fired by the execution of these remaining statements.
- The statements in the batch after the statement that fired the trigger are not executed.

```

Select * from View_Tabela_Exemplo
-- Registrar um novo registo na tabela de forma NÃO correcta
-- através da VIEW.
-- Dá um erro e o registo não é guardado na tabela
INSERT INTO View_Tabela_Exemplo (Custo, IVA, Total)
VALUES (125, 0.21, 167)
go

-- Criar o Trigger
Create Trigger Trigger1 ON View_Tabela_Exemplo
INSTEAD OF Insert as
Begin
    Insert into Tabela_Exemplo (Custo, IVA)
    Select Custo, IVA From Inserted
End
GO

-- Registrar um novo registo na tabela de forma NÃO correcta
-- através da VIEW.
-- Com o funcionamento do Trigger não surge o erro e o
-- registo é inserido na tabela com sucesso
INSERT INTO View_Tabela_Exemplo (Custo, IVA, Total)
VALUES (125, 0.21, 167)
GO

```

### Exemplo 38

O Microsoft SQLServer dispõe de uma ferramenta (Database Mail) que após activada<sup>90</sup> e devidamente configurada permite que sejam enviadas mensagens para uma caixa de correio electrónico. Um trigger pode utilizar esta funcionalidade. Por exemplo, se for alterado o UnitPrice de um produto da tabela *Products* deve ser enviado um mail com a informação do produto e preço alterados.

```

USE ProDados
GO
CREATE TRIGGER tr_Products_UnitPrice ON Products AFTER UPDATE
AS
    DECLARE @sMsg VARCHAR(200),
            @NomeProduto NVARCHAR(40),
            @PreçoAntigo MONEY,
            @PreçoNovo MONEY

    IF UPDATE(UnitPrice)
    BEGIN
        SELECT @NomeProduto = d.ProductName,
               @PreçoAntigo = d.UnitPrice,
               @PreçoNovo = i.UnitPrice
    END

```

<sup>90</sup> Esta ferramenta é activada através da aplicação de configuração “SQL SERVER Surface Área Configuration”

```

FROM inserted i
    INNER JOIN deleted d ON i.ProductID = d.ProductID
SET @sMsg = 'O preço do produto - ' + @NomeProduto
    + ' - foi alterado de '
    + CONVERT(VARCHAR(10), @PreçoAntigo)
    + ' para ' + CONVERT(VARCHAR(10), @PreçoNovo)
    + ' em ' + CONVERT(VARCHAR(30), getdate()) + '.'

EXEC msdb.dbo.sp_send_dbmail
    @profile_name = 'slopes',
    @recipients = 'slopes@estgp.pt',
    @body = @sMsg,
    @subject = 'SQL Server Mail' ;

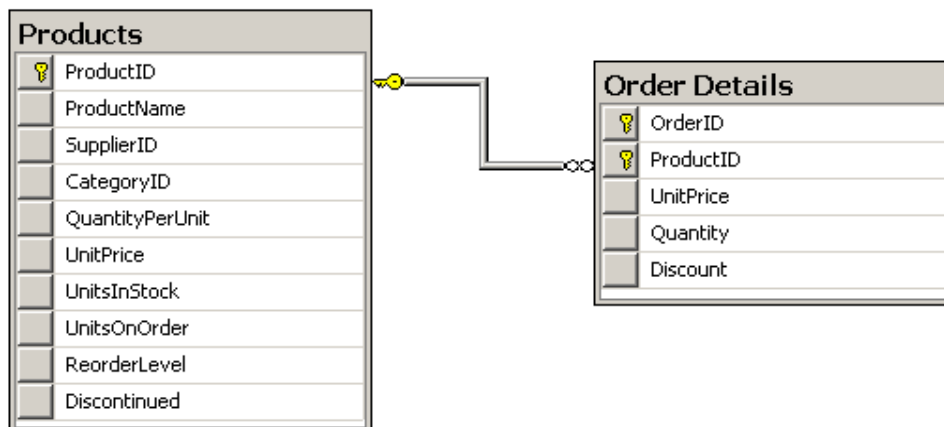
END
RETURN
GO

UPDATE Products SET UnitPrice = '18.00' Where ProductID =1

EXECUTE msdb.dbo.sysmail_help_profileaccount_sp

```

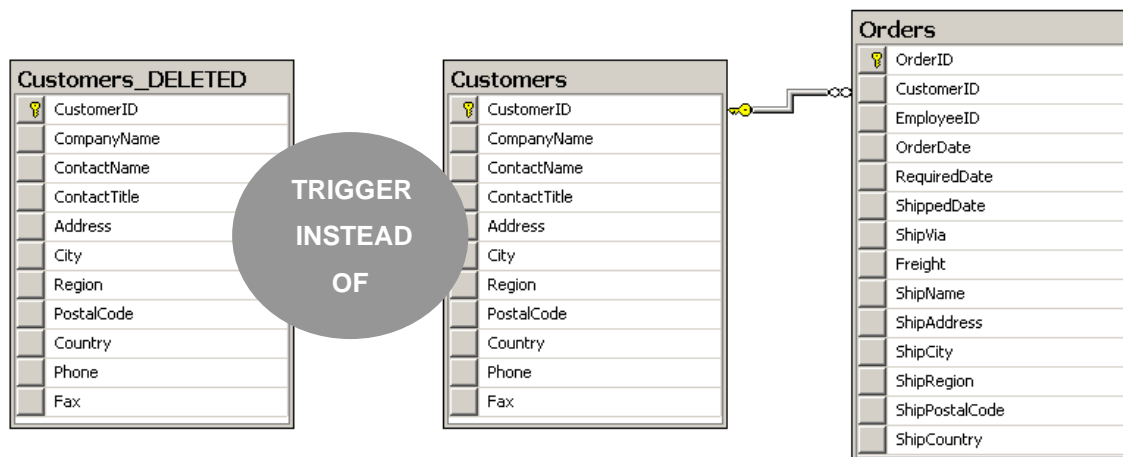
### Exemplo 39



Pretende-se que sempre que é registada ou alterada uma nova linha de encomenda (Order Details) que o valor do *UnitPrice* seja obtido na tabela *Products* tendo em conta o *ProductID* registado. Ou seja, o utilizador fica assim impedido de alterar este valor manualmente. Desenvolva um trigger que permita realizar este requisito, tanto para eventos de *insert* como de *update*.

Atribua ao trigger a designação `tr_ObterPreçoUnitario`.

## Exemplo 40



Considere a seguinte regra a criar na base de dados ProDados: Deve ser impedido qualquer utilizador de eliminar um registo de um cliente/customer sempre que este apresenta pelo menos um registo de vendas na tabela *Orders*. Caso contrário o registo pode ser eliminado e em simultâneo deve ser realizada uma cópia deste para a tabela *Customers\_DELETED* para, se necessário, utilização futura.

Desenvolva as instruções necessárias para o desenvolvimento com sucesso da regra apresentada. Atribua ao trigger a designação `tr_Customers_INSTEADOF_Delete`

Altere o trigger desenvolvido permitindo que apenas o utilizador “SA” possa eliminar registos de clientes/customers, mas mantendo a regra definida em funcionamento.

Desenvolva um procedimento que permita recuperar um registo eliminado, repondo a sua informação na tabela *Customers*. Este procedimento deve receber como parâmetro de entrada o valor do *CustomerID*. Atribua ao procedimento a designação `pr_RecuperarCustomer_DELETED`.

## Exemplo 41

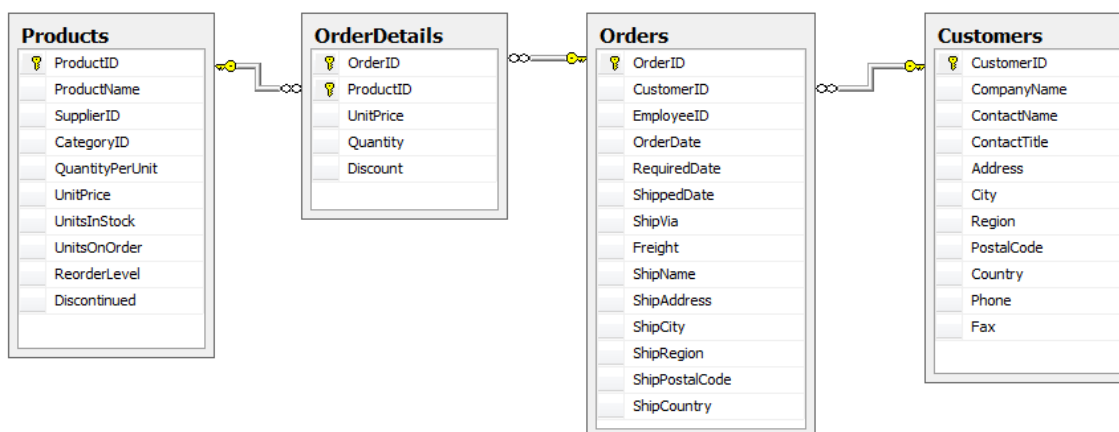
Adicione à estrutura da tabela *Products* um novo campo, o qual deve sinalizar se o registo pode ou não ser eliminado (valor “0” - o registo não pode ser eliminado, valor “1” - o registo pode ser eliminado).

```
Alter Table Products ADD IsDeleted nchar(1) not null default (0)
```

Desenvolva um trigger para associar à tabela *Products*, que apenas permita que o registo de um produto seja eliminado se o valor do campo *IsDeleted* for igual a “1”. Caso contrário o registo não pode ser eliminado, sendo anulada a transacção solicitada pelo utilizador.

Atribua ao trigger a designação `tr_Products_INSTEADOF_Delete`

## Exemplo 42



Considere a estrutura parcial de relações entre tabelas da base de dados ProDados. Pretende-se que construa um *trigger* (tr\_ActualizarValorStock), associado à base de dados ProDados, que permita atualizar o valor do campo Products.UnitsInStock sempre que acontecem na tabela Order Details eventos que o justifiquem.

Considere o possível resultado das seguintes instruções:

```
SET NoCount ON
Select ProductID, UnitsINStock from Products
Where ProductID in (10,11,12)
Select * from [Order Details] Where OrderID = 10248
```

| ProductID | UnitsINStock |
|-----------|--------------|
| 10        | 20           |
| 11        | 10           |
| 12        | 86           |

| OrderID | ProductID | UnitPrice | Quantity | Discount |
|---------|-----------|-----------|----------|----------|
| 10248   | 11        | 21,00     | 12       | 0        |
| 10248   | 42        | 9,80      | 10       | 0        |
| 10248   | 72        | 34,80     | 5        | 0        |

Associado à venda (10248) podem acontecer os seguintes eventos, que vão provocar alteração no valor do stock associado ao produto em causa:

- A quantidade do produto 11 pode ser alterada;
- O produto 11 pode ser substituído pelo produto 10;
- O produto 12 pode ser adicionado à encomenda;
- O produto 11 pode ser eliminado da encomenda;
- Toda a encomenda pode ser eliminada (Orders).

Desenvolva o Trigger que contemple os eventos descritos.

## 5. Abordagem Entidade/Associação

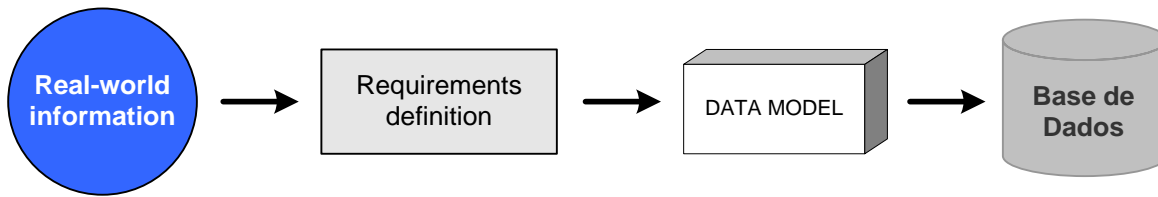


Figura 13 - Modelo de dados

Modelo de Dados:

- Visão dos dados em vez de visão das aplicações
- Eliminação de redundâncias
- Partilha de dados pelas aplicações

*Construir um modelo de dados é: Identificar, Analisar e Registrar a política da organização acerca dos dados*

A definição do modelo de dados é feita a dois níveis:

- Conceptual - Representação fiel da realidade, sem atender a quaisquer constrangimentos impostos pelo modelo informático.
- Físico – Adaptação do modelo conceptual às características do sistema informático.

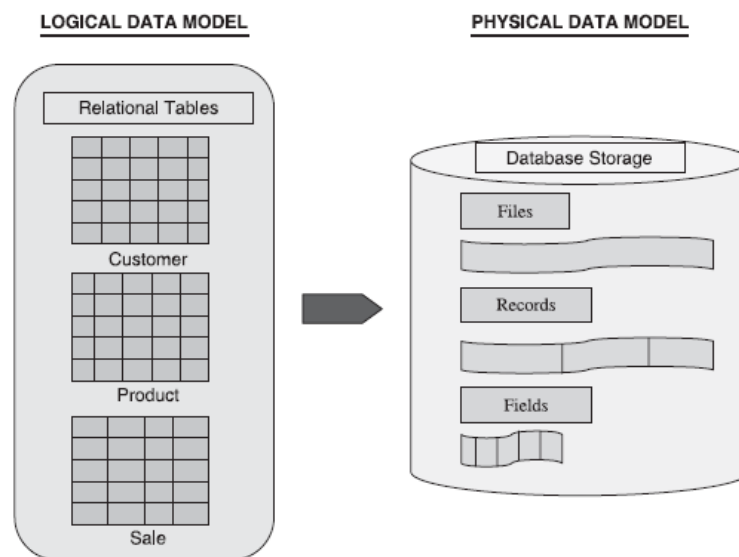


Figura 14 - Definição do modelo de dados

As técnicas de modelação dividem-se em dois grupos:

- Do particular para o geral (*Bottom-up*), adequada para pequenos projetos (6-8 tabelas). Parte da identificação dos níveis mais elementares (atributos) de informação e agrupa-os usando as relações de interdependência entre eles (dependências funcionais). Esta é a abordagem da Teoria da Normalização (Codd1970)



- Do geral para o particular (*Top –down*), adequado para grandes projetos. Parte dos grandes objetos de informação (entidades) identificando as suas inter-relações.
  1. Selecionar entidades e associações entre elas que tenham interesse para a organização.
  2. Especificar os atributos para cada entidade e associação. Esta é a abordagem do Modelo entidade – associações.

O modelo entidade -associação (*Entity –Relationship (ER)*) é especificado a dois níveis:

1. Gráfico  
Diagrama entidade – associação (ER)
2. Descritivo  
Especificações para cada componente do modelo (entidades, associações e atributos)

Esta abordagem foi proposta por P. Chen em 1976. Desde então muitas extensões e abordagens derivadas da mesma vêm sendo propostas sendo hoje correto constatar que se trata de uma "escola de pensamento" motivando inclusivamente a existência de uma conferência internacional realizada regularmente.

Surgiu esta "escola de pensamento" como contraponto à metodologia até então considerada predominante no que toca ao desenho de Bases de Dados relacionais inspirada na chamada teoria da normalização. A base fundamental da abordagem Entidade/Associação (daqui em diante designada abreviadamente pela sigla ER do inglês "*entity/relationship*"), é a de que, ao analisar um dado UoD<sup>91</sup> procura-se identificar nele um conjunto de EMRs<sup>92</sup> que se identificam com um conjunto de padrões predefinidos (estrutura de conceitos da abordagem).

Tudo se passa portanto como se o engenheiro de sistemas de informação encarregue de levar a cabo a fase de análise dispusesse de um "escantilhão" onde estão gravados os padrões relativos à estrutura de conceitos da abordagem em causa e procurasse de forma sistemática emparelhar esses padrões com entidades do mundo real existentes no UoD em questão.

Como resultado obtêm-se uma especificação (dita conceptual) com a definição das entidades efetivamente identificadas à custa dos padrões. Trata-se portanto de uma atitude mental de tipo "reconhecimento de padrões" com a qual o ser humano está bem familiarizado. Mas é uma atitude difícil de replicar por um instrumento computacional. Por essa razão esta é a etapa que é sempre feita por humanos no processo de desenvolvimento de Sistemas de Informação e a que será certamente a última a automatizar.

Apresentam-se de seguida e sucessivamente os vários elementos da estrutura de conceitos da abordagem ER tal como propostos inicialmente por Chen. Ao mesmo tempo vão sendo introduzidos fragmentos do UoD académico que servirão de exemplo ao longo de vários capítulos deste texto, sendo aqui o local onde o aluno se familiarizará com eles pela primeira vez.

Os principais conceitos desta abordagem são os de entidade, associação e atributo. São estes os principais padrões presentes no "escantilhão" proposto em [CHEN76].

---

<sup>91</sup> Universo do Discurso

<sup>92</sup> Entidades do mundo real

## 5.1 Entidade

Entidade é um conceito abstrato. Pretende representar a realidade que queremos modelar. A definição de uma entidade passa pela identificação dos elementos e de um conjunto de atributos comuns do mundo real que estamos a analisar. Por exemplo, considere um Instituto Politécnico. Nesse Instituto, e dependendo do objetivo a alcançar, podemos identificar, as entidades ALUNO, PROFESSOR, DISCIPLINA, etc.. A entidade ALUNO representa todos os alunos do Instituto e o aluno "José Pires" é uma instância da entidade ALUNO.

Imaginemos o seguinte fragmento de uma descrição textual informal do UoD académico (nele são omitidos por enquanto alguns aspetos menos relevantes para a presente discussão. Esses fragmentos omitidos são referenciados por "...").

**"Numa Universidade existem alunos e funcionários ..."**

Este fragmento do UoD, ainda que bastante pequeno, permite-nos identificar imediatamente duas EMRs e emparelhá-las com o respetivo padrão ER. São elas aluno (colocam-se sempre no singular as designações das EMRs) e funcionário. O padrão ER correspondente é designado por conjunto de entidades ("*entity set*"). Assim existirão os conjuntos de entidades designados por aluno e funcionário.

Designa-se por entidade ("*entity*") cada um dos elementos do conjunto de entidades. Assim, por exemplo, o aluno "António Silva" será uma entidade pertencente ao conjunto de entidades aluno. É vulgar utilizar a designação "entidade" quer se trate de uma entidade isolada ou de um conjunto de entidades por forma a aliviar o texto, deixando ao leitor a tarefa de distinguir, pelo contexto, de qual dos caso se trata em cada situação.

Existe uma simbologia gráfica, igualmente introduzida em [CHEN76] para expressar modelos ER. No caso do conjunto de entidades o símbolo adotado é um retângulo.



Figura 15 - Conjunto de entidades: aluno

Para cada entidade é necessário conhecer quais as propriedades que são relevantes para o sistema.

*Atributo - é qualquer propriedade de uma entidade. Um atributo é um elemento atómico (indivisível) de informação. Exemplos: N<sup>o</sup> de empregado, Nome, ...*

## 5.2 Associação, aridade e multiplicidade (tipos de associação)

Nesta secção apresentam-se os conceitos de conjunto de associações, associação, aridade e multiplicidade de associações.

### 5.2.1 Associação

Um conjunto de associações é definido como uma relação matemática entre n entidades. Considere-se o seguinte fragmento do UoD académico:

**"... Os alunos inscrevem-se em disciplinas..."**

Neste caso, inscrição pode ser definido como associação (relembre-se que aqui se trata de conjunto de associações) entre as entidades aluno e disciplina. As várias inscrições individuais (por exemplo do aluno A na disciplina X) são elementos daquele conjunto ou por outras palavras instâncias ou ocorrências da associação. O símbolo usado na abordagem ER para o conjunto de associações é um losango (ver Figura 17).

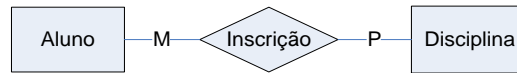


Figura 16 - Conjunto de associações: inscrição

**Associações**, são utilizadas para relacionar entidades. As entidades interagem umas com as outras, através de associações.

### 5.2.2 Aridade

Designa-se por aridade ("*arity*") da associação, o número  $n$  de entidades que nela participam. Assim designam-se por binárias as associações entre duas entidades, por ternárias as associações entre três entidades e em geral por  $n$ -árias as associações entre  $n$  entidades. Uma entidade que participa numa dada associação diz-se um argumento dessa associação.

Podem suceder casos em que uma dada entidade desempenhe o papel de mais do que um argumento de uma mesma associação. Considere-se, por exemplo, o seguinte fragmento da descrição textual do UoD académico:

"...certas disciplinas são consideradas precedências obrigatórias de outras, i.e. não é possível a um aluno ser aprovado numa sem ter sido previamente aprovado noutra. Sabe-se caso a caso quais as disciplinas que constituem precedência de outras..."

A definição ER da associação precedência é binária (i.e. cada instância da associação associa duas disciplinas: uma disciplina e a sua precedente) mas ambas as disciplinas presentes em cada associação pertencem ao mesmo conjunto de entidades: disciplina. Assim tem-se o diagrama da Figura 18.

Identicamente se passaria para o caso de associações de aridade superior a dois.

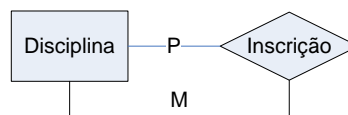


Figura 17 - Associação binária em que o mesmo conjunto de entidades desempenha o papel dos dois argumentos: precedência.

### 5.2.3 Multiplicidade (tipos de associação ou cardinalidade)

As associações classificam-se ainda quanto à multiplicidade. Identificam-se quanto a este aspeto as seguintes classes:

- Associações de muitos-para-muitos;
- Associações de muitos-para-um;
- Associações de um-para-um.

Esta classificação reflete a existência ou não de restrições relativamente ao número de entidades de um dado conjunto de entidades argumento da associação que podem estar relacionadas com entidades de outro conjunto também argumento da mesma associação. Por exemplo, considere-se o caso das inscrições dos alunos em disciplinas. Cada aluno poder-se-á inscrever em várias disciplinas e cada disciplina poderá receber as inscrições de vários alunos. Não há portanto qualquer restrição ao número de alunos que se podem inscrever numa disciplina nem quanto ao número de disciplinas em que um dado aluno se poderá inscrever. Estamos no caso de uma associação do tipo muitos-para-muitos.

Graficamente, a multiplicidade de uma associação indica-se através de um conjunto de símbolos (o número natural 1 e/ou letras indicando números naturais arbitrários) sobre os segmentos de reta que ligam a associação a cada um dos seus argumentos. Por exemplo a Figura 18 inclui as letras M e P para indicar que cada aluno pode participar em P inscrições e cada disciplina pode participar em M inscrições. Neste caso diz-se que a associação inscrição é de M (alunos) para P (disciplinas) i.e. de muitos-para-muitos.

No caso de cada elemento de um dos argumentos de uma associação estar impedido de participar em mais do que uma instância de associação mas os elementos do outro argumento poderem participar num qualquer número de instâncias da associação, então diz-se que se trata de uma associação de muitos-para-um. Considere-se o seguinte fragmento da descrição do UoD académico:

**"...cada disciplina é da responsabilidade de um só departamento mas cada departamento possui à sua responsabilidade mais do que uma disciplina..."**

A Figura 19 inclui a especificação gráfica da associação responsabilidade que é de muitos-para-um entre as entidades departamento e disciplina. Tal se justifica pelo facto de que cada disciplina só pode estar associada a um departamento (símbolo 1 junto ao ramo correspondente da associação), enquanto que cada departamento pode participar em M associações (símbolo M junto ao ramo correspondente da associação).

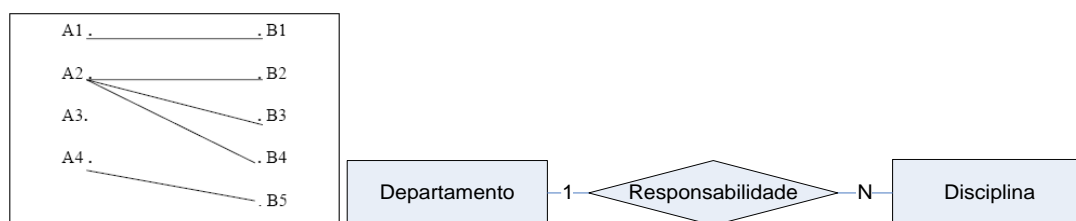


Figura 18 - Associação de muitos-para-um: responsabilidade

Considere-se agora o seguinte fragmento da descrição textual do UoD acadêmico:

**“... cada aluno de doutoramento efetua uma (e só uma) tese. Cada tese só tem um aluno como autor...”**

Este é o caso em que cada elemento de um dado conjunto de entidades (neste caso aluno de doutoramento) só pode estar associado a um elemento de outro conjunto de entidades (neste caso) e reciprocamente (i.e. cada tese também só pode estar relacionada com um aluno de doutoramento). Trata-se aqui de uma associação de um-para-um. Graficamente este tipo de associação representa-se colocando um símbolo 1 em cada um dos segmentos de reta que unem a associação aos seus argumentos (Figura 20).

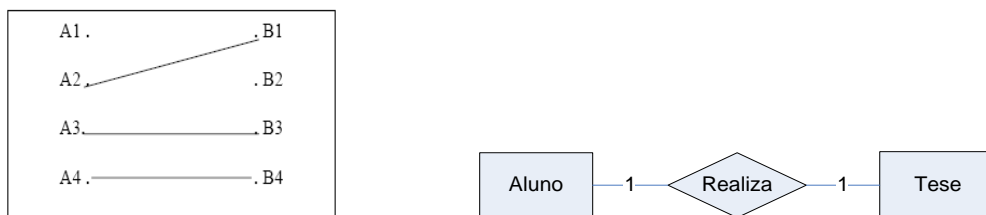
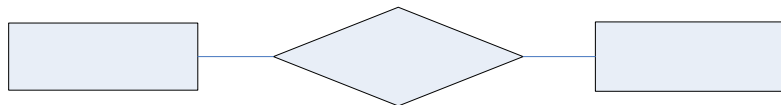


Figura 19 - Associação de um-para-um: realiza

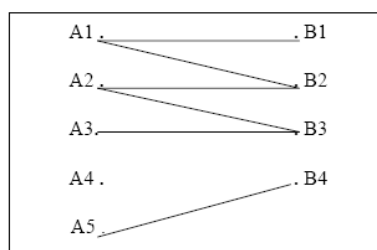
*Complete o seguinte exemplo: Seja um curso em que cada módulo é assegurado por um monitor e cada monitor assegura apenas um módulo.*



Associação M:N - No relacionamento binário de grau M:N

- São sempre necessárias três tabelas, uma para cada entidade e uma terceira para o relacionamento;
- A chave primária de cada entidade serve de chave primária na tabela correspondente
- A tabela relativa ao relacionamento terá de ter entre os seus atributos as chaves primárias de cada uma das entidades

Diagrama de ocorrências



*Exemplo: Um livro pode ser escrito por vários autores, e um autor pode escrever vários livros.*



## Multiplicidade de Associações não binárias

A caracterização da multiplicidade das associações não binárias segue de perto os princípios já estabelecidos para as binárias. Veja-se por exemplo o caso expresso na Figura 21.

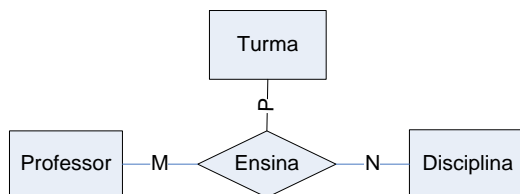


Figura 20 - Associação ternária: ensina

Nela se expressa o seguinte fragmento da descrição textual do UoD acadêmico:

**"Os professores ensinam disciplinas a várias turmas. Cada professor pode ensinar mais do que uma disciplina e cada professor ensina a mais do que uma turma. Como é óbvio, cada turma é ensinada em mais do que uma disciplina."**

Esta relação ternária terá multiplicidade M:N:P, isto é, não existem restrições no relacionamento entre as entidades (qualquer professor pode estar relacionado com qualquer número de turmas e disciplinas, qualquer turma pode estar relacionada com qualquer número de professores e disciplinas e qualquer disciplina pode estar relacionada com qualquer número de professores e turmas).

O artigo original de Chen com a proposta da abordagem ER não incluía referência a casos de associações com aridade superior a dois e multiplicidade diferente de M:N:P (i.e. sem restrições). No entanto é fácil verificar que com a simbologia M:N:P e a possibilidade de cada letra poder ser substituída por um símbolo "1" teremos a possibilidade de exprimir oito hipóteses:

**M:N:P**  
**1:N:P**  
**M:1:P**  
**M:N:1**  
**1:1:P**  
**1:N:1**  
**M:1:1**  
**1:1:1**

Deixando de lado a primeira e a última, cujo significado é óbvio, restam 6 possibilidades. Apresenta-se de seguida uma forma de utilizar estas possibilidades para especificar um conjunto de restrições à multiplicidade das associações que é muito útil ter capacidade de exprimir. Este tipo de restrições está intimamente relacionado com o conceito de dependência funcional que foi definido no contexto da abordagem relacional [CODD70].

Voltemos ao exemplo da Figura 21. As restrições que pretendemos exprimir são do tipo: "Fixado um professor e uma disciplina a turma fica univocamente determinada", i.e. a um dado par de entidades (professor, disciplina) só pode estar associada uma turma. Representa-se esta restrição textualmente na forma:

Professor, disciplina -> turma

No caso da associação ser ternária e uma vez que os tuplos acima não são ordenados, podem definir três restrições daquele tipo:

Professor, disciplina -> turma

Professor, turma -> disciplina

Turma, disciplina -> professor

Um outro tipo de restrição a exprimir é aquele em que apenas uma entidade determina univocamente um par de entidades, a saber:

Professor -> turma, disciplina

Turma -> disciplina, professor

Disciplina -> turma, professor

Associando a este tipo de restrições as convenções de notação adotadas acima para a multiplicidade tem-se (atribuindo o símbolo 1 à/às entidades univocamente determinadas por outras e o símbolo M à multiplicidade do professor, N à da disciplina e P à da turma):

| <b><u>Restrição</u></b>        | <b><u>Notação</u></b> (Prof:Disc:Turma) |
|--------------------------------|---|
| Professor, disciplina -> turma | M:N:1                                   |
| Professor, turma -> disciplina | M:1:P                                   |
| Turma, disciplina -> professor | 1:N:P                                   |
| Professor -> turma, disciplina | M:1:1                                   |
| Turma -> disciplina, professor | 1:1:P                                   |
| Disciplina -> turma, professor | 1:N:1                                   |

Nestas condições, se se pretender exprimir, por exemplo, que para cada par <professor, turma> só existe uma disciplina, tem-se a representação ER gráfica da Figura 22. Identicamente se obtém a representação para os restantes casos.

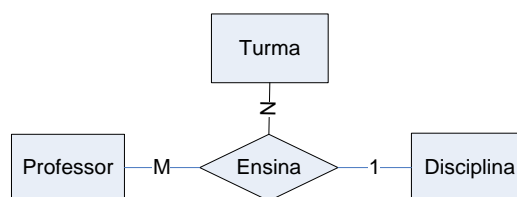


Figura 21 - Associação ternária ensina com restrição professor, turma -> disciplina

### 5.3 Papeis das Entidades nas Associações

As entidades desempenham “papeis” nas associações em que intervêm. Estes indicam qual o significado associado à participação de cada entidade na associação. Em termos da representação gráfica os papéis são indicados junto aos ramos que ligam os conjuntos de entidades ao conjunto de associações. Veja-se o caso da Figura 23 que corresponde à anterior Figura 22 a que se adiciona a especificação dos papéis das várias entidades.

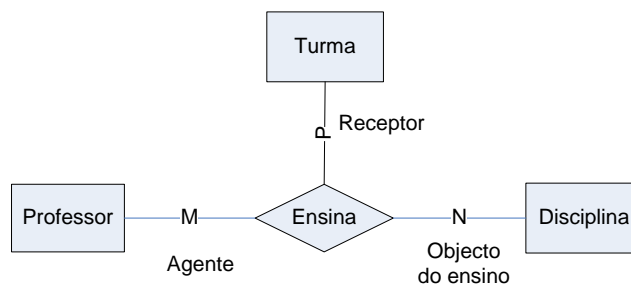


Figura 22 - Associação ternária com indicação do papel das entidades: ensina

Assim a entidade professor desempenha o papel de agente na associação ensina, a turma desempenha o papel de recetor e a disciplina desempenha o papel de objeto-de-ensino. Tem-se assim que o significado (semântica) ligado a cada associação do conjunto ensina passa a ser "um professor ensina (agente) a uma turma (recetor) uma disciplina (objeto-de-ensino)". Note-se que, com a introdução dos papéis, a frase que traduz a semântica da associação (e que se consegue derivar de uma especificação dela de forma automática) constitui praticamente uma descrição em língua natural corrente.

## 5.4 Atributo

Um atributo define-se formalmente como sendo uma função que, a elementos de um conjunto de entidades ou de associações, faz corresponder elementos de um dado conjunto de valores; (Ex: inteiros, strings, ...) ou produto cartesiano de conjuntos de valores.

Atributos, representam os dados da entidade. Por exemplo, para representar a entidade ALUNO, torna-se necessário definir os atributos nome, apelido, morada, telefone, etc..

Cada atributo encontra-se definido num determinado domínio de valores. Assim, o atributo nome é definido por um conjunto de caracteres, telefone, é definido por um conjunto de nove números. Nota: numa Entidade não podem existir dois atributos com a mesma designação.

Graficamente os atributos representam-se como ovals ligadas por segmentos de reta aos respetivos conjuntos de entidades ou associações (Figura 24). Semanticamente, os atributos correspondem à atribuição de valores que caracterizam as entidades e associações. Considere-se o seguinte fragmento da descrição textual do UoD académico.

**" Os alunos são caracterizados por um nome, morada e data de nascimento. Os alunos inscrevem-se em disciplinas numa dada data..."**

A Figura 9 contém a definição dos atributos correspondentes. Note-se que, enquanto nome, morada e data de nascimento caracterizam o aluno, a data caracteriza a inscrição sendo portanto um atributo da associação e não um atributo do aluno ou da disciplina.

Note-se que a definição gráfica da Figura 9 não inclui a especificação dos contradomínios dos atributos os quais deverão ser indicados textualmente, usando por exemplo uma notação funcional:

*nome : aluno -> string (80)*

*data : inscrição -> date*



em que: string (80) é o conjunto de todas as possíveis cadeias de caracteres com dimensão máxima de 80, date é o conjunto de todas as datas.

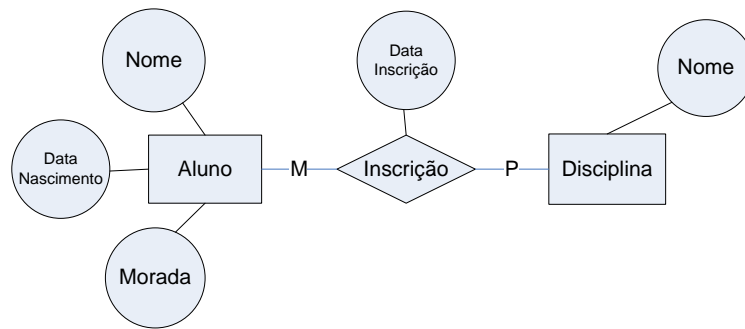


Figura 23 - Conjuntos de entidades e de associações com atributos: inscrição

## 5.5 Chave

As entidades do mundo real presentes em cada UoD são objetos distintos entre si. A sua individualidade tem de ser expressa em qualquer representação desse UoD pelo que deverá existir sempre um mecanismo de identificação unívoca de cada entidade obtida através dos valores que lhe são associados por atributos.

Designa-se por chave de um conjunto de entidades o conjunto (eventualmente singular) de atributos definido de forma que a correspondência entre as entidades e os valores que as identificam seja de um para um. Deverá sempre ser possível encontrar uma chave para cada conjunto de entidades.

Pode também acontecer que exista, para um dado conjunto de entidades, mais do que uma chave possível. Nesse caso escolher-se-á para chave primária a que, do ponto de vista do UoD, exiba uma maior carga semântica que nos leve a considerá-la melhor colocada para o papel de identificador unívoco exteriormente significativo.

Dado que as associações podem ser identificadas pelas entidades nela envolvidas, o conjunto de associações não necessita de ter uma chave própria.

Não foi proposta na referência [CHEN76] uma simbologia gráfica para as chaves. No entanto a existência dessa simbologia é útil. Esta poderá conseguir-se como mostra a Figura 25 sublinhando o nome dos atributos pertencentes à chave.

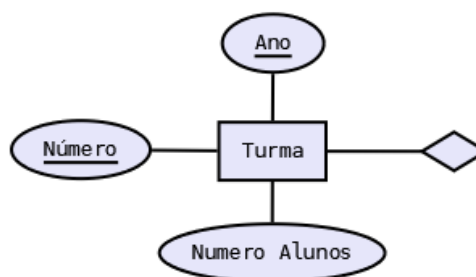


Figura 24 - Chave de um conjunto de entidades: turma

## 5.6 Entidades Fracas

Em certos casos não é possível identificar univocamente as entidades de um dado conjunto usando apenas os seus próprios atributos. Muitas vezes tal sucede porque as entidades dependem para identificação de outras entidades. As entidades nestas circunstâncias designam-se por entidades fracas ("weak entities").

*Entidade Fraca*, é uma entidade dependente de outra entidade. Geralmente tem uma associação de 1:N e a entidade fraca caracteriza-se por não possuir um atributo chave.

Considere-se o seguinte fragmento da descrição textual do UoD académico:

**"... A cada disciplina corresponde um conjunto de peças de avaliação (ex. testes, exames, trabalhos. Estas são definidas no contexto de cada disciplina sendo a sua caracterização válida apenas para a disciplina para que foram definidas..."**

Neste caso não existe uma chave para o conjunto de entidades peça de avaliação. É através da respetiva associação com disciplina que é possível identificar cada uma delas univocamente. Trata-se portanto de uma entidade fraca.

Graficamente uma entidade fraca (ou mais corretamente conjunto de entidades fracas) denota-se por um duplo retângulo em vez de um retângulo simples como é habitual (Figura 26).

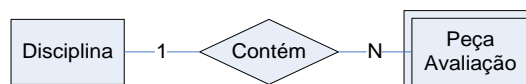


Figura 25 - Conjunto de entidades fracas: peça-de-avaliação

As entidades fracas podem também exibir atributos indicados como pertencentes a uma chave. Esta deve neste caso ser considerada como "chave parcial", i.e. que não identifica completamente as entidades desse conjunto a não ser quando completada pela chave da entidade não fraca com a qual a primeira está relacionada.

Designa-se por associação fraca (ou mais corretamente conjunto de associações fracas) uma associação em que pelo menos um dos argumentos é uma entidade fraca. Não é necessário dispor de um símbolo especial para representar graficamente as associações fracas, dado que, por simples inspeção visual é fácil determinar se pelo menos um dos argumentos é ou não uma entidade fraca.

## 5.7 Particularizações/Generalizações (ISA)

Por vezes é natural classificar uma entidade num conjunto de entidades. Por exemplo para a entidade Empregados podemos criar duas entidades para registo da informação do tipo do seu contrato. São casos particulares das associações do tipo 1:1.

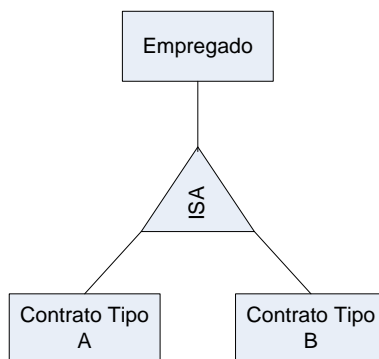


Figura 26 - Entidades Hierárquicas

## 5.8 Associações Recursivas

Uma associação recursiva (unária) ocorre quando uma entidade está relacionada com ela própria.

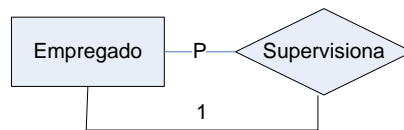


Figura 27 - Entidade Recursiva

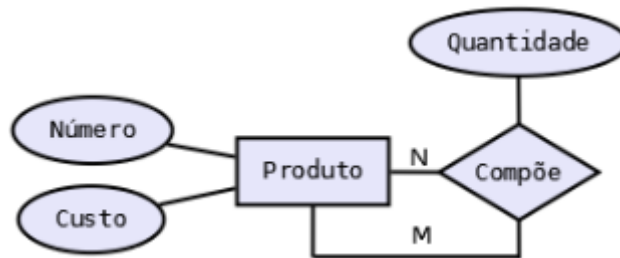


Figura 28 - Entidade Recursiva

## 5.9 Restrições de Existência e de Identidade

O caso da entidade fraca traduz como já foi visto uma restrição de identidade entre entidades de diferentes conjuntos. Poderão no entanto existir, num dado UoD, outro tipo de restrições às quais convém dar um tratamento preferencial dado que ocorrem frequentemente.

Sempre que uma dada entidade pertencente a um dado conjunto de entidades necessita, para ser definida, que outra entidade, pertencente a outro conjunto de entidades com a qual a primeira será associada, tenha sido previamente definida, diz-se que se tem uma restrição de existência da primeira entidade relativamente à segunda. Como se mostra na Figura 30 a restrição de existência é representada graficamente por uma seta que aponta para a entidade dependente.



Figura 29 - Restrição de existência: regente

O exemplo da Figura 30 traduz que não é possível definir uma disciplina sem que lhe seja imediatamente atribuído um Professor.

As entidades fracas pressupõem sempre uma restrição de existência, uma vez que se a entidade fraca depende para identificação de outra entidade, esta terá de existir previamente. Porém, poderão ser definidas restrições de existência entre entidades não fracas (i.e. entidades regulares).

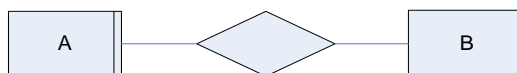
Note-se finalmente que o nível de abstração dos modelos obtidos com a abordagem ER é alto pois as noções captadas com a respetiva estrutura de conceitos são próximas das EMRs e portanto distantes dos conceitos próprios dos eventuais ambientes de realização.

## 5.10 Obrigatoriedade de participação de uma entidade numa associação

Uma entidade pode participar numa associação de duas formas:

- **Obrigatória** - todas as ocorrências dessa entidade, têm que estar associadas a alguma ocorrência da outra entidade que participa na associação.
- **Não obrigatória** - a entidade pode ter ocorrências não associadas a qualquer ocorrência da outra entidade que participa na associação

Representa-se



Entidade A – Obrigatória

Entidade B – Não obrigatória

Exemplos:

|   |  |
|---|--|
| Um departamento tem que ter pelo menos um empregado<br>Um empregado tem de pertencer a um departamento  |  |
| Um departamento pode não ter empregados<br>Um empregado tem de pertencer a um departamento              |  |
| Um departamento tem de ter pelo menos um empregado<br>Um empregado pode não pertencer a um departamento |  |
| Um departamento pode não ter empregados<br>Um empregado pode não pertencer a um departamento            |  |

## 5.11 Transformação para o Modelo Relacional

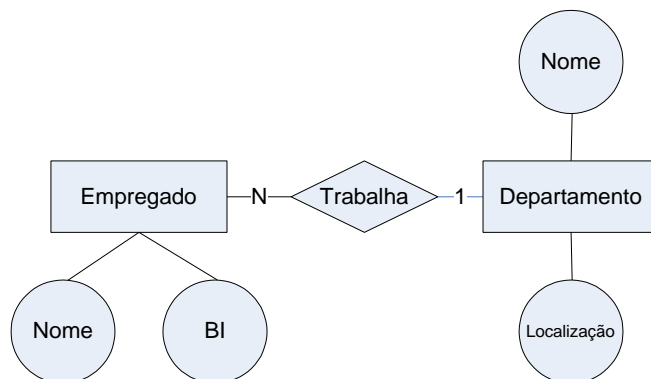
Quando se transforma um diagrama ER para o modelo relacional, é necessário realizar um conjunto de alterações. Este processo de transformação, será feito considerando os vários tipos de associações que envolvem as entidades. Para isto, torna-se ainda necessário considerar alguns conceitos.

**Relação (ou tabela)**, é uma estrutura bidimensional com um conjunto de linhas e colunas. As colunas indicam o tipo de dados a armazenar e as linhas são as instâncias da relação.

**Chave Estrangeira (ou importada ou externa)**, é o atributo que é chave primária numa outra relação.

### 5.11.1 Associações 1:N

Considere o seguinte exemplo:



A transformação deste diagrama passa por colocar o atributo que é a chave primária da relação **Departamentos** (lado **1** da associação) na relação **Empregados** (lado **n** da associação). Assim, de uma forma geral, temos o seguinte modelo relacional:

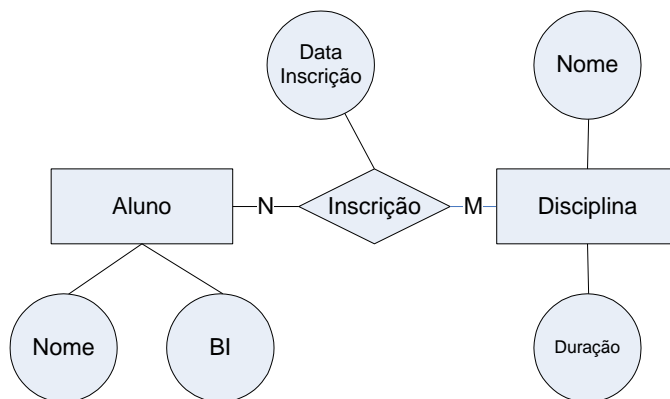
**Empregados**(CodEmpregado, Nome, BI, CodDepartamento, ...)

**Departamentos**(CodDepartamento, Nome, Localizacao, ....)

Onde os atributos sublinhados são as chaves primárias das respectivas relações e o atributo CodDepartamento em **Empregados**, é chave estrangeira, pois referencia a chave primária de **Departamentos**.

### 5.11.2 Associações M:N

Considere o seguinte exemplo:



A transformação deste diagrama passa por adicionar uma relação, além da relação **Alunos** e da relação **Disciplinas**. Assim, de uma forma geral, temos o seguinte modelo relacional:

**Alunos** (CodAluno, Nome, BI, ...)

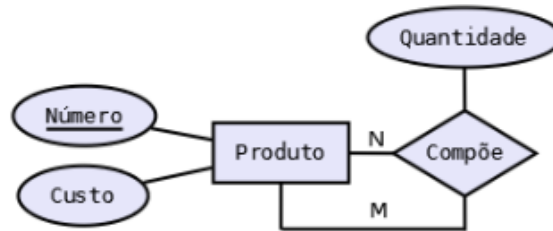
**Disciplinas** (CodDisciplina, Nome, Duracao, ...)

**Inscrições** (CodAluno, CodDisciplina, DataInscricao)

Onde os atributos sublinhados são as chaves primárias das respectivas relações e os atributos CodDisciplina e CodAluno da relação **Inscrições**, são **chaves estrangeiras** pois fazem referência as chaves primárias das relações **Disciplinas** e **Alunos**, respectivamente; além de constituírem a chave primária da referida relação.

### 5.11.3 Casos particulares. Associações recursivas

Considere o exemplo em que um produto é composto por outros produtos numa linha de montagem.



A transformação desse diagrama passa por considerar duas relações **Produtos** e **Componentes**, e colocar a chave primária da relação **Produtos** como atributo da relação **Componentes**. Assim, de uma forma geral, temos o seguinte modelo relacional:

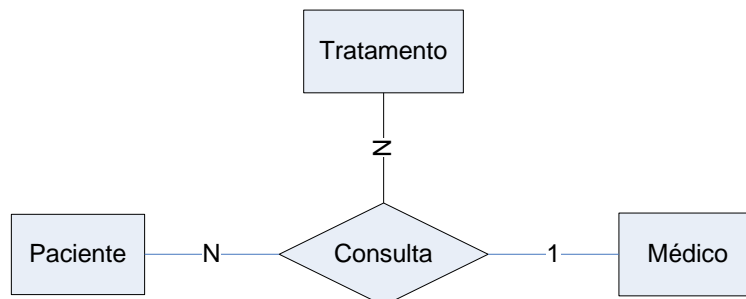
**Produtos** (Numero, Custo, ...)

**Componentes** (Numero, NumeroComponente, Quantidade)

Verifique que os atributos que são chave primária de **Componentes** (Numero e NumeroComponente) também são chave estrangeira. Portanto, estes atributos referenciam o atributo número da relação **Produtos**.

### 5.11.4 Casos particulares. Associações ternárias

Considere o exemplo.



A transformação desse diagrama passa por considerar quatro relações **Paciente**, **Tratamentos**, **Médicos** e **Consultas** (entidade-associação). Esta última irá conter as chaves primárias das três primeiras relações. Assim, de uma forma geral, temos o seguinte modelo relacional:

**Pacientes** (NumeroPaciente, BI, ...)

**Tratamentos** (NumeroTratamento, Descricao, NumeroHoras, ...)

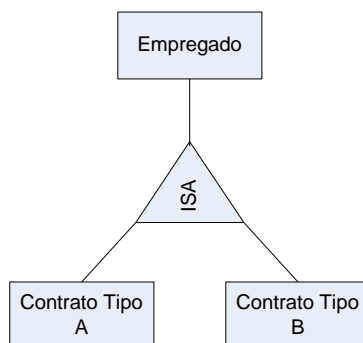
**Médicos** (CodMedico, Nome, ...)

**Consultas** (CodMedico, NumeroTratamento, NumeroPaciente)

Verifique que os atributos que são chave primária de **Consultas** também são chave estrangeira. Portanto, estes atributos referenciam os atributos CodMedico de **Médicos**, NumeroTratamento de **Tratamentos** e NumeroPaciente de **Pacientes**.

### 5.11.5 Casos particulares. Generalização

Considere o exemplo.



A transformação desse diagrama passa por considerar três relações **Empregados**, **Contratos** e **Horas Trabalho**.

Transformando para o modelo relacional:

**Empregados** (NumeroEmpregado, Nome, Morada, ...)

**Contratos Tipo A** (NumeroEmpregado, Categoria, ...)

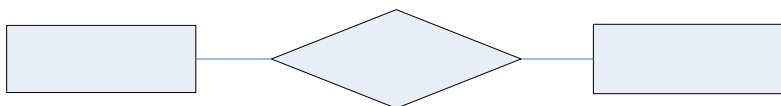
**Contratos Tipo B** (NumeroEmpregado, NumeroHoras, ...)

As chaves primárias das relações **Contratos Tipo A** e **Contratos Tipo B** são também chaves estrangeiras, pois referenciam o atributo NumeroEmpregado, que é chave primária da relação **Empregados**.

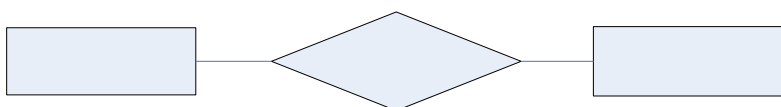
## 5.12 Exercícios de ER

**Ex.ER.01** Para cada par de restrições abaixo, identifique dois tipos de entidades e um tipo de associação. Indique o grau da associação para cada caso. Desenhe um diagrama entidade – associação.

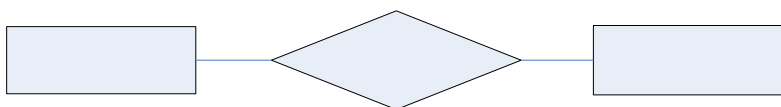
1. Um departamento emprega várias pessoas. Uma pessoa trabalha para quanto muito num departamento.



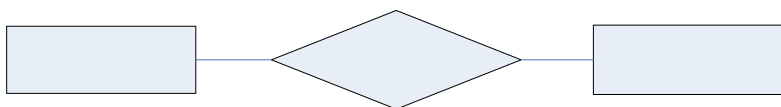
2. Uma equipa consiste em vários jogadores. Um jogador joga para uma só equipa.



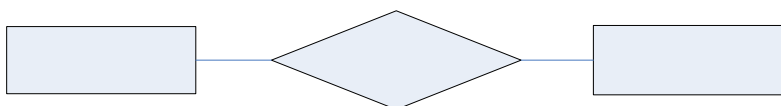
3. Um professor leciona no máximo um curso. Um curso é lecionado por um só professor.



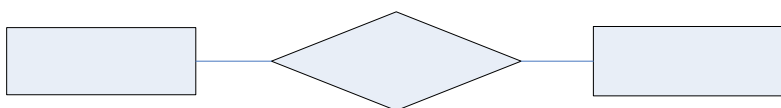
4. Uma nota de encomenda pode ter vários produtos. Um produto pode aparecer em várias notas de encomenda.



5. Um cliente pode receber várias faturas. Uma fatura é de um só cliente.



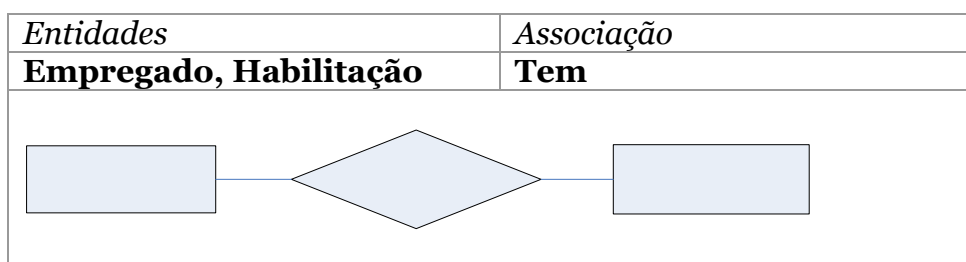
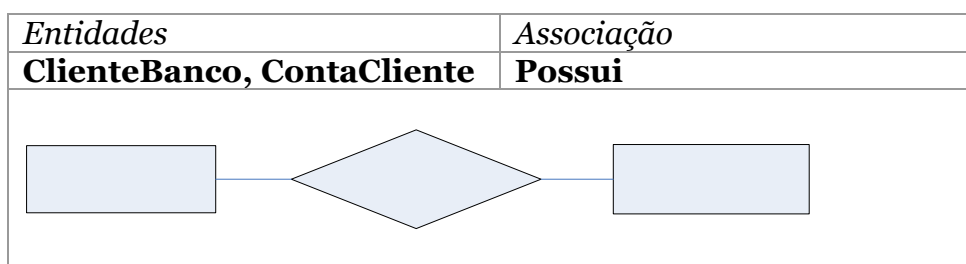
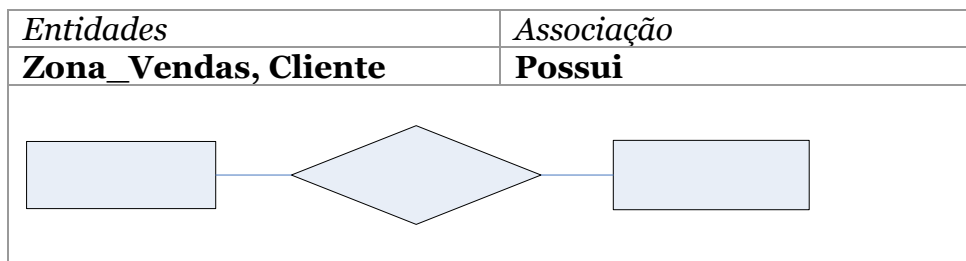
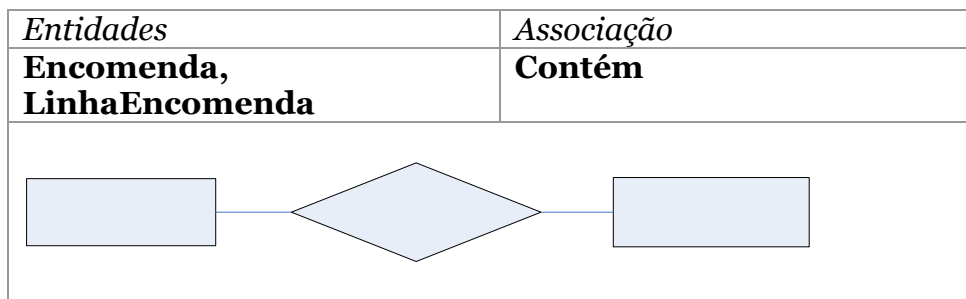
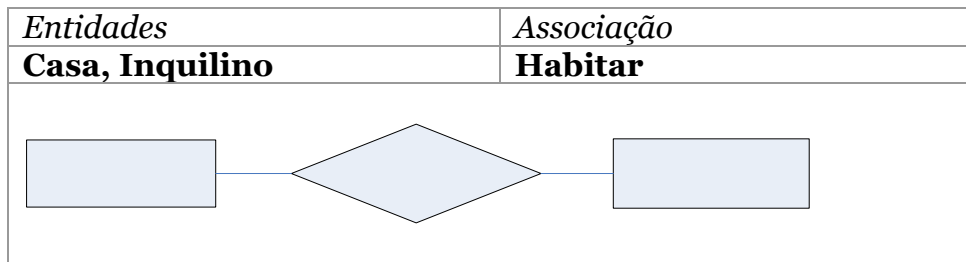
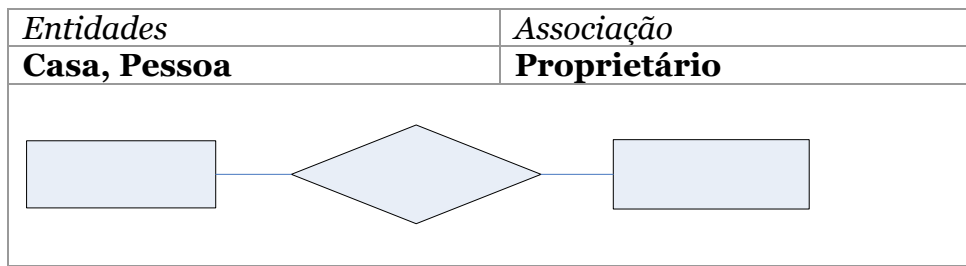
6. Numa clínica médica, cada médico tem vários doentes mas um doente só pode estar registado num médico de cada vez. Supondo que só se incluem os registos de doentes atuais, qual é o grau da associação registado entre as entidade Doente e Médico.



7. Se na questão anterior um paciente pudesse registar-se simultaneamente em vários médicos qual seria o grau da associação?



**Ex.ER.02** Decida plausíveis tipos de participação (obrigatória/não obrigatória) para cada tipo de entidade:



**Ex.ER.03** Considere o seguinte exemplo – Um Docente pretende registar a avaliação dos seus alunos para um determinado momento de avaliação (teste, trabalho individual, trabalho em grupo; exame final, etc.). Por cada momento de avaliação o Docente pretende registar apenas a avaliação obtida pelo Aluno e a data de avaliação. Desenhe o modelo ER respetivo e de seguida faça a derivação para o modelo relacional. Teste a solução encontrada numa nova base de dados a criar no SQL Server.

**Ex.ER.04** Conceba um diagrama ER considerando a informação contida no seguinte texto - Uma empresa apresenta nos seus registos diversa informação sobre cada um dos seus empregados. Cada empregado trabalha incluído num departamento (e apenas num), sendo que num departamento trabalham pelo menos um empregado. Para além do nome, os departamentos têm um orçamento e um código, devendo ficar registado quando é que um empregado iniciou e terminou funções num determinado departamento.

Todos os departamentos possuem obrigatoriamente um e apenas um responsável que é empregado da empresa, devendo haver registo de quando iniciou as suas funções.

A empresa possui vários edifícios espalhados pelo país, sendo que em cada um deles pode haver empregados dos diversos departamentos. Cada localização tem uma morada diferente e capacidade limite no número de trabalhadores que alberga.

Alguns empregados têm um supervisor ao qual devem relatar o andamento das suas tarefas laborais, sendo que os supervisores são também empregados da empresa.

Há ainda a referir que há empregados com contracto e empregados temporários. Para os empregados temporários tem de existir um registo diário das horas de trabalho. Para cada empregado com contracto, é importante saber o respetivo número de contracto, data de início e término do mesmo.

**Ex.ER.05** Um torneio de ping-pong é constituído por diversas partidas. Em cada partida intervêm dois jogadores. Os jogadores são identificados pelo número do bilhete de identidade e são ainda caracterizados pela data de nascimento, enquanto as partidas se identificam apenas à custa dos jogadores que nela participam. As partidas são ainda caracterizadas pela data, hora de início e resultado.

Desenhe um diagrama ER e indique alguma dificuldade de modelação que possa ter encontrado neste universo do discurso e que não tenha conseguido exprimir com a estrutura de conceitos daquela abordagem.

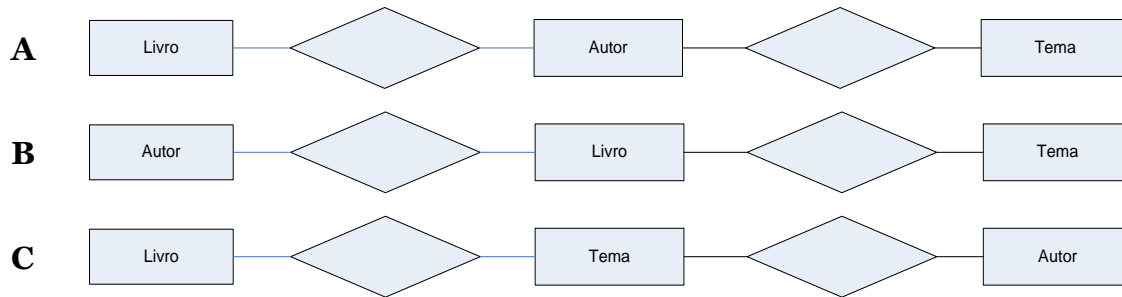
**Ex.ER.06** Para uma base de dados no domínio da gestão de encomendas de produtos é necessário implementar o seguinte requisito técnico:

Um produto pode ser fornecido por mais que um fornecedor. É necessário contudo que seja definido um ranking dos fornecedores para um produto, definindo qual a prioridade de encomendar produtos a um fornecedor. A determinado momento um fornecedor pode alterar a sua posição no ranking se apresentar mais vantagens que os restantes. Os fornecedores podem ser de diferentes nacionalidades.

Desenhe o modelo ER respetivo e de seguida faça a derivação para o modelo relacional.

**Ex.ER.07** Um modelo ER representa autores e a classificação por temas dos livros que escreveu.

Discuta as vantagens das estruturas seguintes. Qual escolheria?



**Ex.ER.08** Uma empresa de importação efetua as suas compras através de contratos.

Cada contrato (identificado por um número) é acordado com um dado fornecedor e diz respeito a várias mercadorias (que são codificadas). Do contrato consta também a data de assinatura, o prazo de validade, a moeda e o valor. No contrato é fixado o preço unitário de compra de cada mercadoria, bem como a quantidade comprada que é especificada numa unidade de medida que é sempre a mesma para cada mercadoria independentemente do contrato.

É necessário manter informação – nome, endereço, telefone e fax – sobre os vários fornecedores que são identificados por um código.

As mercadorias envolvidas num contrato são todas enviadas num único transporte (identificado por um número). Para um transporte é necessário conhecer o tipo de transporte, a data prevista de partida e a data prevista de chegada.

Construa o modelo ER e a respetiva derivação para o modelo relacional, da organização descrita acima.

**Ex.ER.09** Numa companhia cada departamento possui um carro que só pode ser usado por empregados do departamento devidamente autorizados.

Suponha o seguinte modelo ER:



A respetiva derivação para o modelo relacional é a seguinte:

**Carro** (N\_registo, Marca)

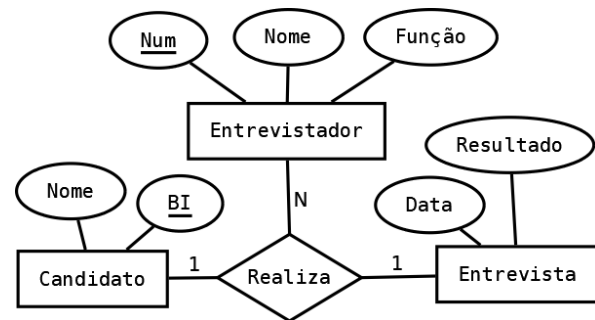
**Departamento** (N\_dep, N\_registo, Nome\_dep, Localização)

**Empregado** (N\_emp, N\_dep, Nome\_emp, Categoria)

- Se souber o número de um empregado autorizado a usar um carro, esta estrutura permitirá determinar qual o carro utilizado?
- Se souber o número de registo de um carro poderá saber-se que empregados estão autorizados a usá-lo?
- Como pode o diagrama ser estendido (por adição de uma associação) para que represente utilizadores autorizados?

- d. Como podem ser representados utilizadores autorizados se for permitido um novo atributo?
- e. Discuta as vantagens e desvantagens de substituir a associação “usado” entre *Carro* e *Departamento* pela associação “usado” entre *Carro* e *Empregado*.
- f. Apresente um modelo ER e a sua derivação no modelo relacional que permitiria resolver os problemas detetados.

**Ex.ER.10** Considere o modelo ER apresentado, que representa o processo de entrevistas em que um candidato apenas pode realizar uma entrevista, mas na qual podem participar vários entrevistadores que em consenso, chegarão a um resultado final. Pretende-se que derive para o modelo relacional o diagrama apresentado.



**Ex.ER.11** Considere a seguinte situação. Durante muito tempo a publicação científica tinha como base de publicação e distribuição as publicações periódicas em suporte de papel, distribuídas essencialmente pelas bibliotecas das instituições de ensino a preços muito altos. A partir do momento em que a produção e publicação científica se tornam num negócio lucrativo, a publicação na Internet desenvolveu-se com grande rapidez e tornou-se praticamente obrigatória.

A instituição SIP (Sistemas de Informação de Portalegre), pretende publicar os artigos recebidos assim como os produzidos internamente na instituição, sendo que todos os artigos necessitam obrigatoriamente de ser aprovados por um júri científico.

O autor<sup>93</sup> ou autores de um artigo científico<sup>94</sup> submetem a sua publicação<sup>95</sup> na revista científica da SIP, sendo que este processo apresenta um custo de 205€. Caso o artigo seja aprovado pelo júri, a sua publicação definitiva<sup>96</sup> apresenta um custo de 380€. Um artigo pode apresentar mais que um autor, sendo que os autores podem pertencer a instituições de ensino, empresariais ou investigação diferentes. Para ambas as tarefas está disponível apenas como forma de pagamento a transferência bancária, sendo necessário registar a data da transação, o montante e o IBAN do cliente.

Aprovação do artigo. O júri científico<sup>97</sup> tem como responsabilidade analisar os artigos submetidos, segundo um conjunto de parâmetros utilizados universalmente. A avaliação dos artigos é realizada através de 5 parâmetros<sup>98</sup> obrigatórios, sendo que cada um pode ter uma

<sup>93</sup> Sobre o autor é necessário registar a seguinte informação: Primeiro Nome, Último Nome, Nacionalidade, Instituição de Origem, Endereço de correio eletrónico. É necessário que os autores sejam identificados por um número incrementado de forma automática a cada novo registo. Um autor pode apresentar vários artigos para publicação.

<sup>94</sup> Sobre o artigo é necessário registar a seguinte informação: IDArtigo (do tipo AAA-123), Título do artigo, Palavras-chave, Resumo/Abstract, Área Científica principal, Área Científica secundária, Autor (es), Data do artigo, Ficheiro PDF com o artigo.

<sup>95</sup> Tarefa definida como “Submeter Artigo”

<sup>96</sup> Tarefa definida como “Publicar Artigo”

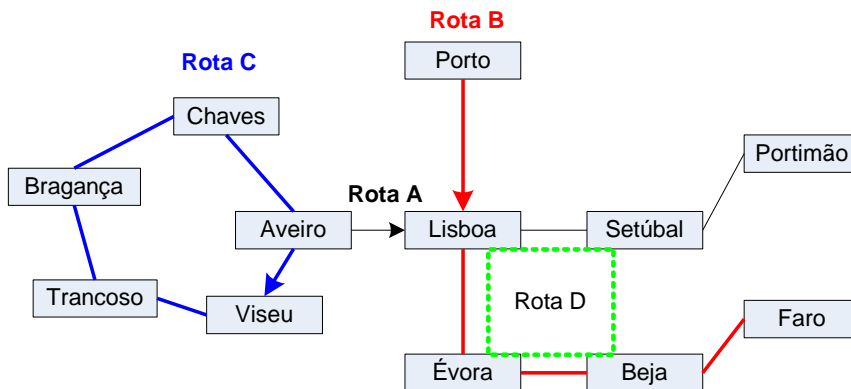
<sup>97</sup> O Júri é constituído por cinco elementos. Um desses elementos é o presidente do Júri. Sobre os elementos do Júri é necessário registar a seguinte informação: Nome Completo, Instituição de origem, email, Grau Académico, Nome de Utilizador, Palavra-chave.

<sup>98</sup> Exemplo de parâmetros: Overall Rating, Technical quality, Organization and presentation, Coverage and depth of the report, Adequate references.

pontuação de 1-5. Cada elemento do júri apresenta a sua avaliação individual e se a avaliação atingir os mínimos estabelecidos o artigo é publicado, caso contrário é retido e informado o ou os autores. O artigo pode ser submetido outra vez para publicação, iniciando-se um novo processo.

Construa o modelo ER e a respetiva derivação para o modelo relacional.

## Ex.ER.12



Uma empresa de transportes rodoviários pretende implementar um sistema de bilheteira automática que será disponibilizada para acesso através da Internet. A cada viagem é atribuída um autocarro, uma rota e um condutor. Sobre o autocarro deve ser registada a sua matrícula, o ano de construção, a data de entrada em funcionamento e a lotação máxima. A rota consiste na sequência ordenada de locais onde o autocarro efetua paragens: a origem da viagem, as várias paragens intermédias e o destino da viagem. Assim, para cada rota, o local de ordem mínima (um) corresponde ao local de partida e o local de ordem máxima dessa rota corresponde ao local onde termina a viagem. A duração de cada local da rota corresponde ao tempo que demora a chegar a esse local desde o local anterior. Para cada local da rota, além da duração também é registado o custo do percurso desde o local anterior. Por isso, o primeiro local de cada rota tem duração e custo iguais a zero. A duração e o custo entre dois locais da rota consistem na soma destes valores desde o local imediatamente seguinte à partida até ao destino. Por exemplo, a duração e o custo entre o local de ordem 3 e o local de ordem 7, corresponde à soma da duração e do custo dos locais de ordem 4, 5, 6 e 7.

Um bilhete corresponde à compra, para uma dada viagem, de um lugar na lotação do autocarro entre dois locais da rota da viagem, onde a origem tem que ter sempre ordem inferior ao destino. Para simplificar o registo dos custos e da duração da viagem, assume-se que todos os autocarros são semelhantes em termos de velocidade e conforto, apenas diferindo na lotação.

Pretende-se:

- Desenhe o modelo ER para o caso apresentado.
- Derive o modelo ER desenhado no modelo relacional.