Bases de Dados

JOSÉ MANUEL FERREIRA MACHADO

Universidade do Minho, 2009

Nota Preliminar

Este texto didáctico é um apoio à unidade curricular de bases de dados, do terceiro ano da licenciatura em Engenharia Informática, e não dispensa à assistência às aulas teóricas e teorico-práticas da disciplina.

Conteúdo

Capítulo 1

Sistemas de Informação

1.1 Definição

Podem ser dadas definições alternativas do conceito de Sistema de Informação (SI). Numa perspectiva estrutural, um SI é um agrupamento de pessoas, processos, dados, modelos, tecnologia e linguagens parcialmente formalizadas, formando uma estrutura coesa, servindo algum propósito ou função organizacional. Numa perspectiva funcional, um SI é um meio implementado tecnologicamente para o registo, armazenamento, e disseminação de expressões linguísticas, assim como para apoio à realização de inferências; assim, o SI facilita a criação e troca de significados que servem propósitos definidos socialmente tais como o controlo, o dar sentido e a argumentação.

Os dados constituem a forma mais primitiva de representar factos ou afirmações verdadeiras ou falsas. É um estabelecimento do nível de cognição muito próximo da realidade, sem a inclusão de qualquer heurística que visa a incorporação de formas de raciocíno. A informação é um conjunto de factos ou afirmações enquadradas num determinado contexto, veiculando intenções e dando algum significado aos factos. O conhecimento enquadra-se num contexto mais alargado, em comunidades e paradigma com história, cultura ou tradição, e é uma estrutura duradoura de factos com significado. Traduz-se num conjunto de crenças declaradas sobre um assunto com reinvindicações legítimas de verdade ou de correcção (Figura 1.1).

Um SI é um sistema onde a informação deve ser capturada, representada, armazenada e processada no contexto da organização (estruturadas em secções, com ou sem níveis hierárquicos, com uma cultura própria e procedimentos comuns), das pessoas (com uma determinada formação, motivação e ergonomia) e da tecnologia, resolvendo os problemas criados por factores internos e externos. Deverá também interagir com o Ambiente Externo, com características políticas, demográficas, económicas e sociais.

O SI pode ser dividido em diferentes tipos, de acordo com a hierarquia da organização (Figura 1.2). Os utilizadores envolvidos estão ligados a cada um

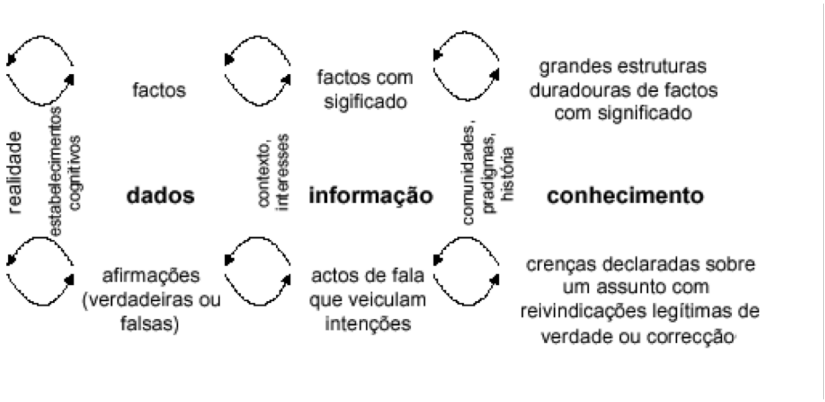

Figura 1.1: Dados, informação e conhecimento

desses níveis de acordo com as funções exercidas. As áreas aplicacionais divergem na base da pirâmide mas convergem a medida que envolvem sistemas e pessoas de nível superior, integrando e partilhando dados, informação e procedimentos (Enterprise Resource Planning – ERP).

A base da pirâmide anterior envolve sistemas de processos transacionais (Transaction Processing System), enquanto os níveis superiores envolvem actividades ditas de gestão (Management Information System) (Figura 1.3)

1.2 Enterprise Resource Planning (ERP)

Designa-se por ERP um conjunto de aplicações informáticas para o apoio de uma forma integrada às funções financeira, logística, produção ou de recursos humanos numa organização. Este assume uma perspectiva dos processos da organização e as várias funções ou actividades são integradas por intermédio de uma base de dados comum. O ERP beneficia:

- da integração de dados: a introdução/alteração de dados num módulo (função) é automaticamente reflectida nos outros módulos;
- da redução de dados redundantes, o que evita a reintrodução de dados;
- da melhoria da comunicação entre funções pela automação da transferência de dados;
- do acesso em tempo real a dados operacionais;
- do reforço do funcionamento fluído dos processos de negócio.

O ERP é todavia um modelo que tem custos associados elevados, apesenta um processo de implementação crítico que exige muitos recursos em tempo e pessoas, além de novas metodologia de abordagem e resolução de problemas. A configurabilidade é limitada.

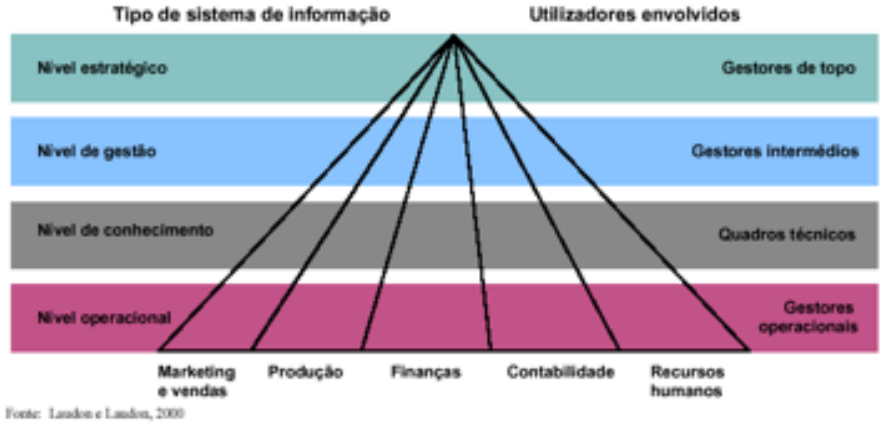

Figura 1.2: Tipos de sistemas de informação

Um dos aspectos mais críticos nos relacionamentos inter-organizacionais tem a ver com o envio / recepção de informação entre sistemas de informação de empresas ou organizações, possivelmente heterógeneos e com objectivos distintos. Existem diferentes técnicas para facilitar essas operações, tais como o email, o voice mail, a partilha de ficheiros, a videoconferência, a webconferência, a Internet, as comunicações móveis e o Electronic Data Interchange (EDI[1]) . A diferença fundamental entre estes tipos de comunicação reside nas diferenças funcionais ao nível destas quatro características:

- Transporte: enviar e receber mensagens;
- Linguagem: significado da mensagem;
- Ontologia: estruturação das conversações;
- Arquitectura: interligação entre os sistemas de acordo com os protocolos

1.3 Resolução de Problemas

A resolução de problemas faz-se dividindo o problema em três sub-problemas:

1. a identificação do problema e os seus dados de entrada;

[1]O EDI tem como objectivo a comunicação electrónica de dados entre empresas ou organizações. A troca de dados é realizada entre aplicações informáticas heterogéneas, baseadas em ferramentas de controlo, gestão e bases de dados diferentes e em sistemas operativos possivelmente diferentes, com pouca intermediação humana. O EDI facilita a troca de dados e as transacções, usando protocolos (e.g., XML) que possibilitam as comunicações; i.e., envio e recepção de informação. O único requisito em termos de equipamento é a garantia de infra-estruturas de comunicação que permitam o envio de ficheiros com informação e transacções, através de redes de valor acrescentado, da internet ou ligações ponto a ponto.

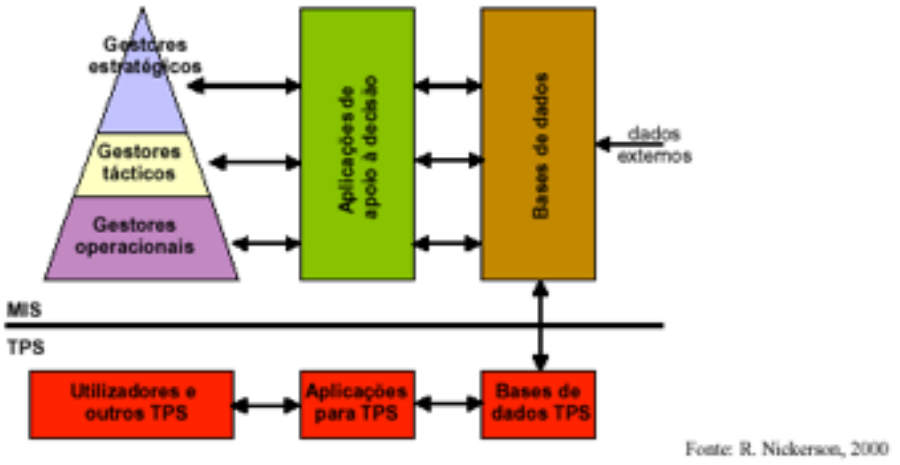

Figura 1.3: TPS vs MIS

   (a) capturar dados;
   (b) introduzir dados;
   (c) validar dados;

2. a transformação dos dados de entrada em resultados, com recurso à informação armazenada em bases de dados;
   (a) processamento;
      i. processar;
      ii. tomar decisões;
   (b) armazenamento;
      i. guardar;
      ii. aceder;
      iii. ordenar;
      iv. actualizar;

3. a apresentação dos resultados;
   (a) produzir saída para o monitor;
   (b) produzir saída para a impressora.

A Figura 1.4 ilustra esta metodologia numa perspectiva reactiva, em que o processador apenas reage a pedidos de resolução do problema.

Figura 1.4: Resolução de Problemas (Método Tradicional)

Figura 1.5: Resolução de Problemas (Abordagem baseada na Inteligência Artificial)

Na figura 1.5 o processador tem um papel pró-activo e é capaz de desenvolver acções sem receber qualquer pedido do exterior.

O desenvolvimento de SI pode ser feito de duas formas diferentes, dependendo da existência ao não de soluções de mercado aceitáveis para o problema.

1.3.1 SI à medida

Partindo duma ideia concreta, avaliando as necessidades e aproveitando a oportunidade são realizadas quatro fases:

1. análise:
   (a) análise de processos
   (b) análise organizacional

(c) identificação de requisitos.

2. Especificação:
   (a) especificação de requisitos funcionais;
   (b) especificação de requisitos não funcionais;
3. desenho:
   (a) definição de tecnologias;
   (b) definição de arquitecturas;
4. implementaçao:
   (a) programação;
   (b) integração.

No final do processo o sistema poderá em fase de funcionamento.

1.3.2 SI adaptado

A implementação de um SI adaptado sofre apenas alterações nas fases:

- desenho:
  - definição de infra-estruturas;
  - configuração;
- implementação:
  - teste;
  - integração.

O ciclo de vida de um SI é apresentado na Figura 1.6.

Qualquer erro que possa surgir nas fases e nas operações descritas poderá levar a problemas de difícil resolução, que atrasarão o processo e poderão levar a prejuízos financeiros e logísticos graves (Figura 1.7).

1.4 Modelos básicos de comunicação

Os modelos básicos de comunicação são os seguintes:

- Troca de mensagens;
- Invocação de Procedimentos remotos;
- Avaliação remota; e
- Objectos móveis.

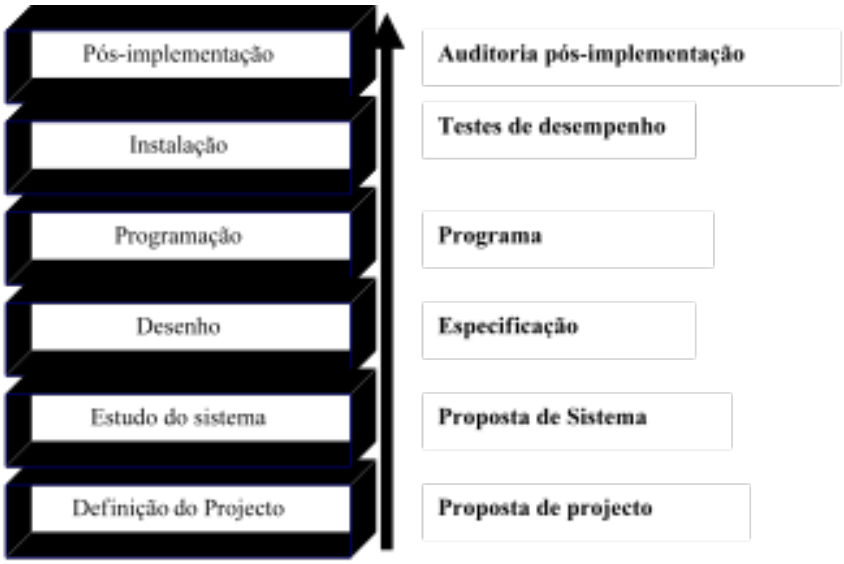

Figura 1.6: Ciclo de vida de um SI

1.5 Governo electrónico

O governo electrónico apresenta-se como uma rara oportunidade que se nos oferece para se tirar vantagem do aumento de produtividade e redução de custos que podem ser obtidos pela utilização de tecnologias Internet. Por outro lado, pode-se afirmar que a Administração Pública tem como principal objectivo servir o cidadão, bem, rapidamente e ao menor custo possível. Sem a preciosa ajuda dos sistemas de informação, não será possível, porém, não só garantir esse objectivo, como também modernizar a própria estrutura em que esta assenta.

A sociedade da informação foi considerada uma prioridade para a Comissão Europeia durante a presidência portuguesa, em 2000. Em 2002, foi criada a Unidade de Missão Inovação e Conhecimento (UMIC), estrutura de apoio ao desenvolvimento da política governamental em matéria de inovação, sociedade da informação e governo electrónico, à qual compete actuar no âmbito das áreas da Inovação, do Governo electrónico, da Economia digital, do Apoio aos cidadãos com necessidades especiais na sociedade da informação, e no Acesso generalizado à Internet, de preferência através de banda larga. No domínio da sociedade da informação, da inovação tecnológica e do governo electrónico, foi também criada a Comissão Interministerial para a Inovação e Conhecimento, à qual compete propor estratégias, promover a articulação dos diversos programas e iniciativas, debater, aprovar e actualizar o elenco das responsabilidades dos diferentes ministérios e organismos públicos, e acompanhar a execução do "Plano de acção e-Europe 2005: Uma sociedade do conhecimento para todos, e

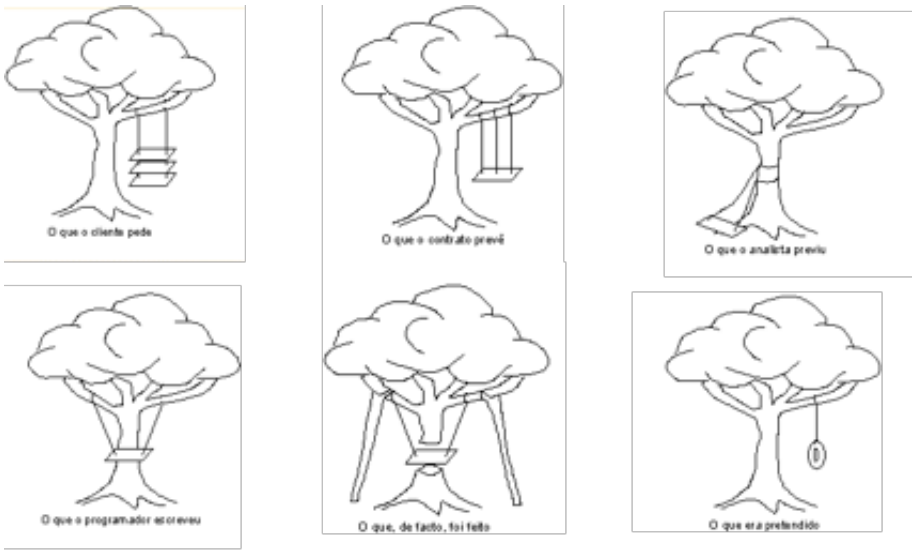

Figura 1.7: Erros nas fases de desenvolvimento de SI

de outros programas da União Europeia no âmbito da inovação, da sociedade da informação e do governo electrónico".

A Sociedade da Informação é aí definida como uma "alavanca para a mudança, capaz de reforçar a competitividade da economia portuguesa, modernizar a Administração Pública, qualificar os portugueses e aumentar a qualidade de vida". A implementação do governo electrónico não pode seguir uma estratégia de rompimento com o passado, com as pessoas e com o funcionamento dos serviços. As soluções financeiras têm de aparecer, os sistemas devem ser integrados e integradores, as implementações devem ser suficientemente rápidas, adaptáveis e escaláveis, e os portais temáticos que são a fachada das instituições, organizações e repartições virtuais, são apenas a interface do sistema. Estes parâmetros estão longe de serem suficientes. Há que apostar nas comunicações, na gestão dos conteúdos e na integração dos meios físicos de interacção entre os humanos e os computadores, através de técnicas convencionais ou inovadoras, tais como o email, o voice mail, a partilha de ficheiros, a videoconferência, a webconferência, a Internet e as comunicações móveis, assim como a difusão e a integração da informação, do conhecimento e dos dados entre sistemas computacionais heterogéneos. É fundamental introduzir os mecanismos para garantir a confiança, a segurança, a fiabilidade no armazenamento e a integração dos dados, da imagem e da voz. É indispensável preparar os sistemas internos de gestão e administração, para receber e enviar a informação aos gestores e aos cidadãos, assim como criar os procedimentos necessários à especificação e desenvolvimento de sistemas de apoio à decisão, capazes de proporcionar as respostas certas e atempadas (e.g., não se deve obrigar os homens do Direito a enviar documentos para os tribunais em email seguro e autenticado, para depois imprimirem-se esses

documentos, sem que estes sejam processados e armazenados informaticamente. Não se deve construir um portal centralizado que permita ao cidadão pedir uma certidão para que depois o pedido chegue à conservatório via fax, a certidão sejam fotocopiada e entregue ao cidadão por correio postal muitos dias depois).

O governo electrónico é um multi-sistema de informação, onde a informação é difundido, armazenada e processada entre sistemas possivelmente heterogéneos que usam protocolos de comunicação compatíveis e sistemas de bases de dados distribuídos e heterogéneos. Os sistemas serão divididos em duas categorias:

- Sistemas internos (sistemas de informação complexos com capacidades em termos operacionais, planeamento e gestão de recursos, documentação e "back-office"), sub-divididos em duas sub-categorias:
- Sistemas de apoio à decisão, tácticos e estratégicos;
- Sistemas operacionais, em particular nas áreas administrativa e financeira;
- Sistemas de relação com o exterior;
- Portais baseados em tecnologia Internet;
- Centro de atendimento aos cidadãos;
- Centrais de compras (e.g., através de sistemas de comércio electrónico usando mecanismos de negociação e argumentação);
- Canais de comunicação entre governo, cidadãos e funcionários.

O ambiente a ser desenvolvido no âmbito do governo electrónico deve, assim, além da concepção e desenvolvimento dos sistemas de informação e das bases de dados, levar em linha de conta que todos os organismos públicos nele têm de ser integrados, e ser centrado em três componentes âncora: Governo e Cidadãos (G-e-C), Governo e Negócios (G-e-N), Governo e Funcionários (G-e-F).

As componentes G-e-C e G-e-N estão intimamente relacionadas com o público em geral e o mundo empresarial e de serviços. A componente G-e-F dirige-se, no essencial, aos funcionários públicos, por forma a se conseguir o seu melhor desempenho e se satisfaçam, assim, os desafios colocados pela Nova Economia.

Resumindo, poder-se-á afirmar que o governo electrónico faz-se aproveitando a dispersão geográfica, a independência das plataformas e a riqueza das soluções já implementadas, ligando-as e integrando-as por meio de comunicações rápidas e usando protocolos e normas de segurança, de difusão e de integração da informação, sem nunca esquecer que não são nem os donos das equipas de fórmula um, nem os mecânicos, nem os patrocinadores, que pilotam as máquinas, mas sim os próprios pilotos.

Capítulo 2

Sistemas gestores de bases de dados

2.1 Definição

Definição 1: Uma base de dados é uma colecção de grande quantidade de dados.

Definição 2: Um Sistema Gestor de Bases de Dados (SGBD) é uma aplicação informática desenvolvida para armazenar e gerir bases de dados.

Os Sistemas Gestores de Bases de Dados organizam grandes quantidades de informação. Uma base de dados é uma colecção de relações, que se encontram relacionadas através de atributos comuns. A informação está contida em dispositivos chamados ficheiros. Os utilitários para gerir bases de dados disponibilizam ao utilizador uma linguagem simples, para fazer a manutenção da base de dados (através de inserções, modificações ou remoções de informação) e para responder a questões que podem ser colocadas pelo utilizador. Essa linguagem (normalmente de comandos) é uma linguagem estruturada que facilita a consulta aos dados organizados na base de dados (e.g., SQL).

O modelo relacional de bases de dados é o resultado do trabalho de E.F.Codd na IBM durante a década de 70. Entre a publicação de um artigo com os fundamentos teóricos do modelo relacional e o aparecimento de um primeiro sistema baseado no modelo relacional passaram vários anos. Só em 1979/80 surgiu uma primeira implementação, através da Relational Software Inc. (actualmente Oracle Corp.). Apesar de o modelo relacional ser muito mais simples e flexível, muitos contrapunham que o seu desempenho nunca iria permitir a sua utilização comercial.

O tempo encarregou-se de provar o contrário, resolvidos os problemas iniciais. A sua implementação foi rápida, contribuindo decisivamente para a massificação da bases de dados nas organizações. O Oracle Server da Oracle Corp., Informix SE e RDS, DB2 da IBM, SQL Server da Microsoft, Sybase SQL da

Código	Designação	Imagem
01	15" PHILIPS 150MT20P TFT	
02	15" PHILIPS 150P3C BLACK	
03	17" PHILIPS 107B30	
04	18" PHILIPS 180B2S White	
05	18.1" PHILIPS TFT 180P2G BLACK	
06	19" PHILIPS 109S	
07	21" PHILIPS 201B 1100 XSD	
08	22" PHILIPS 202P REAL FLAT PRO	

Figura 2.1: Uma tabela

Sybase Inc. entre outros, são os mais vendidos dos grandes fabricantes que aderiram aos SGBDs seguindo o modelo relacional.

A estrutura relação (tabela) (Figura 2.1) constitui uma estrutura bidimensional, com os atributos da relação (uma ou mais colunas) e tuplos da relação (0 ou mais linhas). O esquema conceptual de uma relação é o conjunto dos seus atributos, que traduzem o tipo de dados a armazenar.

Uma base de dados é relacional se verificar as seguintes 12 regras denominadas regras de Codd:

1. Numa base de dados relacional, todos os dados, incluindo o próprio dicionário de dados, são representados de uma só forma, em tabelas bidimensionais.

2. Cada elemento de dados fica bem determinado pela combinação do nome da tabela onde está armazenado, valor da chave primária e respectiva coluna (atributo).

3. Os valores nulos são suportados para representar informação não disponível ou não aplicável, independentemente do domínio dos respectivos atributos.

4. Os metadados são representados e acedidos da mesma forma que os próprios dados.

5. 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 interactiva ou em programas de aplicação.
   - Definição de dados.
   - Definição de views.
   - Definição de restrições de integridade.
   - Definição de acessos (autorizações).
   - Manipulação de transacções (commit, rollback, etc.).

6. Numa view, todos os dados actualizáveis que forem modificados, devem ver essas modificações traduzidas nas tabelas base.

7. Há a 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 actualização.

8. Alterações na organização física dos ficheiros da base de dados ou nos métodos de acesso a esses ficheiros (nível interno) não devem afectar o nível conceptual – independência física.

9. Alterações no esquema da base de dados (nível conceptual), que não envolvam remoções de elementos, não devem afectar o nível externo – independência lógica.

10. As restrições de integridade devem poder ser especificadas numa linguagem relacional, independentemente dos programas de aplicação, e armazenadas no dicionário de dados.

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 de dados.

12. Se existir no sistema uma linguagem de mais baixo nível (tipo record-oriented), ela não deverá permitir ultrapassar as restrições de integridade e segurança.

Não existe nenhuma implementação do modelo relacional que, à luz das doze regras de Codd, possa ser considerada completamente relacional.

2.2 Objectos de uma base de dados relacional

Um sistema de bases de dados relacionais contêm um ou mais objectos chamados tabelas. Os dados ou informação são armazenados nessas tabelas. As tabelas são apenas identificadas pelo nome e contêm colunas e linhas. As colunas contêm

o nome da coluna, o tipo de dado, e qualquer outro atributo para a coluna. As linhas contêm os registos ou dados para as colunas. O grau é o número de colunas de uma tabela. A cardinalidade é o número de tuplos da mesma.

As aplicações devem transferir conjuntos de dados de grande dimensão entre a memória principal dos computadores / servidores e as unidades externas de armazenamento (e..g, discos) através de técnicas sofisticadas de "buffering", paginação e endereçamento. As diferentes interrogações obedecem a codificações especiais e os dados devem estar protegidos da inconsistência às vezes resultante da intervenção de múltiplos utilizadores. Devem ser considerados routinas de recuperação em caso de falha e implementados sistemas de segurança e de controlo de acesso para os utilizadores.

A chave de uma tabela define uma forma mais fácil e mais rápida de aceder à informação. Normalmente, está implicito um sistema de codificação na definição de uma chave, podendo ser uma codificação sequencial, numérica, alfanumérica (e.g. siglas). Podem ser definidos vários tipos de chaves:

Chave candidata – subconjunto de atributos de uma relação que , em conjunto, identificam univocamente qualquer tuplo, e que não pode ser reduzido sem perder essa qualidade.

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

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

Uma entidade será representativa de uma classe de objectos sobre os quais se quer guardar informação (e.g., numa clínica as entidades poderão ser: médicos, pacientes, especialidades, etc). As entidades correspondem, ao nível do modelo relacional, a relações. Cada instância de uma entidade será caracterizada pelos valores de um conjunto de atributos (e.g., um médico tem o seu nome, morada, especialidade etc.). Entre entidades estabelecem-se um conjunto de relações:

- Relações de 1 para 1.
- Relações de 1 para muitos (1 para ∞).
- Relações de muitos para muitos (∞ para ∞).

Estas últimas levam à criação de novas entidades no processo de normalização.

2.3 Normalização da informação

A normalização é um processo sistemático, definido por um conjunto de regras bem definidas, que visa eliminar fontes de redundância nos dados:

- Problemas de manutenção;

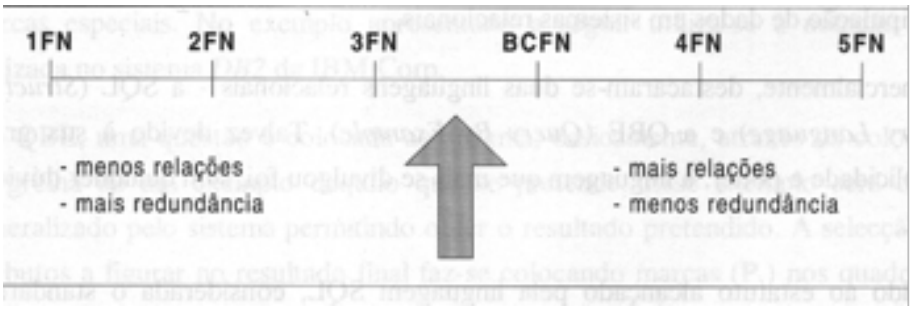

Figura 2.2: Normalização

Cod.Medico:_____ Nome: _____ Especialidade : ___ --- _____

Morada Med. : _____ Contacto: _____ Idade : _____

CONSULTAS:

Data	Hora	N.Benif	Paciente	Idade	Morada	Cod.Postal	Preço

Figura 2.3: Ficha de consulta médica

- Custos de espaço de armazenamento;
- Problemas de desempenho.

O processo de normalização (Figura 2.2) ocorre através de um conjunto de fases que conduzem a base de dados a estados onde a redundância se torna cada vez menor. A cada um destes estados dá-se o nome de forma normal (FN).

Consideremos a ficha de consulta de uma clínica médica (Figura 2.3):
Considere as seguintes restrições:

- Um médico só consulta uma especialidade.
- O preço da consulta depende da especialidade.
- Durante a consulta podem ser feitos tratamentos ao paciente que serão pagos em conjunto com a consulta.
- A clínica tem 30 médicos a consultar 10 especialidades, para um universo de 10000 pacientes, tendo cerca de 50 consultas por dia.

O primeiro passo do processo de normalização (1^aFN), visa eliminar grupos de valores repetidos para um dado atributo, ou conjunto de atributos num dado tuplo. Analisando o exemplo, verifica-se que para a entidade médico podem existir várias consultas. A solução passa por decompor a relação inicial criando novas relações, tantas quantas o número de grupos que se repetem.

A nova relação teria como atributos a chave primária da relação de onde é originária, bem como os atributos que fazem parte do grupo que se repete. O esquema conceptual da base de dados na 1^aFN é o seguinte:

Médico:(*cod_med*, nome,morada,contacto,idade,*cod_esp*,especialidade,preço)

Consulta(*cod_med*,data,hora,*n_ben*,paciente,idade,morada,*cod_post*,localidade, preço)

Note-se que a chave da tabela criada é quadrupla, incluindo a chave da tabela inicial e os atributos data, hora e *n_ben*.

Todo o processo necessário à normalização está traduzido nas dependências existentes entre os dados, existindo três tipos de dependências:

Funcionais – existe uma dependência funcional $X \rightarrow Y$ entre dois conjuntos de atributos X e Y, se uma instância de valores de X determina ou identifica univocamente uma instância de valores dos atributos Y.(este conceito vai ser utilizado para estabelecer a 2^a FN, a 3^a FN e a FB de Boyce-Codd).

Multivalor – numa relação R(X,Y,Z) diz-se que existe uma dependência multivalor $X --\!>\!> Y$ (X multidetermina Y) se, para cada par de tuplos de R contendo os mesmos valores de X também existe em R um par de tuplos correspondentes à troca de Y no par original (é com base nesta dependência que se desenvolve a 4^a FN).

Junção - existe uma dependência de junção se, dadas algumas projecções sobre essa relação, apenas se reconstroí a relação inicial através de algumas junções bem específicas mas não de todas (é com base nesta dependência que se desenvolve a 5^a FN).

Existe uma dependência funcional $X \rightarrow Y$ entre 2 conjuntos de atributos X e Y, se uma instância de valores dos atributos de X determina ou identifica univocamente uma instância de valores dos atributos de Y. Não existem 2 instâncias distintas de Y para uma mesma instância de X.

Se $X \rightarrow Y$ então existe uma dependência funcional entre X e Y em que X determina Y ou Y depende de X. Por exemplo:

N^{o}Funcionário $\rightarrow$ $Nome\_funcionário$, Departamento

(N^{o}Factura, *Cod_produto*) $\rightarrow$ *Qtd_vendida*, *Preço_venda*

Por definição se $X \rightarrow Y$ então a correspondência entre X e Y é do tipo 1:1 ou ∞:1

Axiomas de Armstrong :

Reflexividade (se $X \subseteq Y$ então $X \rightarrow Y$)

Aumentatividade (se $X \rightarrow Y$ então $XZ \rightarrow YZ$)

Transitividade (se $X \rightarrow Y$ e $Y \rightarrow Z$ então $X \rightarrow Z$)

Regras de Inferência :

Decomposição (se $X \rightarrow YZ$ então $X \rightarrow Y$ e $X \rightarrow Z$)

União (se $X \rightarrow Y$ e $X \rightarrow Z$ então $X \rightarrow YZ$)

Pseudotransitividade (se $X \rightarrow Y$ e $YW \rightarrow Z$ então $XW \rightarrow Z$)

O conceito de dependência funcional é um caso especial de uma outra, a dependência multi-valor, que apenas se prova se a relação tiver pelo menos 3 atributos. A 4^a FN usa este tipo de dependência.

Uma relação na 2^aFN é uma relação em que todos os atributos não pertencentes a qualquer chave candidata, devem depender da totalidade da chave. Retomando o exemplo verificamos que na relação Consulta existem duas dependências funcionais:

Consulta(*cod_med*,data,hora,*n_ben*,paciente,idade,morada,*cod_post*,localidade, preço)

cod_med,data,hora,*n_ben* → preço

n_ben → paciente,idade,morada,*cod_post*,localidade

Esta ultima dependência está em desacordo com definição da 2^aFN.

A solução passa por decompor a relação de acordo com as dependências funcionais anteriores:

Consulta(*cod_med*,data,hora,*n_ben*,preço)

Paciente(*n_ben* ,paciente,idade,morada,*cod_post*,localidade)

O problema detectado está agora resolvido, a base de dados está na 2^a FN.

Uma relação na 3^aFN é uma relação em que não existem dependências funcionais entre atributos não chave (dependências transitivas). Retomando o exemplo verificamos que na relação Médicos existem duas dependências funcionais:

Médico:(*cod_med*,nome,morada,contacto,idade,*cod_esp*,especialidade,preço)

cod_med → nome,morada,contacto,idade,*cod_esp*

cod_esp → especialidade,preço

Esta ultima dependência está em desacordo com definição da 3^a FN.

A solução passa por decompor a relação de acordo com as dependências funcionais anteriores:

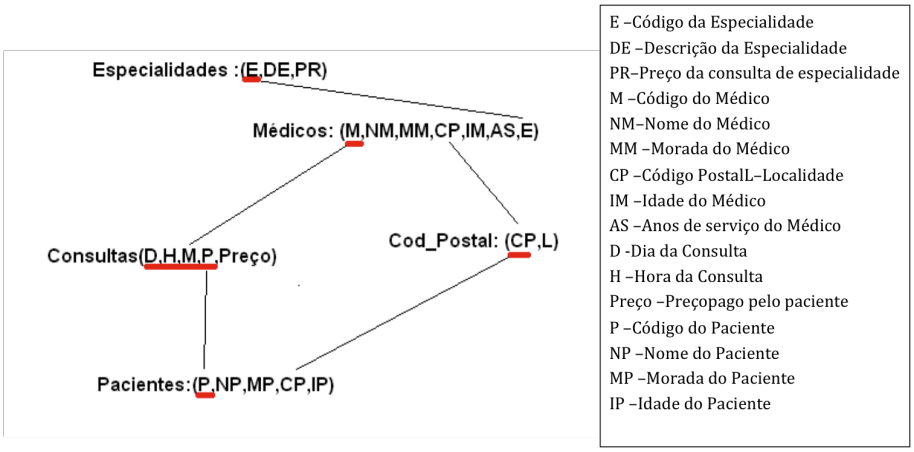

Figura 2.4: Informação na 3^a forma normal

Médicos(*cod_med*,nome,morada,contacto,idade,*cod_esp*)

Especialidades(*cod_esp*, especialidade,preço)

Na relação Pacientes existem duas dependências funcionais:

Paciente(*n_ben*,paciente,idade,morada,*cod_post*,localidade)

n_ben → paciente,idade,morada,*cod_post*

cod_post → localidade

A solução passa novamente por decompor a relação de acordo com as dependências funcionais anteriores:

Pacientes(*n_ben* ,paciente,idade,morada,*cod_post*)

Cod_Postal(*cod_post*, localidade)

Os problemas detectados estão agora resolvidos, a base de dados está na 3^a FN.

O exemplo está ilustrado na Figura 2.4.

Definição: Uma relação está na BCNF se todos os atributos são funcionalmente dependentes da chave, de toda a chave e nada mais de que a chave.

A 3^a FN é aquela em que, na maioria dos casos, termina o processo de normalização, em alguns casos a 3^aFN ainda transporta algumas anomalias que são resolvidas pela FN de BC.

Conderemos que no sentido de prestar um melhor serviço aos pacientes, para cada serviço o paciente é sempre atendido pelo mesmo médico.

Daqui temos a seguinte dependência funcional:

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_1	y_2	z_1
x_1	y_2	z_2
x_3	y_1	z_1
x_4	y_3	z_2

Tabela 2.1: Ilustração de dependências multi-valor

(paciente,*cod_serv*) → medico

representada na relação: R(paciente,*cod_serv*,medico)

No entanto um médico pertence a um só serviço, dependência que não está a ser atendida. Uma solução poderia passar por decompor R em duas relações:

R_1(paciente,médico) e R_2(médico,*cod_serv*).

Estas estão sim na FNBC. Esta solução introduz outro problema: é possível registar dois médicos do mesmo serviço para o mesmo paciente! Esta dependência deverá ser tratada ao nível aplicacional! ou então manter R juntamente com R_2.

Por definição, dada uma relação R(X,Y,Z) diz-se que existe uma dependência multivalor X $--\gg$ Y (X multidetermina Y) se, para cada par de tuplos de R contendo os mesmos valores de X também existe em R um par de tuplos correspondentes à troca dos valores de Y no par original. Por simples análise dos tuplos presentes numa relação não é possível concluir da existência de uma dependência multivalor, a não ser que estas representem todos os casos admissíveis de relacionamento entre os dados. Contudo, a mesma análise permite concluir da inexistência de uma dependência multivalor.

Na tabela 2.1, existem 2 dependências multivalor $X --\gg Y$ e $X --\gg Z$, contudo basta que se retire, por exemplo, o primeiro tuplo da relação para que deixem de se verificar as duas dependências multivalor. Por observação das instâncias de uma relação pode-se concluir da não existência de uma dependência multivalor. Essa observação já não é suficiente para concluir da sua existência. Para isso, terão de se conhecer essas regras que governam a manipulação dessa relação. Parece evidente que só em situações muito específicas é que surgem dependências multivalor.

Por exemplo: dada R(Agente,Produto,Zona) e a regra : todos os agentes vendem todos os produtos que representam em todas as zonas em que actuam; i.e.,

(Agente $--\gg$ Produto e Agente $--\gg$ Zona)

A solução é criar duas relações: $R_1(Agente, Produto)$ e $R_2(Agente, Zona)$.

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_1	y_2	z_2
x_2	y_3	z_2
x_2	y_4	z_2
x_2	y_4	z_4
x_2	y_5	z_4
x_3	y_2	z_5

Tabela 2.2: Ilustração de dependências de junção

Na tabela 2.2, projectando em (X,Y), (X,Z) e (Y,Z) não é possível reconstruir a relação R por junção de qualquer destas relações. Apenas se consegue reconstruir R por junção das 3 projecções. A relação não possui uma dependência de junção (5^a FN).

Uma base de dados relaciona um conjunto de tabelas na forma $R(A_1, A_2, \ldots, A_r)$, com grau r. A cardinalidade é o número de elementos de R.

O conjunto dos possíveis valores de um atributo é o domínio $dom(A_i) = D_i$ (e.g., o domínio dum código é o conjunto dos números positivos e o domínio da idade de um trabalhador activo é o conjunto de números inteiros {18,. . .,65}).

Regra 1: não pode haver 2 tuplos idênticos na mesma relação.

Regra 2: não existe uma ordem pré-definida numa relação.

2.4 Álgebra relacional

Na Álgebra Relacional existem dois tipos de operações:

2.4.1 Operações básicas

União: $R \cup S$

Diferença: $R \setminus S$

Produto Cartesiano: $R \otimes S$

Projecção: $\Pi_{A_i,\ldots,A_j}(R) = \Pi_{i,\ldots,j}(R)$

Selecção: $\sigma_{Cond}(R)$

2.4.2 Operações adicionais

Intersecção: $R \cap S$

Divisão: $R \div S$

Θ-junção (e.g.,<-junção): $R \bowtie_{i\Theta j} S$

Junção natural: $R \bowtie S$

Semi-junção: $R \propto S$

São dados alguns exemplos de aplicação dos operadores relacionais:
União:

A	B	C
a_1	b_2	c_1
a_5	b_1	c_2
a_2	b_4	c_4
a_3	b_3	c_3

$\cup$

A	B	C
a_2	b_3	c_2
a_1	b_2	c_1
a_2	b_4	c_4

$=$

A	B	C
a_1	b_2	c_1
a_5	b_1	c_2
a_2	b_4	c_4
a_3	b_3	c_3
a_2	b_3	c_2

Intersecção:

A	B	C
a_1	b_2	c_1
a_5	b_1	c_2
a_2	b_4	c_4
a_3	b_3	c_3

$\cap$

A	B	C
a_2	b_3	c_2
a_1	b_2	c_1
a_2	b_4	c_4

$=$

A	B	C
a_1	b_2	c_1
a_2	b_4	c_4

Diferença:

A	B	C
a_1	b_2	c_1
a_5	b_1	c_2
a_2	b_4	c_4
a_3	b_3	c_3

$\setminus$

A	B	C
a_2	b_3	c_2
a_1	b_2	c_1
a_2	b_4	c_4

$=$

A	B	C
a_5	b_1	c_2
a_3	b_3	c_3

Produto cartesiano:

A	B
a_1	b_2
a_5	b_1
a_2	b_4

$\otimes$

C	D
c_2	b_3
c_1	d_2

$=$

A	B	C	D
a_1	b_2	c_2	d_3
a_1	b_2	c_1	d_2
a_5	b_1	c_2	d_3
a_5	b_1	c_1	d_2
a_2	b_4	c_2	d_3
a_2	b_4	c_1	d_2

Projecção:

$\Pi_{A,C}$

A	B	C
a_2	b_3	c_4
a_1	b_2	c_1
a_2	b_4	c_4

$=$

A	C
a_2	c_4
a_1	c_1

Selecção:

$\sigma_{A=2}$

A	B	C
a_2	b_3	c_4
a_1	b_2	c_1
a_2	b_4	c_4

$=$

A	B	C
a_2	b_3	c_4
a_2	b_4	c_4

Junção:

A	B	C
a_1	b_2	c_1
a_5	b_1	c_1
a_2	b_4	c_4

$\bowtie$

C	D
c_2	d_3
c_1	d_2
c_4	d_1

$=$

A	B	C	D
a_1	b_2	c_1	d_2
a_5	b_1	c_1	d_2
a_5	b_1	c_1	d_2
a_2	b_4	c_4	d_1

Semi-junção:

A	B	C
a_1	b_2	c_1
a_5	b_1	c_1
a_2	b_4	c_4

$\propto$

C	D
c_2	d_3
c_1	d_2
c_4	d_1

$=$

A	B	C
a_1	b_2	c_1
a_5	b_1	c_1
a_2	b_4	c_4

C	D
c_2	d_3
c_1	d_2
c_4	d_1

$\propto$

A	B	C
a_1	b_2	c_1
a_5	b_1	c_1
a_2	b_4	c_4

$=$

C	D
c_1	d_2
c_4	d_1

No caso específico da semi-junção, note-se que, sendo:

$R_1 =$

A	B	C
a_1	b_2	c_1
a_5	b_1	c_1
a_2	b_4	c_4

e

$R_2 =$

C	D
c_2	d_3
c_1	d_2
c_4	d_1

obtem-se:

$R_1 \propto R_2 = R_1$ e $R_2 \propto R_1 \neq R_2$

porque todos os valores de C em R_1 se relacionam com pelo menos um valor de C em R_2 enquanto existe pelo menos um valor de C em R_2 (c_2) que não se relaciona com qualquer valor de C em R_1.

Divisão:

A	B	C	D
a_1	b_2	c_2	d_3
a_5	b_1	c_1	d_2
a_2	b_4	c_1	d_2
a_3	b_5	c_4	d_4
a_2	b_4	c_2	d_3
a_1	b_2	c_1	d_2

$\div$

C	D
c_2	d_3
c_1	d_2

$=$

A	B
a_1	b_2
a_2	b_4

As seguintes regras de equivalência definem os operadores opcionais com base nos operadores básicos:

Sendo R e S duas tabelas do mesmo grau e com a mesma estrutura $R(A_1, \cdots, A_r)$ e $S(A_1, \cdots, A_r)$:

$R \cap S = R \setminus (R \setminus S)$

Sendo $R(A1, \ldots, Ar)$, $S(Ar - s + 1, \ldots, As)$ $r > s$ e $S \neq \oslash$:

$R \div S = \Pi_{1,\ldots,r-s}(R) \setminus \Pi_{1,\ldots,r-s}((\Pi_{1,\ldots,r-s}(R) \otimes S) \setminus R)$

$R \bowtie_{i\Theta j} S = \sigma_{\$i\Theta\$r+j}(R \otimes S)$

$R \bowtie S = \Pi_{i_1,\ldots,i_m}(\sigma_{R.A_i=S.A_i e \ldots e R.A_k=S.A_k}(R \otimes S))$

$R \propto S = \Pi(R \bowtie S)$

A ponte entre a álgebra relacional e a programação em lógica pode ser feita usando as seguintes regras de conversão:

Sendo $R(A_1, \ldots, A_r)$, $S(A_1, \ldots, A_r)$ e $T(A_1, \ldots, A_r)$
$R \cup S = T$ é definido por:

$t(A_1, \ldots, A_r) \leftarrow r(A1, ..., Ar).$

$t(A_1, \ldots, A_r) \leftarrow s(A_1, \leftarrow, A_r).$

$R \setminus S = T$ é definido por:

$t(A_1, \ldots, A_r) \leftarrow r(A_1, \ldots, A_r) \wedge \neg s(A_1, \ldots, A_r).$

Sendo $R(A_1, \ldots, A_r)$, $S(A_{r+1}, \ldots, A_{r+s})$ e $T(A_1, \ldots, A_{r+s})$
$T = R \otimes S$ é definido por:

$t(A_1, \ldots, A_{r+s}) \leftarrow r(A_1, \ldots., A_r) \wedge s(A_{r+1}, \ldots, A_{r+s}).$

Sendo $R(A_1, \ldots, A_r)$, $T(A_i, \ldots, A_j)$ com $i \leq r$ e $j \leq r$:
$\Pi_{A_i,\ldots,A_j}(R) = T$ é definido por:

$t(A_i, \ldots, A_j) \leftarrow r(A_1, \ldots, A_r).$

Sendo $R(A_1, \ldots, A_r)$
$\sigma_{Cond}(R) = T$ é definido por:

$t(A_1, \ldots., A_r) \leftarrow r(A_1, \ldots., A_r) \wedge Cond.$

Sendo $R(A_1, \ldots, A_r)$, $S(A_1, \ldots, A_r)$ e $T(A_1, \ldots, A_r)$
$R \cap S = T$ é definido por:

$t(A_1, \ldots, A_r) \leftarrow r(A_1, \ldots, A_r) \wedge s(A_1, \ldots, A_r).$

$R \setminus (R \setminus S) = T$ é definido por:

$t(A_1, \ldots, A_r) \leftarrow r(A_1, \ldots, A_r) \wedge \neg(r(A_1, \ldots, A_r) \wedge \neg s(A_1, \ldots, A_r))$.

Aplicando a regra : $A \wedge \neg(A \wedge \neg B) = A \wedge (\neg A \vee B) = A \wedge B$
Então:

$R \setminus (R \setminus S) = R \cap S$

Sendo $R(A_1, \ldots, A_r)$, $S(A_{r-s+1}, \ldots, A_r)$ e $T(A_1, \ldots, A_{r-s})$:
Para calcular $R \div S = T$ define-se:

$R \div S = \Pi_{1,\ldots,r-s}(R) \setminus \Pi_{1,\ldots,r-s}((\Pi_{1,\ldots,r-s}(R) \otimes S) \setminus R)$

$r_1(A_1, \ldots, A_{r-s}) \leftarrow r(A_1, \ldots, A_r)$.

$r_2(A_1, \ldots, A : r) \leftarrow r_1(A_1, \ldots, A_{r-s}) \wedge s(A_{r-s+1}, \ldots, A_r)$.

$r_3(A_1, \ldots, A_r) \leftarrow r_2(A_1, \ldots, A_r) \wedge \neg r(A_1, \ldots, A_r)$.

$r_4(A_1, \ldots, A_{r-s}) \leftarrow r_3(A_1, \ldots, A_r)$.

$t(A_1, \ldots, A_{r-s}) \leftarrow r_1(A_1, \ldots, Ar - s) \wedge \neg r_4(A_1, \wedge, A_{r-s})$.

sendo

$r_1 = \Pi_{1,\ldots,r-s}(R)$

$r_2 = r_1 \otimes S$

$r_3 = r_2 \setminus R$

$r_4 = \Pi_{1,\ldots,r-s}(r_3)$

$R \bowtie_{i\Theta j} S = T$ define-se por:

$t(A_1, \ldots, A_{r+s}) \leftarrow r(A_1, \ldots, A_r) \wedge s(A_{r+1}, \ldots, A_{r+s}) \wedge A_i \Theta A_{r+j}$.

Sendo k o número de atributos comuns:
$R \bowtie S = T$ define-se por:

$t(A_1, \ldots, A_{r+s-k}) \leftarrow r(A_1, \ldots, A_r) \wedge$

$$s(A_{r+1}, \ldots, A_{r+s}) \wedge A_{i1} = A_{j1} \wedge \ldots \wedge A_{ik} = A_{jk}.$$

Capítulo 3

A linguagem SQL

3.1 Introdução

O SQL é uma linguagem usada para comunicar com bases de dados. De acordo com a ANSI (American National Standards Institute) é a linguagem standard para os sistemas de gestão de bases de dados relacionais. Os comandos da SQL são usados para realizar operações tais como alterar / actualizar dados numa base de dados, ou responder a questões a partir da informação armazenada numa base de dados. Os seguintes motores de bases de dados usam SQL: Oracle, Informix, DB2, Sybase, Microsoft SQL Server, Access, Ingres, etc. A maior parte dos sistemas gestores de bases de dados usam SQL mas têm as suas próprias extensões ao SQL. No entanto os comandos mais comuns da SQL tais como "SELECT", "INSERT", "UPDATE", "DELETE", "CREATE"e "DROP"podem ser usados para fazer a maioria das operações necessárias numa base de dados.

3.2 O comando SELECT

O comando SELECT é usado para interrogar a base de dados e recuperar os dados selecionados e que verificam as condições especificadas. A sintaxe básica do comando SELECT é:

```
SELECT "coluna1"[,"coluna2",etc]
 FROM "nometabela"
[WHERE "condição"];

[ ] = opcional
```

Os nomes das colunas que seguem a palavra SELECT determinam as colunas a apresentar como resultado. Pode-se selecionar um número variável de colunas ou usar um metacaracter "*" para selecionar todas as colunas. O nome da tabela após a palavra FROM especifica a tabela a interrogar. A claúsula WHERE

(opcional) especifica que valores ou linhas devem ser apresentados, baseando-se na condição descrita após a palavra WHERE.

Os seguintes operadores podem ser numa claúsula WHERE :

=	Igual
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
<>	Diferente
LIKE	Semelhante

Like é um operador muito versátil que permite selecionar todas as linhas que são parecidas com uma dada sequência. O símbolo de percentagem "%" pode ser usado como um metacaracter que substituí qualquer sequência de caracteres. Por exemplo:

```
SELECT primeiro, ultimo, localidade
FROM empregado
WHERE primeiro LIKE ‘Lu%’;
```

O comando seleciona qualquer atribuído primeiro que comece por “Er”. Os valores do tipo alfanumérico (strings) devem estar entre “”.

Por exemplo:

```
SELECT primeiro, ultimo
FROM empregado
WHERE ultimo LIKE ’%s’;
```

O comando seleciona qualquer atribuído primeiro que termine com “s”.

```
SELECT * FROM empregado
WHERE primeiro = ‘Luis’;
```

Este comando apenas seleciona a linha cuja atributo primeiro é igual a Luis.

Outros exemplos da utilização do comando SELECT são:

```
SELECT primeiro, ultimo, localidade FROM empregado;

SELECT ultimo, localidade, idade FROM empregado
WHERE idade > 30;

SELECT primeiro, ultimo, localidade, estado FROM empregado
WHERE primeiro LIKE ’J%’;

SELECT * FROM empregado;
```

```
SELECT primeiro, ultimo, FROM empregado
WHERE último LIKE '%s';

SELECT primeiro, ultimo, idade FROM empregado
WHERE último LIKE '%ui%';

SELECT * FROM empregado WHERE primeiro = 'Luis';
```

3.3 O comando CREATE TABLE

O comando "CREATE TABLE" é usado para criar uma tabela:

```
CREATE TABLE "nometabela"
("coluna1" "data type",
"coluna2" "data type",
"coluna3" "data type");
```

no caso de usar restrições opcionais:

```
CREATE TABLE "nometabela"
("coluna1" "data type" [restrição],
"coluna2" "data type" [restrição],
"coluna3" "data type" [restrição]);

[ ] = opcional
```

Por exemplo:

```
CREATE TABLE empregado
(primeiro varchar(15),
ultimo varchar(20),
idade número(3),
morada varchar(30),
localidade varchar(20),
estado varchar(20),
primary key(primeiro,ultimo));
```

Os tipos de dados especificam o tipo dos valores a armazenar numa coluna. Se uma coluna chamada "Último_Nome"vai suportar nomes, então o tipo deve ser "varchar"(caracter de comprimento variável).

char(tamanho)	String de comprimento fixo	O tamanho está especificado entre parênteses. Max 255 bytes.
varchar(tamanho)	String de comprimento variável	O tamanho máximo está especificado entre parênteses.
número(tamanho)	Valor numérico inteiro	com um número máximo de dígitos "tamanho"
data	Valor de data e/ou hora	
número(tamanho,d)	Valor numérico	com um número máximo de dígitos "tamanho" e um número máximo de casas decimais "d".

Estão também disponíveis os comandos ALTER TABLE e DROP TABLE para fazer, respectivamente, alterações à estrutura de uma tabela e remover a tabela e o seu respectivo conteúdo. Exemplos de utilização dos comandos SQL CREATE TABLE, ALTER TABLE e DROP TABLE são apresentados de seguida no contexto do SQL*PLUS:

3.4 Informação sobre uma tabela

O comando DESCRIBE (ou DESC) dá a estrutura de uma tabela

```
DESCRIBE tabela
```

3.5 O comando ALTER TABLE

Para alterar a estrutura de uma tabela deve-se usar o comando ALTER TABLE

```
alter table tabela add(  nomecoluna tipo, ...);
alter table tabela modify ( coluna novotipo);
alter table drop coluna;
```

```
SQL> create table sign_sales(Color varchar2(30),date_sold date,
  2  price_each number);

Table created.
SQL> alter table students add( sign_shape number);

Table altered.

SQL> alter table students modify (sign_shape varchar2(10));

Table altered.
```

```
SQL> alter table students drop column sign_shape ;

Table altered.
SQL> DESCRIBE sign_shape
Name                      Null?    Type
----------------- -------- ------------------
COLOR                                VARCHAR2(30)
DATE_SOLD                            DATE
PRICE_EACH                           NUMBER
```

3.6 O comando DROP TABLE

Para eliminar uma tabela usa-se o comando DROP TABLE

```
drop table tabela;
```

Exemplo:

```
DROP TABLE empregado;

SQL> drop table sign_sale;

Table dropped.
```

A seguinte script elimina todas as tabelas:

```
SET NEWPAGE 0
SET SPACE 0
SET LINESIZE 80
SET PAGESIZE 0
SET ECHO OFF
SET FEEDBACK OFF
SET HEADING OFF
SET MARKUP HTML OFF
SET ESCAPE \
SPOOL DELETEME.SQL
select 'drop table ', table_name, 'cascade constraints \;' from user_tables;
SPOOL OFF
@DELETEME
```

3.7 Restrições

Constraints são restrições sobre dados que devem manter-se verdadeiros em situação de mudança de estado, i.e., em transacções. Podem ser de diferentes

tipos: baseadas no atributo, baseadas no tuplo, definição de chaves ou restrições referenciais. O sistema verifica se uma acção viola as restrições e em caso positivo aborta a execução da mesma. A implementação de *constraints* no Oracle é um pouco diferente do SQL standard.

Quando as tabelas são criadas, é normal associar a uma ou mais colunas algumas restrições. Uma restrição é basicamente uma regra associada a uma coluna que deve ser respeitada por todos os dados que forem armazenados nessa coluna. Por exemplo, a restrição "unique"especifica que não é possível dois registos diferentes terem essa coluna com o mesmo valor. As outras duas restrições mais populares são o "not null"que especifica que uma coluna não pode ter o valor nulo, e "primary key". A restrição "primary key"define uma identificação única para cada registo. A seguinte script permite criar / recriar uma base de dados:

3.7.1 Exemplo

```
drop table student cascade constraints;
drop table faculty cascade constraints;
drop table class cascade constraints;
drop table enrolled cascade constraints;
drop table emp cascade constraints;
drop table works cascade constraints;
drop table dept cascade constraints;
drop table flights cascade constraints;
drop table aircraft cascade constraints;
drop table certified cascade constraints;
drop table employees cascade constraints;
drop table suppliers cascade constraints;
drop table parts cascade constraints;
drop table catalog cascade constraints;
drop table sailors cascade constraints;

create table student(
snum number(9,0) primary key,
sname varchar2(30),
major varchar2(25),
standing varchar2(2),
age number(3,0)
);
create table faculty(
fid number(9,0) primary key,
fname varchar2(30),
deptid number(2,0)
);
create table class(
name varchar2(40) primary key,
```

```
meets_at varchar2(20),
room varchar2(10),
fid number(9,0),
foreign key(fid) references faculty
);
create table enrolled(
snum number(9,0),
cname varchar2(40),
primary key(snum,cname),
foreign key(snum) references student,
foreign key(cname) references class(name)
);
create table emp(
eid number(9,0) primary key,
ename varchar2(30),
age number(3,0),
salary number(10,2)
);
create table dept(
did number(2,0) primary key,
dname varchar2(20),
budget number(10,2),
managerid number(9,0),
foreign key(managerid) references emp(eid)
);
create table works(
eid number(9,0),
did number(2,0),
pct_time number(3,0),
primary key(eid,did),
foreign key(eid) references emp,
foreign key(did) references dept
);
create table flights(
flno number(4,0) primary key,
origin varchar2(20),
destination varchar2(20),
distance number(6,0),
departs date,
arrives date,
price number(7,2)
);
create table aircraft(
aid number(9,0) primary key,
aname varchar2(30),
cruisingrange number(6,0)
```

```
);
create table employees(
eid number(9,0) primary key,
ename varchar2(30),
salary number(10,2)
);
create table certified(
eid number(9,0),
aid number(9,0),
primary key(eid,aid),
foreign key(eid) references employees,
foreign key(aid) references aircraft
);
create table suppliers(
sid number(9,0) primary key,
sname varchar2(30),
address varchar2(40)
);
create table parts(
pid number(9,0) primary key,
pname varchar2(40),
color varchar2(15)
);
create table catalog(
sid number(9,0),
pid number(9,0),
cost number(10,2),
primary key(sid,pid),
foreign key(sid) references suppliers,
foreign key(pid) references parts
);
create table sailors(
sid number(9,0) primary key,
sname varchar2(30),
rating number(2,0),
age number(4,1)
);
```

3.7.2 Adição de restrições

```
create table tabela ( coluna tipo, coluna tipo ...,
primary key(colunachave,colunachave,...);
```

sendo colunachave o nome de uma coluna que é parte da chave.

As chaves estrangeiras devem referir-se a tuplos únicos.

```
create table tabela ( coluna tipo, coluna tipo...,
```

```
primary key(colunachave,colunachave,...),
foreign key(colunachaveestrangeira, colunachaveestrangeira,...)
references tabelaestrangeira,
foreign key(colunachaveestrangeira, colunachaveestrangeira,...)
references tabelaestrangeira,...);

Alter table tablename add tableconstraint;
```

3.7.3 Observação de restrições

Para ver as restrições associadas a uma tabela deve-se usar a view USER_CONSTRAINTS. Execute:

```
DESCRIBE USER_CONSTRAINTS
```

para mais informação.

```
select column_name, position, constraint_name from User_cons_columns;
```

3.7.4 Utilização de restrições

```
create table tabela ( coluna tipo, coluna tipo ...,
foreign key(colunachaveestrangeira,colunachaveestrangeira,...)
references tabelaestrangeira deferrable);
set constraints all deferred;
set constraints all immediate;
```

3.7.5 Eliminação de restrições

```
alter table tabela drop constraint umarestrição;
```

Exemplo:

```
SQL> create table bids( bid_id varchar2(10), bidder_id varchar2(10),
item_id varchar2(10),
  2  bid_amount number, primary key(bid_id),
foreign key ( item_id ) references
  3  auction_items, foreign key (bidder_id) references
members(member_id) deferrable);

Table created.

SQL>
SQL> select constraint_name, constraint_type from user_constraints where
  2  table_name='BIDS';
```

```
CONSTRAINT_NAME                C
------------------------------ -
SYS_C001400                    P
SYS_C001401                    R
SYS_C001401                    R

SQL>
SQL> alter table students drop constraint SYS_C001400;

Table altered.

SQL> alter table students add primary key( bid_id );

Table altered.

SQL> set constraints all deferred;

Constraint set.
```

3.7.6 Adiamento da verificação de Constraint

É às vezes necessário adiar a verificação de alguma *constraint*, por exemplo:

```
CREATE TABLE galinha (cID INT PRIMARY KEY,
                      eID INT REFERENCES ovo(eID));

CREATE TABLE ovo(eID INT PRIMARY KEY,
                 cID INT REFERENCES galinha(cID));
```

Estas definições provocam um erro no Oracle, porque na criação da tabela galinha é feita referência à tabela *ovo* que ainda não existe. Alterar a ordem não resolve o problema. Para resolver o problema deve-se criar primeiro as tabelas sem restrições:

```
CREATE TABLE galinha(cID INT PRIMARY KEY,
                     eID INT);
CREATE TABLE ovo(eID INT PRIMARY KEY,
                 cID INT);
```

E depois adicionar as restrições (chaves estrangeiras):

```
ALTER TABLE galinha ADD CONSTRAINT galinhaREFovo
    FOREIGN KEY (eID) REFERENCES ovo(eID)
    INITIALLY DEFERRED DEFERRABLE;
ALTER TABLE ovo ADD CONSTRAINT ovoREFgalinha
    FOREIGN KEY (cID) REFERENCES galinha(cID)
```

```
    INITIALLY DEFERRED DEFERRABLE;
```

INITIALLY DEFERRED DEFERRABLE permite adiar a verificação de restrições até ao comando COMMIT. Por exemplo:

```
INSERT INTO galinha VALUES(1, 2);
INSERT INTO ovo VALUES(2, 1);
COMMIT;
```

Para eliminar as tabelas deve-se eliminar primeiro as *constraints*

```
ALTER TABLE ovo DROP CONSTRAINT ovoREFgalinha;
ALTER TABLE galinha DROP CONSTRAINT galinhaREFovo;
DROP TABLE ovo;
DROP TABLE galinha;
```

3.7.7 Violação de Constraint

O Oracle retorna uma mensagem de erro em caso de violação de "constraint". Dependendo do tipo de acesso à base de dados (ODBC, JDBC, PL/SQL) o controlo de erros pode ser implementado. O Oracle disponibiliza mensagens de erro tais como:

```
ORA-02290: check constraint (YFUNG.GR_GR) violated
```

ou

```
ORA-02291: integrity constraint (HONDROUL.SYS_C0067174)
violated - parent key not found
```

3.8 O Comando INSERT

Os valores são inseridos pela ordem das colunas na estrutura da tabela. O primeiro valor é inserido na primeira coluna e assim sucessivamente.

O comando INSERT é usado para inserir ou adicionar dados a uma tabela:

```
INSERT INTO "nometabela"
[(primeira_coluna,...última_coluna)]
VALUES (primeira_valor,...última_valor);
[ ] = opcional
```

Exemplo:

```
INSERT INTO empregado
(primeiro, ultimo, idade, morada, localidade, estado)
VALUES ('Luis', 'Duque', 45, 'Rua dos Bragas, 20', 'Braga', 'Minho');
```

O comando INSERT INTO VALUES insere um e um só registo de cada vez que é usado. É possível inserir de 0 a N registos (sendo N a cardinalidade da selecção determinada pela claúsula SELECT) usando INSERT INTO SELECT.

```
INSERT INTO empregado_temp
(primeiro, ultimo, idade, morada, localidade, estado)
SELECT primeiro, ultimo, idade, morada, localidade, estado
FROM empregado where localidade  = 'Braga'
```

É importante aqui que as duas tabelas (neste caso empregado_temp e empregado) tenham estruturas compatíveis. Se tiverem a mesma estrutura pode-se executar:

```
INSERT INTO empregado_temp
SELECT *
FROM empregado where condição
```

3.9 O Comando UPDATE

O comando UPDATE é usado para actualizar registos:

```
UPDATE "nometabela"
SET "nomecoluna" = "novovalor"[,"proxcoluna" = "novovalor2"...]
WHERE "nomecoluna" OPERADOR "valor"
[OPERADOR "coluna" OPERADOR "valor"];
[ ] = opcional
```

Exemplos:

```
UPDATE agenda SET indicativo = '253'
WHERE localidade = 'Braga';

UPDATE agenda
SET ultimo_nome = 'Ferreira', indicativo='253'
WHERE ultimo_nome = 'Fereira';

UPDATE empregado
SET idade = idade+1
WHERE primeiro_nome='Maria' and último_nome='Pereira';
```

Exercício: desenvolva as produções em SQL que lhe permitam fazer as seguintes actualizações de estado da base de dados:

1. Joana Ferreira casou-se com Paulo Gomes. O seu último nome mudou para Ferreira Gomes.
2. Daniel Santos faz anos hoje. Vai ficar um ano mais velho.
3. Todas as secretárias serão agora intituladas “Assistentes Administrativas”.
4. Todos aqueles que ganham menos de 30000 vão ter um aumento de 3500.
5. Todos aqueles que ganham mais de 33500 irão ter um aumento de 4500.
6. Todos aqueles que auferem o título de ”Programador II”vão ser promovidos a ”Programador III”.
7. Todos aqueles que auferem o título de ”Programador”vão ser promovidos a ”Programador II”.

3.10 O Comando DELETE

O comando DELETE é usado para apagar / eliminar registos:

```
DELETE FROM "nometabela"
WHERE "nomecoluna" OPERADOR "valor"
[OPEARDOR "coluna" OPERADOR "valor"];
[ ] = opcional
```

Exemplos:

```
DELETE FROM empregado;
```

Nota: todos os registos são removidos!

```
DELETE FROM empregado
WHERE últimonome = ‘Neves’;
```

```
DELETE FROM empregado
WHERE primeironome = ’Miguel’ or primeironome = ‘Manuel’;
```

Exercício: use o comando SELECT para verificar as eliminações:

1. A Joana Ferreira Gomes foi embora;
2. É tempo de poupar. Os empregados que ganham mais de 7000 vão ser despedidos.

3.11 Exemplo de Sessão

```
SQL> insert into signs values
(19.95,'White','Rectangle','Park Somewhere Else');

1 row created.

SQL>
SQL> select * from signs;

PRICE_EACH COLOR     SHAPE   DESCRIPTION
---------- --------- ------------ -------------------
19.95      White     Rectangle    Park Somewhere Else

SQL>
SQL> update signs set price_each = 4.00 where price_each < 20;

1 row updated.

SQL>
SQL> delete from signs;

1 row deleted.

SQL>
```

3.12 O comando SELECT - conceitos avançados

A sintaxe completa do comando SELECT é a seguinte:

```
SELECT [ALL | DISTINCT] coluna1[,coluna2]
FROM table1[,table2]
[WHERE "condições"]
[[GROUP BY "coluna-lista"]
[HAVING "condições]]
[ORDER BY "coluna-lista" [ASC | DESC] ]
```

As palavras reservadas ALL e DISTINCT são usadas para SELEC(T)ionar todos (ALL) ou os valores sem repetições.

Por exemplo:

```
SELECT DISTINCT idade
FROM empregado_info;
```

O comando SELECT pode ser usados para calcular operações de agregação com grupos:

```
SELECT coluna-1ista, SUM(coluna2)
FROM "lista-de-tabelas"
GROUP BY "coluna-lista";

Por exemplo:

SELECT max(Vencimento), dept
FROM empregado
GROUP BY dept;

SELECT quantidade, max(preço)
FROM artigos_encomendados
GROUP BY quantidade;
```

É possível estabelecer uma condição que restrinja o grupo:

```
SELECT coluna1, SUM(coluna2)
FROM "lista-de-tabelas"
GROUP BY "colunal"
HAVING "condição";
```

Por exemplo:

```
SELECT dept, avg(Vencimento)
FROM empregado
GROUP BY dept;

SELECT dept, avg(Vencimento)
FROM empregado
GROUP BY dept
HAVING avg(Vencimento) > 20000;
```

A ordenação é implementada usando a claúsula ORDER BY.

```
SELECT coluna1, SUM(coluna2)
FROM "lista-de-tabelas"
ORDER BY "coluna-lista" [ASC | DESC];
[ ] = opcional
ASC = Ordem ascendente - defeito
DESC = Ordem descendente
```

Por omissão, a ordem é decrescente, por isso a palavra DESC é dispensável enquanto isto não é verdade para a ordem ascendente (ASC).

Por exemplo:

```
SELECT empregado_id, dept, nome, idade, Vencimento
FROM empregado_info
WHERE dept = 'Vendas'
ORDER BY vencimento;

SELECT empregado_id, dept, nome, idade, Vencimento
FROM empregado_info
WHERE dept = 'Vendas'
ORDER BY vencimento, idade ASC;
```

As funções de agregação são enunciadas na tabela:

MIN	retorna o valor mais pequeno duma dada coluna
MAX	retorna o valor maior duma dada coluna
AVG	Retorna a média de todos os valores numéricos duma coluna
COUNT	Retorna a cardinalidade do conjunto de todos os registos
COUNT(*)	Retorna a cardinalidade da tabela

Por exemplo as seguintes operações retornam um valor e não uma tabela como foi sempre visto até este momento.

```
SELECT AVG(Vencimento)
FROM empregado;

SELECT AVG(Vencimento)
FROM empregado
WHERE Cargo = 'Programador';

SELECT Count(*)
FROM empregados;
```

É possível usar-se condições lógicos via operadores lógicos:

```
SELECT coluna1, SUM(coluna2)
FROM "lista-de-tabelas"
WHERE "condição1" AND "condição2";
```

Ou

```
 SELECT empregadoid, primeironome, últimonome, Cargo, Vencimento
FROM empregado_info
```

```
WHERE vencimento >= 50000.00 AND cargo = 'Programador';

 SELECT empregadoid, primeironome, ultimonome, cargo, vencimento
FROM empregado_info
WHERE (vencimento >= 50000.00) AND (cargo = 'Programador');

 SELECT primeironome, últimonome, Cargo, vencimento
FROM empregado_info
WHERE (cargo = 'Vendas') OR (cargo = 'Programador');
```

Os operadores IN e BETWEEN simplificam também a escrita das condições:

```
SELECT coluna1, SUM(coluna2) FROM lista-de-tabelas
WHERE coluna3 IN (lista-de-valores);

SELECT coluna1, SUM(coluna2) FROM lista-de-tabelas
WHERE coluna3 BETWEEN valor1 AND valor2;
```

Exemplo:

```
 SELECT empregadoid, ultimonome, Vencimento
FROM empregado_info
WHERE ultimonome IN ('Hernandez', 'Jones', 'Roberts', 'Ruiz');
```

É equivalente a:

```
SELECT empregadoid, ultimonome, Vencimento
FROM empregado_info
WHERE ultimonome = 'Hernandez' OR últimonome = 'Jones'
 OR ultimonome = 'Roberts' OR últimonome = 'Ruiz';
```

Assim como:

```
SELECT empregadoid, idade, ultimonome, Vencimento
FROM empregado_info
WHERE idade BETWEEN 30 AND 40;
```

é equivalente a:

```
SELECT empregadoid, idade, ultimonome, Vencimento
FROM empregado_info
WHERE idade >= 30 AND idade <= 40;
```

No SQL, podem-se usar operadores tais como:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo

Assim como:

ABS(x)	retorna o valor absoluto de x
SIGN(x)	retorna o sinal de x (-1, 0, ou 1) (negativo, zero, ou positivo)
MOD(x,y)	módulo - retorna o resto da divisão inteira de x por (x%y)
FLOOR(x)	retorna o maior inteiro que é menor ou igual a x
CEILING(x) or CEIL(x)	retorna o menor inteiro que é maior ou igual a x
POWER(x,y)	retorna o valor de x elevado à potência de y
ROUND(x)	retorna o valor de x arredondo ao inteiro mais próximo
ROUND(x,d)	retorna o valor de x arredondado ao número com d casas decimais mais próximo
SQRT(x)	retorna a raíz quadrada de x

Exemplo:

```
SELECT round(vencimento), primeironome FROM empregado_info
```

Este comando seleciona o vencimento arredondado ao inteiro mais próximo e o primeironome do empregado.

Todas as questões colocadas até agora têm uma ligeira limitação. O comando SELECT aplica-se a apenas uma tabela. É tempo de introduzir agora uma das características mais importantes das bases de dados relacionais – o "join", o operador que torna os sistemas de bases de dados relacionais "relacionais".

As junções permitem ligar dados de duas ou mais tabelas, de forma a obter uma única tabela como resultado. Uma "junção" reconhece-se num SELECT que houver mais de uma tabela referida na cláusula FROM.

Exemplo:

```
SELECT "lista-de-colunas"
FROM table1,table2
WHERE "condição(ões)_pesquisa"
```

Agora, sempre que um cliente já registado efectuar uma compra, apenas a segunda tabela *vendas* necessita de ser actualizada. Os dados redundantes foram assim eliminados e a base de dados está normalizada. Nota-se que ambas as tabelas tem uma coluna comum *cliente_número* coluna. Essa coluna, contem o número único do cliente será usada para juntar as duas tabelas. Usando as duas tabelas, se se quiser saber o nome dos clientes e os artigos que compraram, o seguinte comando pode resolver essa questão:

```
SELECT cliente_info.primeironome, cliente_info.ultimonome,
vendas.artigo
FROM cliente_info, vendas
WHERE cliente_info.cliente_numero = vendas.cliente_numero;
```

Esta junção particular é conhecida por "Inner Join"ou "Equijoin". É o tipo mais conhecida de junção. Nota-se que o identificador de cada coluna é precedido pelo nome da tabela e um ponto. Não é obrigatório, mas é um bom hábito fazê-lo para que em caso de ambiguidade se saiba o significado das colunas e a que tabelas elas pertencem. Por exemplo é obrigatório quando as colunas comum têm o mesmo nome.

```
SELECT cliente_info.primeironome, cliente_info.ultimonome,
vendas.artigo
FROM cliente_info INNER JOIN vendas
ON cliente_info.cliente_número = vendas.cliente_numero;
```

As operações da álgebra relacional podem ser traduzidas para SQL:

União	(select * from R) union (select * from S)
Diferença	(select * from R) except (select * from S) ou select * from R where R.* not in (select * from S)
Produto	select R.*,S.* from R,S
Projecção	select Ai,..,Aj from R
Selecção	select * from R where Cond
Intersecção:	(select * from R) intersect (select * from S) ou select * from R where R.* in (select * from S)
Θ-junção	select R.*,S.* from R,S where Ai Θ Ar+j
Junção	select R.*,S.Ar+1,S.Ar+s-k from R,S where R.Ai1 = S.Ai1 and ... and R.Aik = S.Aik

Exercício:
Dado o esquema da base de dados da Figura 3.1 apresente os comandos SQL para criar a base de dados.

Solução (neste exemplo as tabelas deve ser criadas por esta ordem para evitar que o processo de criação de chaves estrangeiras não origine um erro por inexistência da chave referenciada):

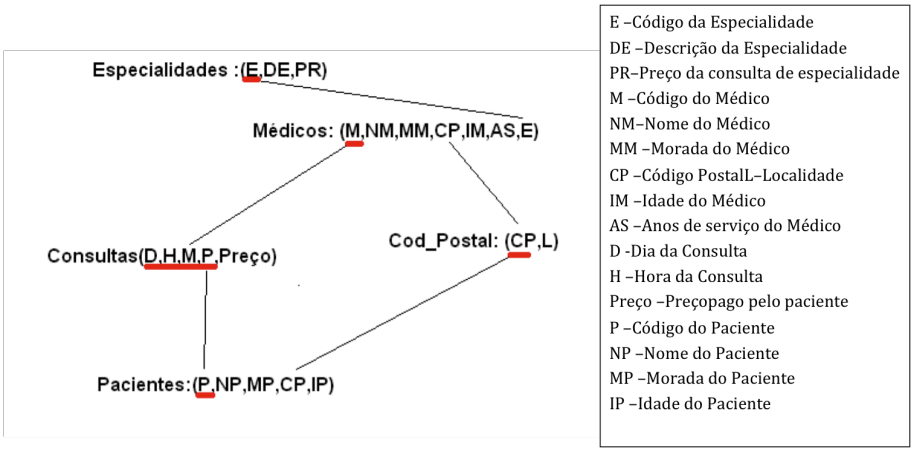

Figura 3.1: Esquema conceptual da base de dados

```
create table cod_postal
(CP varchar2(4) primary key,
L varchar2(100));

create table especialidades
(e number(2) primary key,
de varchar2(40),
pr number(12,2)
);

create table medicos
(M number(3) primary key,
NM varchar2(50),
MM varchar2(100),
CP varchar2(4),
DN date,
AX number(2),
E number(2),
foreign key(E) references especialidades(E),
foreign key(cp) references cod_postal(cp)
);

create table pacientes(
P number(5) primary key,
NP varchar2(50),
MP varchar2(100),
CP varchar2(4),
```

```
DN date,
foreign key(CP) references cod_postal(cp)
);


create table consultas(
DH date,
M number(3),
P number(5),
Preco number(12,2),
primary key(DH,M,P),
foreign key(M) references medicos(M),
foreign key(P) references pacientes(P)
);
```

Para eliminar as tabelas da base de dados e voltar a criá-las deverá ser executado o seguinte programa. Atenção à ordem dos comandos DROP por causa das chaves estrangeiras. Neste caso, obtou-se por criar as chaves estrangeiras de outra forma, podendo ser por qualquer ordem.

```
drop table consultas;
drop table pacientes;
drop table medicos;
drop table cod_postal;
drop table especialidades;

create table especialidades
(e number(2) primary key,
de varchar2(40),
pr number(12,2)
);

create table medicos
(M number(3) primary key,
NM varchar2(50),
MM varchar2(100),
CP varchar2(4),
DN date,
AX number(2),
E number(2)
);

create table consultas(
DH date,
```

```
M number(3),
P number(5),
Preco number(12,2),
primary key(DH,M,P)
);

create table pacientes(
P number(5) primary key,
NP varchar2(50),
MP varchar2(100),
CP varchar2(4),
DN date
);

create table cod_postal
(CP varchar2(4) primary key,
L varchar2(100));

alter table medicos add
(foreign key(E) references especialidades(E),
foreign key(cp) references cod_postal(cp)
);

alter table consultas add (
foreign key(M) references medicos(M),
foreign key(P) references pacientes(P)
);

alter table pacientes add(
foreign key(CP) references cod_postal(cp)
);
```

Resolva:

1. Qual é o nome dos médicos com mais de 10 anos de serviço?

   $\Pi_{NM}(\sigma_{AS>10} medicos)$

   ```
   Select NM from medicos where AS > 10
   ```

2. Indique o nome dos médicos e respectiva especialidade.
   $\Pi_{NM,DE}(medicos \bowtie_E especialidades)$

```
Select NM,DE from especialidades,medicos
where especialidades.E = medicos.E
```

3. Qual é o par (nome e morada) dos pacientes residentes em Braga?
   $\Pi_{NP,MP}(pacientes \bowtie_{CP} \sigma_{L='Braga'} cod\_postal)$

```
Select NP,MP from pacientes,cod_postal
where pacientes.cp = cod_postal.cp and cod_postal.L = 'Braga'
```

4. Qual é o nome dos médicos da especialidade de oftalmologia?
   $\Pi_{NM}(medicos \bowtie_{E} \sigma_{DE='Oftalmologia'} especialidades)$

```
select NM from medicos,especialidades
where medicos.E = especialidades.E and
especialidades.DE = 'Oftalmologia'
```

5. Quais são os médicos com mais de 40 anos com a especialidade de Clínica Geral?
   $\sigma_{idade(DN)>40} medicos \propto_{E} \sigma_{DE='ClnicaGeral'} especialidades$

```
select me.* from medicos me,especialidades es
where me.E = es.E and (sysdate - me.DN) / 365.25  > 40 and es.DE = 'Clínica Geral'
```

6. Quais são os médicos de Oftalmologia que consultaram pacientes de Braga?
   $(medicos \propto_{E} \sigma_{DE='Oftalmologia'} especialidades)$
   $\propto_{M} (consultas \bowtie_{P} pacientes \bowtie_{CP} \sigma_{L='Braga'} cod\_postal)$

```
select me.* from especialidades es,medicos me,consultas co,
pacientes pa, cod_postal cl
where es.e = me.e and co.m = me.m and co.p = pa.p
and cl.cp = pa.cp and es.de = 'Oftalmologia' and L = 'Braga'
```

7. Quais são os médicos com mais de 50 anos que deram consultas de tarde a pacientes com menos de 20 anos?

   $\sigma_{idade(DN)>50} medicos \propto_{M} (\sigma_{hora(DH)>'12:00'} consultas \bowtie_{P} \sigma_{IP<20} pacientes)$

```
Select me.* from medicos me,consultas co, pacientes pa
where me.m = co.m and co.p = pa.p and (sysdate - me.dn) / 365.25  > 50
and format(co.Dh,'HH24:mi') > '12:00' and pa.ip < 20
```

8. Quais os pacientes com mais de 10 anos que nunca foram consultados a Oftalmologia?
$\sigma_{idade(DN)>10}pacientes \backslash (\sigma_{idade(DN)>10}pacientes \propto_P (consultas \bowtie_M medicos \bowtie_E \sigma_{DE='Oftalmologia'}especialidades))$

```
Select pa.* from pacientes pa
where (sysdate - pa.dn) / 365.25 > 10 and not exists
(select * from consultas co,medicos me, especialidades es
where me.m = co.m and co.p = pa.p and me.E = es.E and es.de = 'Oftalmologia')
```

9. Quais são as especialidades consultadas no mês de Janeiro de 2002?
$especialidades \propto_E (medicos \bowtie_M \sigma_{month(DH)=1 \wedge year(DH)=2002}consultas)$

```
select es.* from especialidades es,medicos me, consultas co
where to_char(DH,'yyyy') = '2002' and to_char(DH,'mm') = '1' and
es.E = me.E and co.M = me.M
```

10. Qual é o nome dos médicos com mais de 30 anos ou menos de 5 anos de serviço?
$\Pi_{NM}(\sigma_{idade(DN)>30 \vee AS<5}medicos)$

```
select me.NM from medicos me
where me.AS < 5 or me.IM >30
```

11. Quais são os médicos de Clínica Geral que não consultaram em Janeiro de 2002?
$medicos \propto_E \sigma_{DE='ClinicaGeral'}especialidades \backslash$
$(medicos \propto_M (\sigma_{month(D)=1 \wedge year(D)=2002}consultas))$

```
select me.* from medicos me,especialidades es
where DE = 'Clínica Geral' and es.E = me.E
minus
select me.* from medicos me, consultas co
where to_char(DH,'yyyymm') = '200201'
```

12. Quais são os pacientes que já foram consultados por todos os médicos?

$(\Pi_{pacientes.*,M}(pacientes \bowtie_P consultas \bowtie_M medicos) \div \Pi_M medicos$

```
select pa.* from pacientes pa where not exists
(select * from medicos me
where me.M not in
(select co.M from consultas co where cp.P = pa.P)
```

13. Quais são as especialidades que não foram consultadas durante os meses de Janeiro e Março de 2002?
$especialidades \setminus$
$especialidades \propto_E (medicos \bowtie_E \sigma_{momth(D)=1 \wedge year(D)=2002} consultas)$

```
select es.* from especialidades es
where not exists
(select co.* from consultas co where co.M in
(select me.M from medicos me where me.E = es.E)
)
```

14. Quais são os médicos que nunca consultaram pacientes de Braga?
$medicos \setminus$
$medicos \bowtie_M (consultas \bowtie_P pacientes \bowtie_{CP} \sigma_{L='Braga'} cod\_postal)$

```
select me.* from medicos me
where not exists
(select co.* from consultas co where co.M = me.M and
co.P in (select pa.P from pacientes pa where pa.cp in
(select cp.cp from cod_postal cp
where cp.L = 'Braga')
)
)
```

15. Quais são os pacientes que só foram consultados a Clínica Geral?

$pacientes \propto_P$
$(consultas \bowtie_P medicos \bowtie_E \sigma_{DE='ClinicaGeral'} especialidades) \setminus$
$pacientes \propto_P$
$(consultas \bowtie_P medicos \bowtie_E \sigma_{DE\neq'ClinicaGeral'} especialidades)$

```
select pa.* from pacientes pa where pa.P not in
(select co.P from consultas co where co.M in
(select me.M from medicos me where me.E in
(select es.E from especialidades es
 where es.DE <> 'Clínica Geral'))) and exists
 (select co2.* from consultas co2 where co2.P = pa.P)
```

Exercícios
Dada a Figura 3.1:

1. Qual é a média de idades dos médicos com mais de 15 anos de serviço?
2. Qual é a média de anos de serviço dos médicos para cada uma especialidade?
3. Quantas consultas estão registadas por médico?
4. Qual é a média de idades dos médicos por especialidade?
5. Para cada médico apresentar o valor facturado em 2003.
6. Qual é o número de médicos de cada especialidade?
7. Para cada especialidade com menos de dois médicos listar o valor máximo e mínimo facturado por consulta, bem como o valor médio.
8. Quais são os médicos cujo valor facturado em 2002 é superior à média?
9. Actualizar o preço de referência das especialidades em 10
10. Remover os pacientes residentes em Braga.
11. Incremente em 15% o preço das consultas dos médicos com mais de 40 anos de idade.
12. Actualize o preço de refencia de Clínica Geral com o valor médio cobrado nas consultas de especialidade em 2003.
13. Remover os médicos que nunca consultaram.
14. Remover os códigos postais sem ligaçoes a médicos e/ou paciente
15. Remover as consultas executadas depois das18 horas pormédicos com mais de 60 anos.

O exemplo seguinte ilustra o ganho em volume de informação com o processo de normalização.
É dado o registo R_0 com a informação não normalizada:

R_0: médicos(m,nm,md,im,as,ce,de,pr,p,np,mp,ip,d,h,preço)

Será usado um exemplo considerando a cardinalidade de cada conjunto de entidade:

- 50 médicos
- 10000 pacientes
- 6 consultas diárias por médico

- 10 especialidades
- 1000 dias

Assim como o tamanho de cada atributo:

Col.	Tamanho (em bytes)
m	2
nm	40
mm	50
im	2
as	2
ce	2
de	40
pr	4
p	2
np	40
mp	50
ip	2
d	8
h	5
preço	4

Obtem-se:

Tamanho inicial da relação R_0 : $253bytes$

Cardinalidade de R_0 :$50 * 6 * 1000 = 300000$

Tamanho R_0: $300000 * 253 = 75900000b = 72.38Mb$

Existe um grupo repetido <p,np,mp,ip,d,h,preço> em R_0. Deste modo, na primeira forma normal R_0 é dividido em $R_{1,1}$ e $R_{1,2}$ tal que:

$R_{1,1}$ = medico(m,nm,mm,im,as,ce,de,pr)

$R_{1,2}$ = consultas(m,p,np,mp,ip,d,h,preço)

Obtem-se:

Tamanho inicial da relação $R_{1,1}$: $142bytes$

Cardinalidade de $R_{1,1}$: 50

Tamanho $R_{1,1}$: $50 * 142 = 7100b = 0.01Mb$

Tamanho inicial da relação $R_{1,2}$: $113bytes$

Cardinalidade de $R_{1,1}$: $50 * 6 * 1000 = 300000registos$

Tamanho $R_{1,1}$: $300000 * 113 = 33900000b = 32.33Mb$

Tamanho da base de dados na 1^a FN : $33907100b = 32.34Mb$

Na relação $R_{1,2}$ existe a dependência $p \longrightarrow np, mp, ip$ que é uma dependência funcional de parte da chave total <m,p> . Deste nodo da 2^a FN resultam as relações:

$R_{2,1} = R_{1,1}$

$R_{2,2}$ = consultas(m,p,d,h,preço)

$R_{2,3}$ = pacientes(p,np,mp,ip)

Tamanho inicial da relação $R_{2,1}$: $142bytes$

Cardinalidade de $R_{2,1}$: 50

Tamanho $R_{2,1}$: $50 * 142 = 7100b = 0.01Mb$

Tamanho inicial da relação $R_{2,2}$: $21bytes$

Cardinalidade de $R_{2,2}$: $50 * 6 * 1000 = 300000$

Tamanho $R_{2,2}$: $300000 * 21 = 6300000b = 6.01Mb$

Tamanho inicial da relação $R_{3,2}$: $94bytes$

Cardinalidade de $R_{3,2}$: 10000

Tamanho $R_{3,2}$: $10000 * 94 = 940000b = 0.90Mb$

Tamanho da base de dados na 2^a FN : $7247100b = 6.91Mb$

Na relação $R_{2,1} = medicos(m, nm, mm, im, as, ce, de, pr)$ existem as dependências $m \longrightarrow nm, mm, im, as, ce$ e $ce \longrightarrow de, pr$, dependências transitivas. Deste modo, após a aplicação da 3^a FN ontem-se:

$R_{3,1} = medicos(m, nm, mm, im, as, ce)$

$R_{3,2} = especialidades(ce, de, pr)$

$R_{3,3} = consultas(m, p, d, h$,preço)

$R_{4,3} = pacientes(p, np, mp, ip)$

Tamanho inicial da relação $R_{3,1}$: $98bytes$

Cardinalidade de $R_{3,1}$: 50

Tamanho $R_{3,1}$: $50 * 98 = 4900b = 0.005Mb$

Tamanho inicial da relação $R_{3,2}$: $46bytes$

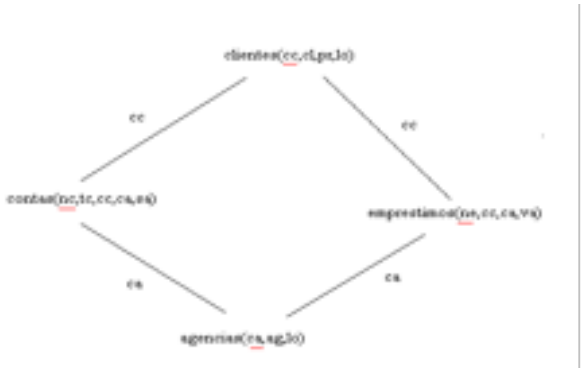

Figura 3.2: Esquema conceptual da base de dados

Cardinalidade de $R_{3,2}$: 10

Tamanho de $R_{3,2} : 10 * 46 = 460b = 0.0004Mb$

Tamanho inicial da relação $R_{3,3} : 21bytes$

Cardinalidade de $R_{3,3} : 50 * 6 * 1000 = 6300000$

Tamanho $R_{3,3} : 10000 * 21 = 6300000b = 6.01Mb$

Tamanho inicial da relação $R_{3,4} : 94bytes$

Cardinalidade de $R_{3,4}$: 10000

Tamanho $R_{3,4} : 10000 * 94 = 940000 = 0.90Mb$

Tamanho da base de dados na 3^a FN : $7245360b = 6.91Mb$

Os resultados da normalização encontram-se resumidos no quadro seguinte:

Fase	Espaço (em Bytes)	Espaço (em Mbytes)	Ganho Total	Ganho entre fases
Início	75 900 000	72.38		
1^aFN	33 907 100	32.34	55.33%	55.33%
2^aFN	7 247 100	6.91	90.45%	78.63%
3^aFN	7 245 360	6.91	90.45%	0.02%

Exercício Resolvido:
Dado o esquema da Figura 3.2, resolva:

1. Quais os clientes deste banco?

```
SELECT cc, cl FROM clientes

SELECT * FROM clientes
```

2. Quais os clientes que residem em Braga?

```
SELECT * FROM clientes
WHERE lo = 'BRAGA'

SELECT * FROM clientes
WHERE lo like '%BRAGA%'
```

3. Quais os clientes com conta(s) na agência ca = 123?

```
SELECT DISTINCT cc FROM contas
WHERE ca = '123'

SELECT DISTINCT c.*
FROM contas co,clientes c
WHERE c.cc = co.cc and co.ca = '123'

SELECT cc,cl FROM clientes c
WHERE EXISTS
(SELECT * FROM contas co
WHERE co.cc = c.cc and co.ca = '123')
```

4. Quais os clientes que residem na mesma localidade das agências?

```
SELECT DISTINCT clientes.* FROM clientes,agencias
WHERE clientes.lo = agencias.lo

SELECT DISTINCT c.* FROM clientes c,agencias a
WHERE c.lo = a.lo
```

5. Quais os clientes com empréstimos de valor superior a 500000 euros?

```
SELECT DISTINCT c.* FROM clientes c, emprestimos e
WHERE c.cc = e.cc AND e.va > 500000
```

6. Quais os nomes dos clientes com a mesma profissão que o cliente cc=1234?

```
SELECT DISTINCT c1.cl FROM clientes c1,clientes c2
WHERE c1.pr = c2.pr AND c2.cc = '1234'
```

7. Listar as contas da agência (ca=123) por ordem decrescente do valor do seu saldo?

```
SELECT DISTINCT co.* FROM contas co
WHERE co.ca = '123' ORDER BY sa DESC
```

8. Quantas contas existem em todas as agências do banco?

```
SELECT count(*) FROM contas
```

9. Quantos clientes possuem conta(s) na agência ca=123?

```
SELECT count(distinct cc) FROM contas
WHERE ca = '123'
```

10. Listar o número de contas existentes em cada agência?

```
SELECT ca, count(*) FROM contas GROUP BY  ca
```

11. Para cada agência com menos de 1000 contas, listar os valores máximos e mínimos dos saldos dessas contas, assim como o saldo médio.

```
SELECT ca, max(sa), min(sa), avg(sa) FROM contas
GROUP BY ca HAVING count(*) < 1000
```

12. Quais os clientes cuja profissão é desconhecida?

```
SELECT * FROM clientes WHERE pr is null
```

13. Quais os clientes da agência ca = 123?

```
SELECT DISTINCT c.*
FROM clientes c, contas co
WHERE co.ca = '123' AND co.cc = c.cc
UNION
SELECT DISTINCT c.*
FROM clientes c, emprestimos e
```

```
WHERE e.ca = '123' AND e.cc = c.cc

SELECT DISTINCT c.*
FROM clientes c, contas co, emprestimos e
WHERE (co.ca = '123' AND co.cc = c.cc)
OR (e.ca = '123' AND e.cc = c.cc)
```

14. Quais os clientes que são simultaneamente depositantes e devedores na agência ca = 123?

```
SELECT DISTINCT c.*
FROM clientes c, contas co
WHERE co.ca = '123' AND co.cc = c.cc
INTERSECT
SELECT DISTINCT c.*
FROM clientes c, emprestimos e
WHERE e.ca = '123' AND e.cc = c.cc

SELECT DISTINCT c.*
FROM clientes c, contas co, emprestimos e
WHERE (co.ca = '123' AND co.cc = c.cc)
AND (e.ca = '123' AND e.cc = c.cc)
```

15. Quais os clientes que são apenas depositantes na agência ca = 123?

```
SELECT DISTINCT c.*
FROM clientes c, contas co
WHERE co.ca = '123' AND co.cc = c.cc
EXCEPT
SELECT DISTINCT c.*
FROM clientes c, emprestimos e
WHERE e.ca = '123' AND e.cc = c.cc

SELECT DISTINCT c.*
FROM clientes c, contas co, emprestimos e
WHERE (co.ca = '123' AND co.cc = c.cc)
AND NOT (e.ca = '123' AND e.cc = c.cc)
```

16. Quais os clientes com, pelo menos, um empréstimo no banco?

```
SELECT c.* FROM clientes c
WHERE  EXISTS
(SELECT * FROM emprestimos e where e.cc = c.cc)
```

17. Quais as agências com depositantes residentes em Lisboa?

```
SELECT ag.* FROM agencias ag
WHERE  ag.ca IN
 (SELECT ca FROM contas where cc in
(select cc from clientes  where lo = 'Lisboa'))
```

18. Quais os clientes cujo saldo total das suas contas é superior a qualquer empréstimo contraído neste banco?

```
SELECT c.* FROM clientes c WHERE
(SELECT sum(sa) FROM contas co WHERE c.cc = co.cc)
>
(SELECT max(va) FROM emprestimos)
```

19. Quais os clientes que possuem contas em todas as agências do Porto?

```
SELECT c.* FROM clientes c WHERE
(SELECT count(DISTINCT co.ca) FROM contas co, agencias ag
WHERE co.cc = c.cc AND co.ca = ag.ca AND ag.lo = 'PORTO')
=
(SELECT count(distinct ca) FROM agencias WHERE lo = 'PORTO')

SELECT c.* FROM clientes c WHERE NOT EXISTS
(SELECT * FROM agencias ag
WHERE ag.lo = 'PORTO' AND NOT EXISTS
(SELECT * FROM contas co
WHERE co.cc = c.cc AND co.ca = ag.ca))
```

20. Para cada cliente apresentar o seu saldo total.

```
SELECT cc,sum(sa) FROM contas GROUP BY cc

SELECT c.cc,c.cl,
(SELECT sum(sa)  FROM contas co where co.cc = c.cc)
FROM clientes c
```

3.13 Funções

```
CREATE OR REPLACE  FUNCTION SONHO.PESQUISA
```

```
 (n1 number, n2 number, n3 number, n4 number, n5 char,
  n6 varchar2, n7 varchar2,
 n8 char,  n9 integer, n10 integer, apl char, n11 integer)
 return integer as r integer;  x integer;
begin
r := 0;
select max(ped) into r from pesquisa_temp;
if (r = 0)
then
   r := 1;
 else
 r := r + 1;
 end if;

insert into pesquisa_temp
 values(r,n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,apl);
commit;
loop
x := 0;
select ped into x from pesquisa_temp  where ped = r;
if (x != r)
then
   exit;
end if;
end loop;
return r;
end;
```

3.13.1 Execuçao da função em Microsoft Visual Basic 6.0

```
SQL = "Declare n1 Number(8) := " + trus(1) + "; n2 Number(8) := " + _
trus(2) + "; n3 Number(9):=" + trus(3) + _
"; n4 Number(8):=" + trus(4) + "; n5 char(3):=" + trus(5) + _
"; n6 varchar2(10):=" + trus(6) + "; n7 varchar2(100) := " + _
Trim(trus(7)) + "; n8 Char(1) := " + _
Trim(trus(8)) + "; n9 Integer:=" + trus(9) + "; n10 Integer:=" + _
trus(10) + "; n11 Integer:=" + _
trus(11) + "; reg Integer; "+ _
"Begin reg := sonho.pesquisa (n1, n2,n3,n4,n5,n6,n7,n8,n9,n10," + _
apl + ",n11); End;"

db.Execute Trim(SQL)
```

3.13.2 Procedimentos em Informix

```
CREATE PROCEDURE read_address (lastname CHAR(15)) - one argument
RETURNING CHAR(15), CHAR(15), CHAR(20),
CHAR(15),CHAR(2), CHAR(5); -- 6 items
DEFINE p_lname,p_fname,
p_city CHAR(15); --define each procedure variable
DEFINE p_add CHAR(20);
DEFINE p_state CHAR(2);
DEFINE p_zip CHAR(5);

SELECT fname, address1, city, state, zipcode
INTO p_fname, p_add, p_city, p_state, p_zip
FROM customer WHERE lname = lastname;
RETURN p_fname, lastname, p_add, p_city, p_state, p_zip; --6 items
END PROCEDURE
DOCUMENT 'This procedure takes the last name of a customer as',
--brief description
'its only argument. It returns the full name and address of the customer.'
WITH LISTING IN '/acctng/test/listfile'
-- compile-time warnings go here
; -- end of the procedure read_address

EXECUTE PROCEDURE read_address ('Putnum');

CREATE PROCEDURE address_list ()
DEFINE p_lname, p_fname, p_city CHAR(15);
DEFINE p_add CHAR(20);
DEFINE p_state CHAR(2);
DEFINE p_zip CHAR(5);
.
.
.
LET p_fname, p_lname,p_add, p_city, p_state, p_zip =
read_address ('Putnum');
.
..
-- use the returned data some way
END PROCEDURE;

CREATE PROCEDURE tryit()
.
.
.
CREATE TABLE gargantuan (col1  INT, col2 INT, col3 INT);
CREATE TABLE libby.tiny (col1  INT, col2 INT, col3 INT);
END PROCEDURE;
```

```
CREATE PROCEDURE scope()
DEFINE x,y,z INT;
LET x = 5; LET y = 10;
LET z = x + y; --z is 15
BEGIN
DEFINE x, q INT; DEFINE z CHAR(5);
LET x = 100;
LET q = x + y; -- q = 110
LET z = 'silly'; -- z receives a character value
END
LET y = x; -- y is now 5
LET x = z; -- z is now 15, not 'silly'
END PROCEDURE;

EXECUTE PROCEDURE read_address('Smith')
INTO p_fname, p_lname, p_add, p_city, p_state, p_zip;
CALL read_address('Smith')
RETURNING p_fname, p_lname, p_add, p_city, p_state, p_zip;

CREATE PROCEDURE call_test()
RETURNING CHAR(15), CHAR(15);
DEFINE fname, lname CHAR(15);
CALL read_name('Putnum') RETURNING fname, lname;
IF fname = 'Eileen' THEN RETURN 'Jessica', lname;
ELSE RETURN fname, lname;
END IF
END PROCEDURE;
```

3.14 Triggers

Triggers no Oracle são contruções em PL/SQL semelhantes a *procedures*. No entanto, enquanto uma *procedures* é executada explicitamente a partir de um bloco e através de uma invocação (CALL), um *trigger* é executado implicitamente sempre que um determinado evento ocorre (INSERT, DELETE, ou UPDATE). O *trigger* pode ser executado antes ou depois do evento (BEFORE ou AFTER).

3.14.1 Síntaxe básica de um Trigger

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
    {BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <table_name>
    [REFERENCING [NEW AS <new_row_name>] [OLD AS <old_row_name>]]
    [FOR EACH ROW [WHEN (<trigger_condition>)]]

    <trigger_body>
```

De notar:

- Triggers do tipo BEFORE e AFTER são apenas usados para tabelas (podem ser usados outro tipo de triggers para views, para implementar updates de views).
- Podem ser especificados até 3 eventos usando a palavra OR. UPDATE pode ser seguido, opcionalmente, pela palavra OF e uma lista de atributos em <table_name>. Desse modo, a claúsula OF define um evento que afecta apenas os atributos especificados. Por exemplo:

  ... INSERT ON R ...

  ... INSERT OR DELETE OR UPDATE ON R ...

  ... UPDATE OF A, B OR INSERT ON R ...
- Com a opção FOR EACH ROW, o trigger do is row-level; senão é do tipo statement-level.
- Para triggers do tipo row-level:
  - As variáveis do tipo NEW e OLD estão disponíveis para referenciar o valor do tuplo respectivamente depois ou antes da transacção. Note-se: No corpo do trigger, NEW e OLD devem ser precedidos por (":"), mas na claúsula WHEN, tal não se verifica.
  - A claúsula REFERENCING é usada para atribuir sinónimos à variáveis NEW e OLD.
  - Uma restrição pode ser especificada na claúsula WHEN. Esta condição não pode conter subqueries.
- <trigger_body> é um bloco em PL/SQL. Existem algumas restrições a ter em conta de forma a não entrar em ciclos infinitos.

Exemplos:

```
CREATE TABLE T4 (a INTEGER, b CHAR(10));

CREATE TABLE T5 (c CHAR(10), d INTEGER);

CREATE TRIGGER trig1
    AFTER INSERT ON T4
    REFERENCING NEW AS newRow
    FOR EACH ROW
    WHEN (newRow.a <= 10)
    BEGIN
        INSERT INTO T5 VALUES(:newRow.b, :newRow.a);
    END trig1;
.
run;
```

```
CREATE OR REPLACE TRIGGER "PCE"."TCCORRENTE" AFTER
INSERT ON "PCE"."CONSULTAS" REFERENCING OLD AS OLD NEW AS NEWROW FOR EACH ROW
declare r number;
begin

select P into r from ccorrente where
P = :newrow.P and A = to_char(:newrow.d,'yyyy')
and M = to_char(:newrow.d,'mm');
update ccorrente set v = v + :newrow.preco where P = :newrow.P and A = to_char(:newrow.d,'yy
and M = to_char(:newrow.d,'mm');

exception
when NO_DATA_FOUND
 then insert into ccorrente(p,A,m,v) values (:newrow.p,  to_char(:newrow.d,'yyyy'), to_char(

end tccorrente;
```

3.14.2 Erros na definição de um Trigger

Após a mensagem:

```
Warning: Trigger created with compilation errors.
```

Os erros podem ser consultados com:

```
show errors trigger <trigger_name>;
```

ou

```
SHO ERR
```

(abreviatura de SHOW ERRORS) para aceder aos esrros mais recentes.

3.14.3 Consulta de Triggers

```
select trigger_name from user_triggers;
```

Para mais detalhe sobre um trigger em particular:

```
select trigger_type, triggering_event, table_name,
referencing_names, trigger_body
from user_triggers
where trigger_name = '<trigger_name>';
```

3.14.4 Elimação de Triggers

```
drop trigger <trigger_name>;
```

3.14.5 Desactivação de Triggers

```
alter trigger <trigger_name> {disable / enable};
```

3.14.6 Suspensão de Triggers em situação de Erro

```
create table Person (age int);
CREATE TRIGGER PersonCheckAge
AFTER INSERT OR UPDATE OF age ON Person
FOR EACH ROW
BEGIN
    IF (:new.age < 0) THEN
        RAISE_APPLICATION_ERROR(-20000, 'no negative age allowed');
    END IF;
END;
.
RUN;
```

Ao executar:

```
insert into Person values (-3);
```

obtem-se a mensagem de erro:

```
ERROR at line 1:
ORA-20000: no negative age allowed
ORA-06512: at "MYNAME.PERSONCHECKAGE", line 3
ORA-04088: error during execution of trigger 'MYNAME.PERSONCHECKAGE'
```

3.14.7 Exemplo

```
CREATE TRIGGER upqty
UPDATE OF quantity ON items-- an UPDATE trigger event
BEFORE(EXECUTE PROCEDURE upd_items_p1)-- a BEFORE action

CREATE TRIGGER ins_qty
INSERT ON items -- an INSERT trigger event

CREATE PROCEDURE upd_items_p1()
DEFINE GLOBAL old_qty INT DEFAULT 0;
LET old_qty = (SELECT SUM(quantity) FROM items);
END PROCEDURE;

CREATE PROCEDURE upd_items_p2()
DEFINE GLOBAL old_qty INT DEFAULT 0;
```

```
DEFINE new_qty INT;
LET new_qty = (SELECT SUM(quantity) FROM items);
IF new_qty > old_qty * 1.50 THEN
RAISE EXCEPTION -746, 0, 'Not allowed - rule violation';
END IF
END PROCEDURE;

CREATE PROCEDURE upd_items_p1()
DEFINE GLOBAL old_qty INT DEFAULT 0;
LET old_qty = (SELECT SUM(quantity) FROM items);
END PROCEDURE;

CREATE TRIGGER up_items
UPDATE OF quantity ON items
BEFORE(EXECUTE PROCEDURE upd_items_p1())
AFTER(EXECUTE PROCEDURE upd_items_p2());

UPDATE items SET quantity = quantity * 2 WHERE manu_code = 'KAR‘

CREATE TABLE log_record
(item_num SMALLINT,
ord_num INTEGER,
username CHARACTER(8),
update_time DATETIME YEAR TO MINUTE,
old_qty SMALLINT,
new_qty SMALLINT);

FOR EACH ROW(INSERT INTO log_record
VALUES (pre_upd.item_num, pre_upd.order_num, USER, CURRENT,
pre_upd.quantity, post_upd.quantity));

CREATE TRIGGER up_price
UPDATE OF unit_price ON stock
REFERENCING OLD AS pre NEW AS post
FOR EACH ROW WHEN(post.unit_price > pre.unit_price * 2)
(INSERT INTO warn_tab VALUES(pre.stock_num, pre.order_num,
pre.unit_price, post.unit_price, CURRENT

CREATE TRIGGER upd_totpr
UPDATE OF quantity ON items
REFERENCING OLD AS pre_upd NEW AS post_upd
FOR EACH ROW(EXECUTE PROCEDURE calc_totpr(pre_upd.quantity,
post_upd.quantity, pre_upd.total_price) INTO total_price)
```

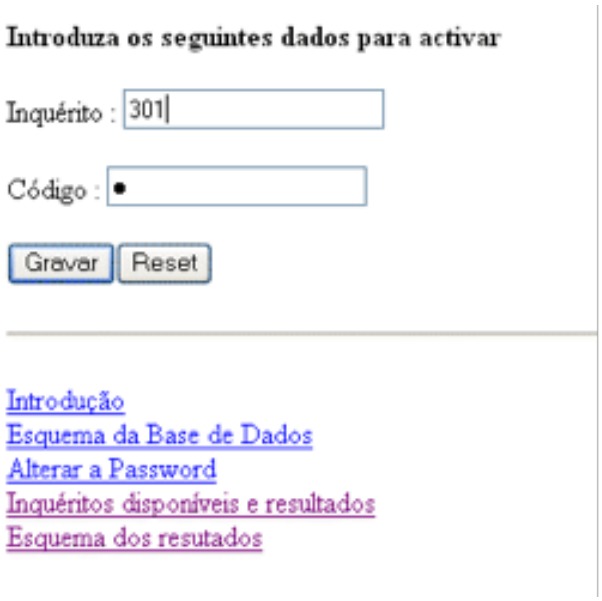

Figura 3.3: Execução do Programa 1

3.15 PHP e bases de dados

3.15.1 Programa 1

O seguinte programa em PHP permite construir a seguinte página apresentada na Figura 3.3:

```
<?php
require("local.inc");
?>
<html>

<head>
<title>Construção de Inquérito </title>
</head>

<body>

<?php
echo "<form method='POST' action='inq.php'>";

echo "</ol>
  <p><b> Introduza os seguintes dados para activar</b></p>
```

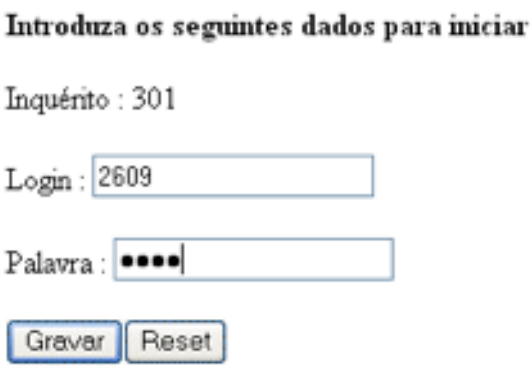

Figura 3.4: Execução do Programa 2

```
  <p>Inquérito : <input type='text' name='q' size ='20'></p>
  <p>Código    : <input type='password' name='palavra' size ='20'></p>
  <p><input type='submit' value='Gravar' name='b1'>
<input type='reset' value='Reset' name='b2'></p>";
?>
</form>
<hr><br>
<a href=/apsi/infos/trb.html>Introdução</a><br>
<a href=inquerito.zip>Esquema da Base de Dados</a><br>
<a href=mudapass.php>Alterar a Password</a><br>
<a href=lista.php>Inquéritos disponíveis e resultados</a><br>
<a href=resultados.zip>Esquema dos resutados</a>
</body>

</html>
```

3.15.2 Programa 2

O seguinte programa em PHP permite construir a seguinte página apresentada na Figura 3.4:

```
<?php
require("local.inc");
?>
<html>
<head>
<title></title>
```

```
</head>
<body>
<?php
$palavra = $_POST["palavra"];
$q = $_POST["q"];
if ($palavra == '9') {
echo "<form method='POST' action='inquerito.php'>";
echo "</ol>
 <p><b> Introduza os seguintes dados para iniciar</b></p>
   <p>Inquérito : ".$q."</p>
   <p>Login     : <input type='textbox' name='usern' size ='20'></p>
   <p>Palavra   : <input type='password' name='passw' size ='20'></p>
   <input type='hidden' name='palavra' value=".md5($palavra).">
   <input type='hidden' name='q' value=".$q.">
  <p><input type='submit' value='Gravar' name='b1'>
<input type='reset' value='Reset' name='b2'></p>
</form>";
}

else {
echo "Código Errado - Contacte o Docente";
}
?>
</body>
</html>
```

3.15.3 Programa 3

O seguinte programa em PHP permite construir a seguinte página apresentada na Figura 3.5:

```
<?php
require("local.inc");
?>
<html>
<head>
<title></title>
</head>
<body>
<?php
$usern = $_POST["usern"];
$passw = $_POST["passw"];
$palavra = $_POST["palavra"];
$q = $_POST["q"];
```

Figura 3.5: Execução do Programa 3

```
if ($palavra == md5("9")) {
$db = odbc_connect(BASE, DBUSER, DBPASSWORD)
$sql ="select * from alunos where t1 = '".$usern."'";
$rs = odbc_exec($db,$sql);
if (odbc_fetch_row($rs)) {
$PWD1 = odbc_result($rs,"t3");
$nome = odbc_result($rs,"t2");
}
else {
$PWD1 = "";
}
if (trim($passw) == trim($PWD1)) {
$qx = $q - 300;
echo "<form method='POST' action='gravainq2.php'>";
```

```
$sql = "select * from alunos_grupos where grupo = ".$qx;
odbc_exec($db,$sql);
while (odbc_fetch_row($rs)) {
$alu = odbc_result($rs,"aluno");
echo "<img src='http://labia.di.uminho.pt/apsi/photos/".$alu.".jpg' width=60>";
}
echo "<h2>Inquérito : ".$q."</h2><br>";
echo "<h2> ".$nome."</h2>";
echo "<img src='http://labia.di.uminho.pt/apsi/photos/".$usern.".jpg' width=100>";
$sql = "select * from questoes where quest = '".$q."' order by linha";
$rs = new recordset($db,$sql);
echo "<ol>";
while (odbc_fetch_row($rs)) {
$texto = odbc_result($rs,"texto");
$caixa = odbc_result($rs,"caixa");
$nomecaixa = trim(odbc_result($rs,"caixa"));
if (substr($nomecaixa,0,1) == "c") {
$j = substr($nomecaixa,1,3);
$c[$j] = $nomecaixa;
}
if ($caixa == 0) {
   echo "</ol>";
   echo "<p><b>".$texto."</b></p>";
   echo "<ol>";
}
elseif ($caixa == 1) {
echo "<li>".$texto." <input type='checkbox' name='c[".$j."]' value='ON'></li>"; }
elseif ($caixa == 3) {
echo  "<li>".$texto; }
  elseif ($caixa == 4) {
echo  $texto." <input type='checkbox' name='c[".$j."]' value='ON'></li>"; }
else {
   echo "</ol><p><b>".$texto."</b></p>
   <p><input type='textbox' name='".$nomecaixa."' size ='120'></p><ol>";
}
}
echo "</ol>
<p><b> Grave o seu inquérito</b></p>
<p><input type='hidden' name='palavra' value = '".$palavra."' size ='10'></p>
<p><input type='hidden' name='usern' value = '".$usern."' size ='10'></p>
<p><input type='hidden' name='passwd' value = '".$passw."' size ='10'></p>
<p><input type='hidden' name='q' value = '".$q."' size ='10'></p>
<p><input type='submit' value='Gravar' name='b1'>
<input type='reset' value='Reset' name='b2'></p>";
echo "</form>";
}
```

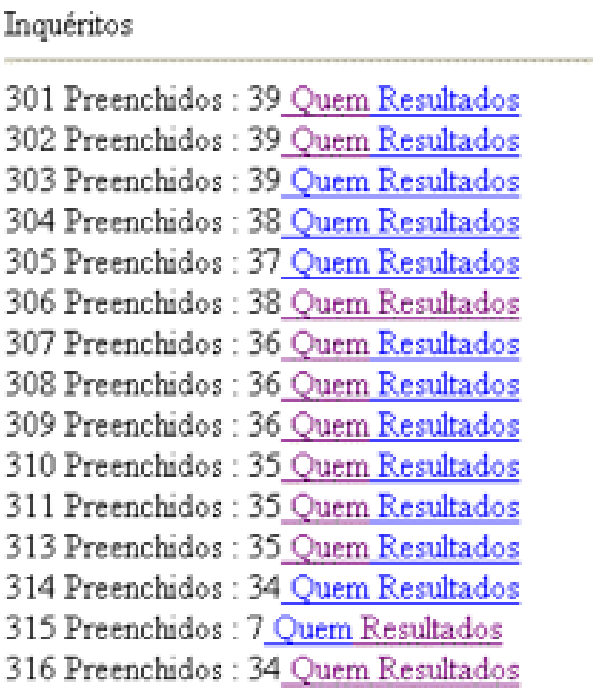

Figura 3.6: Execução do Programa 4

```
else { echo "Password Errada"; }

$db->disconnect();
}
else { echo "Parâmetros Ilegais - Contacte o Docente"; }
?>
</body>
</html>
```

3.15.4 Programa 4

O seguinte programa em PHP permite construir a seguinte página apresentada na Figura 3.6:

```
<?php
require("local.inc");
?>
```

```
<html>
<head>
<title>Lista de Inquéritos </title>
</head>
<body>
<?php
$db= odbc_connect(BASE, DBUSER, DBPASSWORD));
$sql ="select distinct quest as q from questoes where quest > 100";
echo "Inquéritos<br><hr>";
$rs = odbc_exec($db,$sql);
while (odbc_fetch_row($rs)) {
$q = odbc_exec($sql,"q");
$sql ="select count(*) as q1 from respostas where inquerito = ".$q;
$rs2= odbc_exec($db,$sql);
if odbc_fetch_row($rs) ) {
$q1 = odbc_result($rs,"q1");
}
echo $q." Preenchidos : ".(integer) $q1."<a href=quem.php?q=$q> Quem</a>
<a href=respostas.php?q=$q> Resultados </a><br>";
}
odbc_close($db);
?>
</body>
```

3.15.5 Programa 5

O seguinte programa em PHP permite construir a seguinte página apresentada na Figura 3.7:

```
<?php
require("local.inc");
?>
<html>
<head>
<title>Lista de Inquéritos </title>
</head>
<body>
<?php
$q = $_GET["q"];
$db = odbc_connect(BASE, DBUSER, DBPASSWORD);
$sql ="select aluno,t2,res1 from respostas,alunos
where respostas.aluno = alunos.t1 and
inquerito= ".$q." order by aluno";
echo "Alunos que preencheram o Inquérito $q<br><hr>";
```

Figura 3.7: Execução do Programa 5

```
$rs = odbc_exec($db,$sql);
$k = 0;
while (odbc_fetch_row($rs)) {
$q1 = $rs->row["aluno"];
$q2 = $rs->row["res1"];
$t2 = $rs->row["t2"];
$c1 = 0;
for ($i=0; $i<256;$i++) {
if (substr($q2,$i,1) == "1") {
$c1 = $c1 + 1;
}
}
echo $q1." ".$t2." "."($c1)<a href=quais.php?a=$q1> Quais </a><br>";
$k = $k + 1;
}
echo "Totais : $k";
odbc_close($db);
?>
```

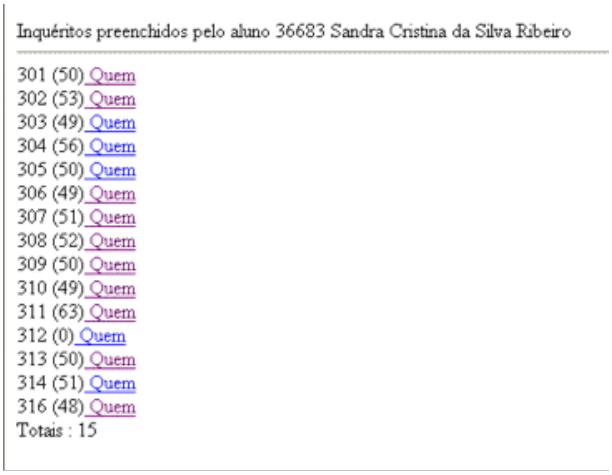

Figura 3.8: Execução do Programa 6

```
</body>
</html>
```

3.15.6 Programa 6

O seguinte programa em PHP permite construir a seguinte página apresentada na Figura 3.8:

```
<?php
require("local.inc");
require("/www/".DIR."/psw.inc");
require("/www/".DIR."/rs.inc");
?>
<html>
<head>
<title>Lista de Inquéritos </title>
</head>
<body>
<?php
$a = $_GET["a"];
$db=odbc_connect(BASE, DBUSER, DBPASSWORD) {
$sql ="select t2 from alunos where t1 = '".$a."'";
$rs = odbc_exec($db,$sql);
if (odbc_fetch_row($rs)) {
$t2 = odbc_result($rs,"t2");
}
$sql ="select inquerito,res1 from respostas where aluno= ".$a." order by inquerito";
```

```
echo "Inquéritos preenchidos pelo aluno $a $t2<hr>";
$rs = odbc_exec($db,$sql);
$k = 0;
while (odbc_fetch_row($rs)) {
$q1 = odbc_exec($rs,"inquerito");
$q2 = odbc_fetch_row($rs,"res1");
$c1 = 0;
for ($i=0; $i<256;$i++) {
if (substr($q2,$i,1) == "1") {
$c1 = $c1 + 1;
}
}
echo $q1." ($c1)<a href=quem.php?q=$q1> Quem </a><br>";
$k = $k + 1;
}
echo "Totais : $k";
odbc_close($db);
?>
</body>
</html>
```

3.15.7 Programa 7 - Gravação do inquérito

```
<?php
require("local.inc");
?>
<html>
<head>
<title>Gravação de Inquérito </title>
<font face="verdana">
<?php
$c = $_POST["c"];
$passwd = $_POST["passwd"];
$palavra = $_POST["palavra"];
$usern = $_POST["usern"];
$q = $_POST["q"];
$db = odbc_connect(BASE, DBUSER, DBPASSWORD);
$sql ="select * from alunos where t1 = '".$usern."'";
$rs = odbc_exec($db,$sql);
if (odbc_fech_row($rs) {
$PWD1 = odbc_rsult($rs,"t3");
}
else {
$PWD1 = "";
```

```
}
if ((trim($passwd) == trim($PWD1)) and ( $palavra == md5('9'))) {
$sql ="delete from respostas where inquerito = ".$q." and aluno = '".$usern."'";
$rs = new recordset($db);
odbc_do($db,$sql);
for ($k = 1; $k <= 6; $k++) {
$resulta[$k] = "";
for ($i = ($k-1)*250+1; $i <= 250*$k; $i++) {
if ($c[$i] == "ON") {
$resulta[$k] = $resulta[$k]."1";
}
else {
$resulta[$k] = $resulta[$k]."0";
}
}
}
$date = date("d-m-Y H:i");
$at1 = "res1,res2,res3,res4,res5,res6,d1,inquerito,aluno";
$resulta = "'". $resulta[1] . "','". $resulta[2] . "','" . $resulta[3] .
"','" . $resulta[4] ."','" .$resulta[5].   "','". $resulta[6]."'";
$vt1 = $resulta.",'".$date."',".$q.",'".$usern."'";
$sql ="insert into respostas(".$at1.") values (".$vt1.")";
odbc_do($db,$sql);

echo "Inquérito gravado com sucesso - Obrigado";

}
else
{
echo "Password errada";
}
odbc_close($db);
 ?>
</font>
</body>
</html>
```

3.15.8 Estrutura das tabelas

A estrutura das tabelas está apresentada na Figura 3.9:

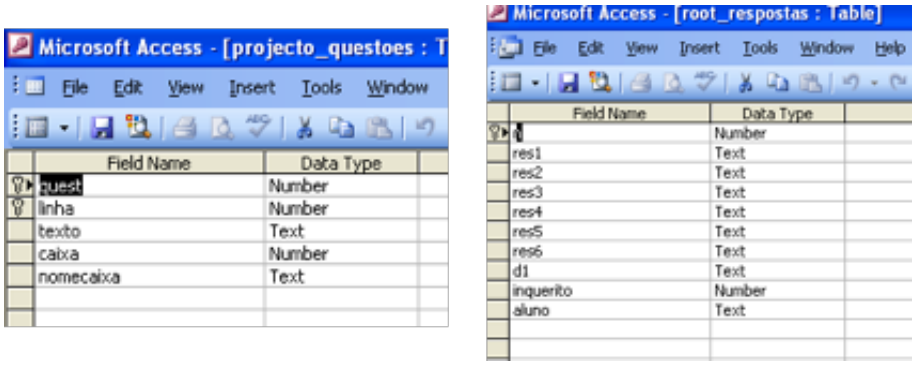

Figura 3.9: Estruturas das tabelas

3.16 .NET e bases de dados

3.16.1 Establecer ligação com o Oracle usando ODP.NET

C#

```
stringconstr= "UserId=Scott;Password=tiger;DataSource=orcl9i";
OracleConnectioncon= newOracleConnection(constr);
con.Open();
Console.WriteLine("Connectedto database!");
```

Visual basic

```
Dim constrAs String= "UserId=Scott;Password=tiger;DataSource=orcl9i"
Dim con As OracleConnection= NewOracleConnection(constr)
con.Open()
Console.WriteLine("Connectedto database!")
```

3.16.2 Criar e executar comandos SQL sem retorno em .NET

C#

```
Stringblock= "INSERT INTO testblob(id, name) VALUES (100, 'José')";
OracleCommandcmd= newOracleCommand();
cmd.CommandText= block;
cmd.Connection= con;
cmd.ExecuteNonQuery();
```

Visual basic

```
Dim sql as string
Dim my Command as OracleClient.Oracle
```

```
Commandsql= "INSERT INTO testblob(id, name) VALUES (100, ‘José’)"
 myCommand= NewOracleClient.OracleCommand(sql, oracledb)
myCommand.ExecuteNonQuery()
```

3.16.3 Criar e executar comandos SQL com retorno em .NET

Visual basic

```
Dim sql as string
dim myCommand as OracleClient.OracleCommand
dim myReader As OracleClient.OracleDataReader
sql= "selectid, nome fromtabela "
myCommand= NewOracleClient.OracleCommand(sql, oracledb)
myReader= myCommand.ExecuteReader()
If myReader.Read() Then
If myReader.IsDBNull(0) Then
A = myReader. GetValue(0)
Endif
If myReader.IsDBNull(1) Then
B = myReader. GetString(1)
End if
End if
```

3.16.4 Exemplo de um SELECT

Visual basic

```
oracledb = New OracleClient.OracleConnection
oracledb.ConnectionString = "User ID=sil;Data Source=MARTE;password=x"
oracledb.Open()

sql = "select id_termos,de_termos from termos_xml order by id_termos"

myCommand = New OracleClient.OracleCommand(sql, oracledb)
myReader = myCommand.ExecuteReader
Dim codigo, descricao As String
While myReader.Read
codigo = myReader.GetString(0)
If myReader.IsDBNull(1) Then
descricao = ""
Else
descricao = myReader.GetString(1)
End If
```

```
...........
End While
myReader.Close()
oracledb.Close()
```

3.16.5 Exemplo de um UPDATE

Visual basic

```
oracledb = New OracleClient.OracleConnection
oracledb.ConnectionString = "User ID=sil;Data Source=MARTE;password=x"
oracledb.Open()

sql = "update tabigif set digif='" & desc.Text & "' where cigif='" & TC & "'"

myCommand = New OracleClient.OracleCommand(sql, oracledb)
myCommand.ExecuteNonQuery()
oracledb.Close()
```

3.16.6 Exemplo de um LOB temporário

Visual basic

```
oracledb = New OracleClient.OracleConnection
oracledb.ConnectionString = "User ID=pce;Data Source=MARTE;password=x"
oracledb.Open()
Dim AE As New System.Text.UnicodeEncoding

Dim ByteArray As Byte() = AE.GetBytes(pxml.DocumentElement.OuterXml)

Dim pdata As New OracleClient.OracleParameter
Dim tx As OracleClient.OracleTransaction
Dim sql As String

tx = oracledb.BeginTransaction
myCommand = New OracleClient.OracleCommand
myCommand = oracledb.CreateCommand
myCommand.Transaction = tx
sql = "declare xx Clob; "+ _
"begin dbms_lob.createtemporary(xx, false, 0); :tempClob := xx; end;"
```

```
pdata.Direction = ParameterDirection.Output
pdata.OracleType = OracleClient.OracleType.Clob
pdata.ParameterName = "tempClob"

myCommand.Parameters.Add(pdata)
myCommand.CommandText = sql
myCommand.ExecuteNonQuery()

Dim tempLob As OracleClient.OracleLob
tempLob = myCommand.Parameters(0).Value
            tempLob.BeginBatch(OracleClient.OracleLobOpenMode.ReadWrite)

tempLob.Write(ByteArray, 0, ByteArray.Length)

tempLob.EndBatch()

myCommand.Parameters.Clear()
myCommand.CommandText = "pce.updateXMLtoTermos"

myCommand.CommandType = CommandType.StoredProcedure

Dim oraparameter As New OracleClient.OracleParameter
("strXML", OracleClient.OracleType.Clob)
oraparameter.Direction = ParameterDirection.Input
oraparameter.SourceVersion = DataRowVersion.Original
oraparameter.SourceColumn = "MSGDATA"
oraparameter.Value = tempLob
myCommand.Parameters.Add(oraparameter)
myCommand.Parameters.Add("codigo_", OracleClient.OracleType.VarChar, 20,
ParameterDirection.Input).Value = TC
myCommand.ExecuteNonQuery()
tx.Commit()
oracledb.Close()
```

3.17 SQLPlus

3.17.1 Fim de sessão

```
quit
```

3.17.2 Alteração de password

O comando passw altera a password do utilizador. Não use os caracteres @ ou / .

```
passw
```

3.17.3 Terminação de comandos

Os comandos são guardados num buffer até aparecer o símbolo ;. Podem ocupar várias linhas. Em caso de erro faça:

```
edit
```

O editor de texto associado ao SQLPlus pode ser alterado:

```
define _editor = nomeeditor
```

3.17.4 Confirmação de dados

```
commit;
```

Em caso de erro para não confirmar a operação:

```
rollback;
```

3.17.5 Logging

```
spool nomeficheiro
```

3.17.6 Para terminar o logging

```
spool off
```

3.17.7 Caracteres especiais

```
set escape \

select 'It''s mine\; I like it' from dual;
```

Para terminar:

```
set escape off
```

3.17.8 Carregamento de comandos a partir de um ficheiro

```
@nomeficheiro.sql
```

3.17.9 Execução de comandos no login

Os comandos devem constar no ficheiro *login.sql*.

3.17.10 Formato da data

O Oracle oferece muitas opções para a data. O comando:

```
ALTER SESSION
```

permite alterar o formato da data. Use:

```
SET NLS_DATE_FORMAT
```

seguido da string de formato da data. O formato por omissão é DD-MON-YY.

String	Significado
YYYY	Ano
MM	Mês
DD	Dia
HH	Hora
HH24	Hora (24 horas)
MI	Minuto
SS	Segundo

```
ALTER SESSION SET NLS_DATE_FORMAT='YYYY/MM/DD HH24:MI'
```

3.17.11 Formato de Saída

```
SET NEWPAGE 0
SET SPACE 0
SET LINESIZE 80
SET PAGESIZE 0
SET ECHO OFF
SET FEEDBACK OFF
SET HEADING OFF
```

3.17.12 Ajuda adicional

```
help comando
```

Por exemplo:

```
help index
```

Exemplo:

```
SQL> spool answer.txt
SQL> DEFINE _EDITOR= pico
SQL> spool off
SQL> commit;

Commit complete.

SQL> select username from junk;
select username from junk
                     *
ERROR at line 1:
ORA-00942: table or view does not exist
SQL> c /junk/user_users
  1* select username from user_users
SQL> /

USERNAME
------------------------------
STUDENTNAME

SQL> quit
Disconnected from Personal Oracle9i Release 9.0.1.1.1 - Production
With the Partitioning option
JServer Release 9.0.1.1.1 - Production
```

3.18 Segurança em bases de dados

As bases de dados são utilizadas para armazenar diversos tipos de informações, inclusive dados privados, confidenciais e valiosos.

São meios de proteger as informações armazenadas numa base de dados:

- Criptografia
- Senhas
- Cópias de segurança (no oracle ver comandos *exp* e *imp*).

3.19 Transacções

Uma transacção é um conjunto de procedimentos executado numa base de dados, que para o utilizador é visto como uma única operação. A integridade de uma transacção depende de 4 propriedades, conhecidas como ACID:

Atomicidade. Todas as acções que compõem a transacção devem ser concluídas com sucesso, para que a mesma seja efectivada (commit). Qualquer acção que falhe obriga a transacção a ser desfeita (rollback).

Consistência. Nenhuma parte de uma transacção pode ser parcial. A transacção deve ser implementada na íntegra.

Isolamento. Cada transação é independente das restantes e todas as operações são parte de uma transacção. Nenhuma outra transacção, no mesmo sistema, pode interferir no funcionamento duma transacção. Outras transacções não podem visualizar os resultados parciais das operações de uma transacção em andamento; i.e., os resultados da transacção só ficam disponíveis aos restantes utilizadores e processos após a efectivação (commit).

Durabilidade. Os resultados de uma transacção são permanentes e apenas podem ser desfeitos por uma transacção subsequente.

3.20 Bases de dados dependentes da dimensão temporal

3.20.1 Abordagem baseada em informação negativa

Tabela p^+

T	A_1	...	A_n
t_i	x_1	...	x_n
t_x	x_1	...	x_n

Tabela p^-

T	A_1	...	A_n
t_j	x_1	...	x_n
t_y	x_1	...	x_n

- $A_1, \ldots, A_n \in p$ no ponto t_k sse

  $\exists t_1 | t_1, A_1, \ldots, A_n \in p^+ \wedge t_1 < t_k$ e
  $\neg\exists t_2 | t_2, A_1, \ldots, A_n \in p^- \wedge t_2 < t_k \wedge t_2 > t_1$

- $A_1, \ldots, A_n \notin p$ no ponto t_k sse

  $\exists t_1 | t_1, A_1, \ldots, A_n \in p^- \wedge t_1 < t_k$ e
  $\neg\exists t_2 | t_2, A_1, \ldots, A_n \in p^+ \wedge t_2 < t_k \wedge t_2 > t_1$
  ou $\neg\exists t_2 | t_2, A_1, \ldots, A_n \in p^+ \wedge t_2 < t_k$

3.20.2 Abordagem baseada em intervalos

Tabela p^*

From	To	A_1	...	A_n
t_i	t_j	x_1	...	x_n
t_x	t_y	x_1	...	x_n

- $A_1, \ldots, A_n \in p$ no ponto t_k sse

  $\exists t_1, t_2 | t_1, t_2, A_1, \ldots, A_n \in p^* \wedge t_1 \leq t_k \leq t_2$

- $A_1, \ldots, A_n \notin p$ no ponto t_k sse

  $\neg \exists t_1, t_2 | t_1, t_2, A_1, \ldots, A_n \in p^* \wedge t_1 \leq t_k \leq t_2$

3.21 Bases de dados distribuídas

Em termos de tratamento informático, qualquer sistema, quer este seja suportado por um ambiente de bases de dados ou de gestão de ficheiros, envolve, fundamentalmente, dois elementos:

- Dados;
- Processamento.

No contexto específico das bases de dados, e em termos de distribuição destes dois elementos, é possível definir algumas configurações distintas, desde as configurações em que a sua distribuição é nula, ou seja, os dois elementos encontram-se centralizados numa só máquina, até às configurações em que estes dois elementos se encontram distribuídos por várias máquinas, ligadas por meios de comunicação.

No sistema centralizado (Figura 3.10), os dados e o processamento estão centralizados. No sistema cliente-servidor (Figura 3.11), os dados estão centralizados enquanto o processamento é distribuído pelo diferentes clientes. O servidor é responsável apenas pela execução de interrogações em linguagem SQL.

A arquitectura 3-camadas (Figura 3.12) suporta o processamento pela camada dos clientes e pela camada aplicacional. Os comandos em SQL são também executados na camada organizacional. Não existe qualquer tipo de ligação física entre os clientes e a base de dados, funcionando o nível aplicacional como uma camada de segurança e de verificação de permissões. O nível aplicacional pode ser implementado usando tecnologia Web (via Intranet) fazendo com que não seja necessário instalar qualquer software de apoio na camada de clientes onde um simples "browser" é suficiente para executar as operações.

No esquema de uma base de dados distribuída (Figura 3.13), o processamento e os dados são distribuídos. Em cada nodo há uma base de dados capaz de sincronizar-se, fragmentar-se e distribuir interrogações de forma optimizada e transparente por todos os nodos.

Um sistema de bases de dados relacionais contêm um ou mais objectos chamados tabelas. Os dados ou informação são armazenados nessas tabelas. As tabelas são apenas identificadas pelo nome e contêm colunas e linhas. As colunas contêm o nome da coluna, o tipo de dado, e qualquer outro atributo para a coluna. As linhas contêm os registos ou dados para as colunas.

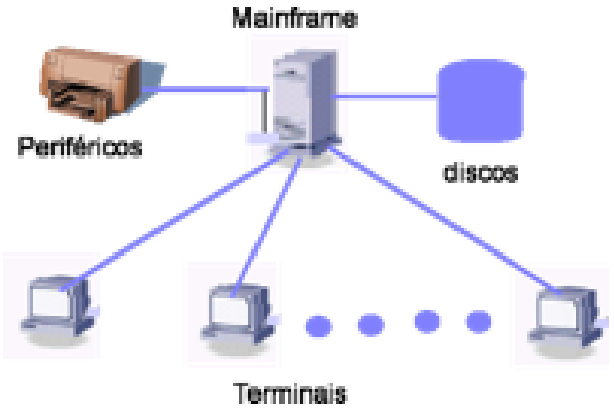

Figura 3.10: Sistema centralizado

3.21.1 Sistemas gestores de bases de dados distribuídas

Por definição, uma base de dados distribuída é um sistema de bases de dados cujos dados se encontram fisicamente dispersos por várias máquinas, ligadas por meios de comunicação, mas integrados logicamente, de uma forma mais abrangente, define-se uma base de dados distribuída como sendo um conjunto de centros de computação, ligados por redes de computadores em que, por um lado, cada centro constitui um sistema de bases de dados por si; por outro lado, os vários centros concordaram em cooperar, de tal forma que um utilizador de um dos centros pode utilizar dados armazenados nos outros centros como se esses dados lhe fossem locais.

Existem dois tipos de SGBDs distribuídos:

Homogéneos. Neste caso, todos os nodos usam o mesmo SGBD, fazendo lembrar um único sistema de bases de dados mas em que, ao contrário de todos os dados estarem armazenados num único repositório, os dados estão armazenados por vários repositórios ligados por meios de comunicação.

Heterogéneos. Este tipo de sistema de bases de dados distribuídas caracteriza-se pela existência de SGBDs diferentes nos vários nodos. As diferenças entre os SGBDs presentes podem colocar-se a vários níveis, desde SGBDs diferentes baseados no mesmo modelo de dados (por exemplo, Oracle, DB2, Informix, Sybase, etc., consistindo em implementações distintas do modelo relaciona!), até SGBDs baseados em modelos de dados diferentes (relacionais, rede, QO, etc.).

Os SGBDs distribuídos podem também ser caracterizados de acordo com o tipo de infra-estruturas de comunicação que utilizam:

LAN (Local Area Network). Os computadores localizam-se relativamente próximo uns dos outros (menos de 1km), interligados por cablagem directa.

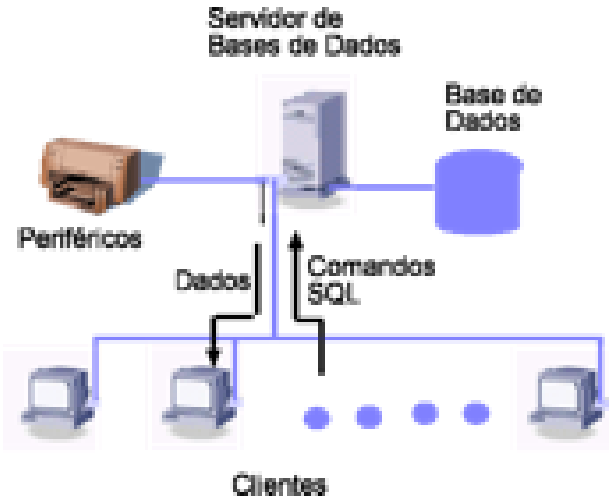

Figura 3.11: Sistema cliente-servidor

Actualmente, as taxas de transmissão mais comuns situam-se entre os 10 Mbps (Ethernet) e os 100 Mbps (Fast Ethernet), sendo a transmissão do tipo broadcasting.

WAN (Wide Area Network). Neste caso, os computadores encontram-se geograficamente distantes, interligados por meios de comunicação próprios ou alugados a terceiros. As taxas de transmissão podem rondar os 50 Kbps, sendo a transmissão do tipo "ponto-a-ponto".

Se os dados estão distribuídos por pontos geograficamente distintos, sempre que houver necessidade de lhes aceder, estes terão que ser transferidos desde os seus locais de origem, até ao local que os solicitou. Estas "viagens" significam custos, quer em termos de exploração dos meios de comunicação, quer em termos de tempo consumido.

Uma alternativa, que pode reduzir significativamente estes custos, consiste em replicar os dados mais frequentemente solicitados pelos nodos que mais os solicitam. Desta forma, aumenta-se o processamento local a cada nodo pois há maior probabilidade de os dados necessários se encontrarem armazenados localmente.

Naturalmente, a necessidade de manter as réplicas permanentemente sincronizadas traz algumas dificuldades. De facto, para satisfazer este requisito é necessário garantir que cada actualização só entra em vigor quando todas as suas réplicas forem também actualizadas com sucesso. Numa situação dessas, a finalização de uma transacção distribuída envolvendo actualização de dados tem que utilizar um protocolo específico que garanta que todas as réplicas ficam sincronizadas, caso isso não seja possível a transacção deve falhar em todas elas, desfazendo os seus efeitos. Infelizmente, num sistema distribuído existem

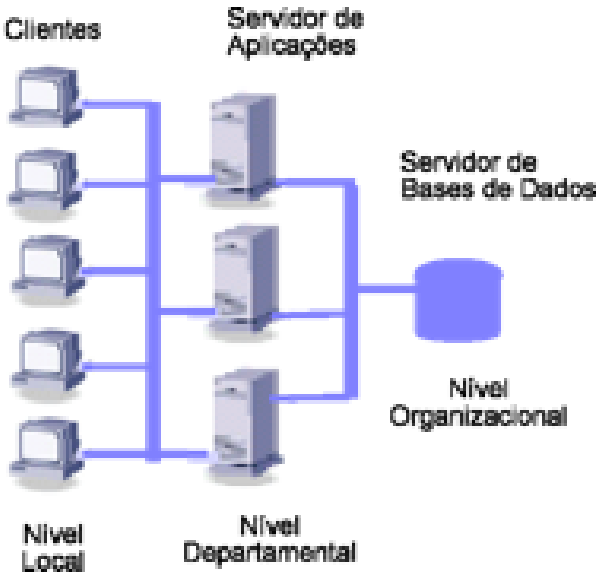

Figura 3.12: Arquitectura 3-camadas

vários factores que podem colidi com a necessidade de manter todas as réplicas sincronizadas, desde falhas nas comunicações, até falhas dos próprios nodos individuais.

Existem dois tipos básicos de replicação. Um, mais simples, designado primary site replication, em que um dos nodos (primary site) detém a posse dos dados, sendo o único que os pode actualizar. Este, de forma sincronizada ou não, propaga essas actualizações para os nodos onde estão as suas réplicas - denominadas snapshots. Numa versão mais sofisticada, designada dynamic ownership, a autorização para actualizar dados replicados vai-se movimentando entre os nodos. Em todo o caso, em cada momento apenas um dos nodos possui a autorização. Um outro tipo de replicação mais complexo, designado replica ção simétrica, permite que qualquer réplica seja actualizada, a qualquer momento, eventualmente em simultâneo, propagando-se o efeito dessas actualizações, de forma sincronizada ou não, para as restantes réplicas.

A fragmentação de dados pode assumir três formas:

Fragmentação vertical (Figura 3.14). A tabela lógica é dividida em tabelas cujos esquemas são subconjuntos do esquema da tabela original (na sua versão mais simples, a chave da tabela lógica propaga-se a todos os fragmentos verticais). Por analogia com as operações da álgebra relacional, a fragmentação vertical pode ser vista como a execução de projecções sobre a tabela lógica, armazenando as tabelas resultantes (tabelas físicas) em nodos diferentes.

Fragmentação horizontal (Figura 3.15). Neste caso, a tabela lógica e

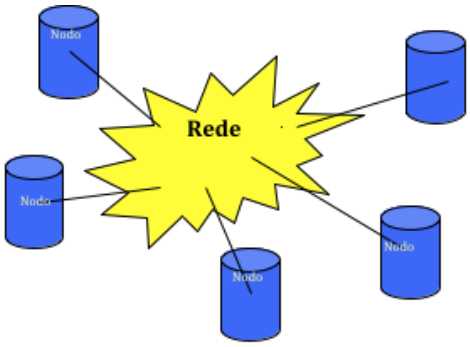

Figura 3.13: Base de dados distribuída

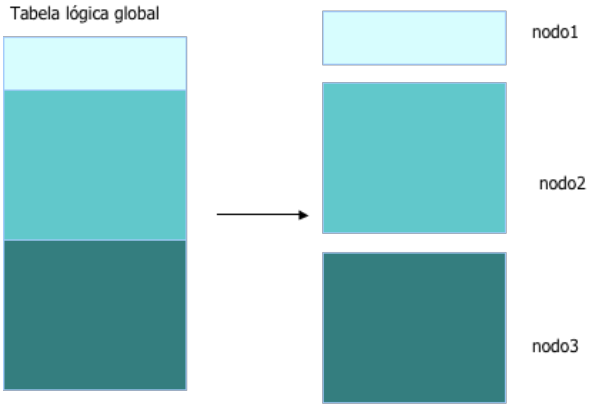

Figura 3.14: Fragmentação vertical

dividida em varias partes, cada uma delas com o mesmo esquema da tabela original. Continuando a analogia com as operações da álgebra relacional, o exemplo da Figura pode ser obtido executando três operações de selecção sobre a tabela lógica e armazenando as tabelas resultantes (tabelas físicas) em nodos diferentes.

Fragmentação mista. Sobre uma mesma tabela lógica são definidos fragmentos verticais e horizontais..

Idealmente, um sistema de bases de dados distribuídas deveria apresentar-se ao nível aplicacional como um só sistema correndo sobre uma "grande máquina". Por outras palavras, o nível aplicacional deveria aceder a dados localizados algures na rede com a mesma facilidade com que acederia se esses dados residissem numa mesma máquina. Ao suportar esta característica, a tecnologia de bases de dados distribuídas combina dois conceitos divergentes:

- Distribuição física dos dados.

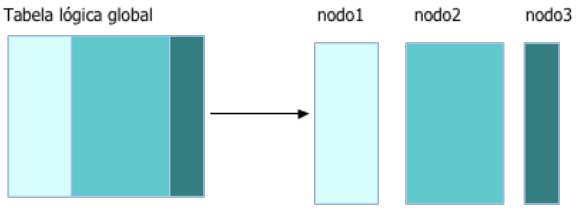

Figura 3.15: Fragmentação horizontal

- Integração lógica dos mesmos.

O desenvolvimento de bases de dados distribuídas tem as seguintes características:

Dificuldades de desenvolvimento. Já que a localização dos dados e o código necessário para a navegação na rede teriam que ser especificados nos acessos aos dados.

Dificuldades de manutenção. Se, por qualquer razão, a localização dos dados fosse alterada para outro(s) nodo(s) na rede, seria necessário actualizar todas as aplicações por forma a referenciar as novas localizações.

Disponibilidade permanente. Alterações na arquitectura da rede, por adição de novos nodos ou remoção de nodos existentes, assim como falhas de nodos, não devem impedir os restantes nodos de continuarem a sua operação, se não total, pelo menos parcialmente.

Fragmentação transparente. Uma mesma tabela pode estar fragmentada por vários nodos. Essa fragmentação deve ser transparente aos utilizadores que deverão continuar a "ver" uma só tabela lógica.

Duplicação transparente. Alguns dados, por razões de prevenção e/ou desempenho, podem estar duplicados em vários nodos. Da mesma forma, essa duplicação deve ser transparente aos utilizadores. O SGBD é o único responsável por manter a sua coerência.

Processamento de questões distribuído. O processamento de uma questão pode necessitar da cooperação de vários nodos. Todo esse processo é coordenado pelo sistema sem intervenção dos utilizadores.

Independência do hardware, sistema operativo e tecnologias de rede. Não deverá ser obrigatório que o hardware, o sistema operativo ou a tecnologia de rede utilizadas sejam iguais em todos os nodos. O mesmo deverá acontecer com os SGBDs utilizados. Ainda que SGBDs diferentes cooperem seria conveniente que o utilizador pudesse aceder aos dados presentes noutras máquinas utilizando o mesmo interface que para os dados presentes localmente. O utilizador deveria estar abstraído destes detalhes.

Relativamente à concepção de bases de dados distribuídas existem, basicamente, duas abordagens:

Top-down. Correspondente à divisão de uma base de dados pré-existente ou, mais genericamente, de um esquema de base de dados pré-definido em várias partes a armazenar em diferentes nodos. Normalmente, o resultado deste processo será um ambiente distribuído homogéneo de bases de dados.

Bottom-up. Correspondente, não à desagregação, mas sim à integração de várias bases de dados pré-existentes numa base de dados global, distribuída por várias máquinas. Dada a mais que provável heterogeneidade das várias bases de dados em presença, esta será a abordagem que mais dificuldades oferece.

O processamento distribuído de uma questão consiste na decomposição dessa questão em sub questões, de acordo com a distribuição dos dados pretendidos pelos vários nodos. Posteriormente, nos nodos respectivos, a resolução de cada uma destas sub questões é optimizada localmente, de forma idêntica aos sistemas centralizados. Resumindo, a execução de uma questão global consiste, normalmente, num conjunto de execuções de sub questões locais, provavelmente em paralelo, cujos resultados intermédios são movimentados entre os nodos participantes e finalmente reunidos para constituir a resposta final.

3.22 XML e Bases de dados

Um documento XML é uma base de dados no sentido básico do termo (i.e., é um conjunto de dados e contém dados com tipos diferentes):

Vantagem: apresenta um formato aceite universalmente que facilita a troca de informação.

Desvantagem: o acesso aos dados é lento devido a questões de "parsing" e de conversão de texto.

O XML oferece muitas das características encontradas em sistemas de bases de dados:

- armazenamento (ficheiros XML);
- esquemas (DTDs, XSL, etc...);
- linguagens de interrogação (XQuery, XPath, XQL, XML-QL, QUILT, etc...);
- interfaces de programação (SAX, DOM, JDOM).

Falhas:

- armazenamento eficiente;

- índices;
- segurança;
- transacções;
- integridade;
- multi-utilização;
- triggers;
- multi-interrogação / junção;
- etc...

Exemplo:

```
<encomenda numero="12345">
<cliente nclie="123">
<nomecli> Empresa lda</nomecli>
<localidade> Braga </localidade>
...
</cliente>
<dataencom>20-12-2003</dataencom>
<artigo codarti = "XYZ">
<designacao>Monitor X</designacao>
<quantidade>1</quantidade>
<preco>351</preco>
</artigo>
<artigo codarti = "XYZ2">
</artigo>
</encomenda>
```

encomenda

numero	nclie	dataencom
12345	123	20-12-2003
...	...	...

clientes

nclie	nomecli	localidade
123	Empresa lda	Braga
...	...	...

artiencom

codarti	designacao	quantidade	preço
XYZ	Monitor X	1	351
XYZ2	...	...	...
...	...	...	...

```
<database databasename=...>
<table tablename = ...>
<row>
<column1> ... </column1>
<column2>...</column2>
...
</row>
<row>
...
</row>
</table>
<table tablename = ...>
...
</table>
...
</database>
```

3.23 XML e Oracle

O Oracle disponibiliza o tipo de dados sys.XMLTYPE para gravar conteúdos em XML em tabelas. Por exemplo:

```
create table tabxml (
idser varchar2(10),
grupo varchar2(10),
sintaxe varchar2(20),
documento sys.XMLTYPE,
primary key(idser,grupo,sintaxe));
```

ou

```
create table PURCHASEORDER (PODOCUMENT sys.XMLTYPE);
```

No primeiro caso além do atributo do tipo XMLTYPE existem outros atributos, no segundo a tabela apenas tem um atributo. Para inserir um texto em XML deve-se utilizar a função sys.XMLTYPE.createXML().

Por exemplo:

```
insert into PURCHASEORDER (PODOCUMENT) values (
   sys.XMLTYPE.createXML('
  <PurchaseOrder>
  <Reference>BLAKE-20010625140342980PDT</Reference>
   <Actions>
   <Action>
  <User>KING</User>
     <Date/>
  </Action>
```

```
 </Actions>
 <Reject/>
  <Requester>David E. Blake</Requester>
  <User>BLAKE</User>
 <CostCenter>S30</CostCenter>
  <ShippingInstructions>
  <name>David E. Blake</name>
  <address>400 Oracle Parkway Redwood Shores, CA, 94065 USA
</address>
 <telephone>650 999 9999</telephone>
 </ShippingInstructions>
   <SpecialInstructions>Air Mail</SpecialInstructions>
     <LineItems>
      <LineItem ItemNumber="1">
        <Description>The Birth of a Nation</Description>
         <Part Id="EE888" UnitPrice="65.39" Quantity="31"/>
        </LineItem>
       </LineItems>
    </PurchaseOrder>
 '));
```

Ou

```
insert into tabxml values('RXU','UN','amostra',
sys.XMLTYPE.createXML(
'<amostra>
<numero>123</numero>
<episodio>445</episodio>
<modulo>CON</modulo>
<facturacao>
<resultado>
<codigo>abc</codigo>
<preco>100.25</preco>
</resultado>
<resultado>
<codigo>abd</codigo>
<preco>200.5</preco>
</resultado>
</facturacao>
</amostra>'));
```

A interrogação é feita usando o comando SELECT. Para obter todo o texto em XML, na utiliza-se, por exemplo [1]:

```
select p.podocument.getClobVal() from purchaseorder p;
```

[1] No SQLPlus, tendo em conta o tamanho da string rsultante use o comando SET LONG 10000.

O resultado, considerando a informação inserida anteriormente é:

```
P.PODOCUMENT.GETCLOBVAL()
---------------------------------
<PurchaseOrder>
   <Reference>BLAKE-20010625140342 98PDT</Reference>
   <Actions>
      <Action>
         <User>KING</User>
         <Date/>
      </Action>
   </Actions>
   <Reject/>
   <Requester>David E. Blake</Requester>
   <User>BLAKE</User>
   <CostCenter>S30</CostCenter>
   <ShippingInstructions>
      <name>David E. Blake</name>
      <address>400 Oracle Parkway Redwood Shores, CA, 94065 USA</address>
      <telephone>650 999 9999</telephone>
   </ShippingInstructions>
   <SpecialInstructions>Air Mail</SpecialInstructions>
   <LineItems>
      <LineItem ItemNumber="1">
         <Description>The Birth of a Nation</Description>
         <Part Id="EE888" UnitPrice="65.39" Quantity="31"/>
      </LineItem>
   </LineItems>
```

Outro exemplo é:

```
select t.documento.getClobVal() from tabxml t;
```

A função EXTRACT permite instanciar parte do texto XML.

```
select P.PODOCUMENT.extract('/PurchaseOrder/ShippingInstructions').getClobVal()
   from PURCHASEORDER P
```

where

```
P.PODOCUMENT.extract('/PurchaseOrder/Reference/text()').getStringVal() =
   'BLAKE-20010625140342 98PDT')
```

O resultado é:

```
P.PODOCUMENT.EXTRACT('/PURCHASEORDER/SHIPPINGINSTRUCTIONS').GETCLOBVAL()
----------------------------------------
```

```
<ShippingInstructions>
  <name>David E. Blake</name>
  <address>400 Oracle Parkway Redwood Shores, CA, 94065 USA</address>
  <telephone>650 999 9999</telephone>
</ShippingInstructions>
```

Outro exemplo é:

```
select P.DOCUMENTO.extract('/amostra/facturacao').getClobVal()
    from tabxml P
   where P.DOCUMENTO.extract('/amostra/numero/text()').getStringVal() = '123'
```

3.24 optimização de interrogações

3.24.1 Avaliação do desempenho de uma interrogação

Os procedimentos comuns para avaliar o desempenho de uma interrogação são:

- reescrever a interrogação em álgebra relacional;
- encontrar algoritmos para calcular resultados intermédios da forma mais simples;
- executar os algoritmos e obter os resultados.

Estes procedimentos podem esbarrar em dificuldades na medida em que na álgebra relacional podem-se obter expressões equivalentes, há problemas em lidar com sub-interrogações e os tamanhos intermédios obtidos são muito importantes. Os resultados da avaliação dependem da extensão das relações, o que nem sempre é possível saber dada a dinâmica de crescimento das tabelas.

Exemplo:

Medicos(M,NM,CE,Anos)
50 bytes por tuplo, 100 tuplos

Consultas(M,P,D,Preço)
20 bytes por tuplo, 5000 tuplos
O paciente P = 200 só tem 1 consulta!

Interrogação:

```
SELECT m.NM
FROM Medicos m,Consultas c
where m.M = c.M and P = 200 and Anos > 5
```

Tradução para a Àlgebra relacional:

$$\Pi_{NM}(\sigma_{P=200 \wedge Anos>5}(Medicos \bowtie Consultas))$$

A selecção e a projecção são realizadas após a junção!

Custo da solução: a volta de $100 * 5000 * 2 = 1000000$ tuplos processados!

Nova expressão em Álgebra Relacional:

$$\Pi_{NM}(\sigma_{P=200}(Consultas) \bowtie \sigma_{Anos>5}(Medicos)$$

É feito um *full table scan* às 2 tabelas, obtendo duas tabelas mais pequenas (cardinalidade respectivamente C_1 e 1) e depois é feita a junção entre as 2 tabelas resultantes.

Custo da solução: a volta de $100 + 5000 + C_1$ tuplos processados!
Por exemplo admitindo uma redução de cerca de 50% na selecção sobre Medicos ($C_1 = 50$), o resultado é:

$$100 + 5000 + 50 = 5150$$

Analisemos agora a expressão:

$$\Pi_{NM}(\sigma_{Anos>5}(\sigma_{P=200}(Consultas) \bowtie Medicos))$$

Assume-se também que existe um indíce em P!

É feito um *index scan em Consultas*, depois é feita a junção e finalmente a selecção e a projecção.

Custo da solução: a volta de $C_1 * 2 = 100$ tuplos processados!

O comando analisado pode ser traduzido de várias maneiras para a àlgebra relacional, obtendo-se resultados muito diferentes. A optimização de queries considera dferentes soluções ou caminhos antes de decidir executar uma determinada interrogação.

As expressões seguintes são expressões equivalentes mas com custos muito diferentes:

$$\sigma_C(E_1 \cup E_2) = \sigma_C(E_1) \cup \sigma_C(E_2)$$

$$\sigma_C(E_1 \otimes E_2) = E_1 \bowtie_C E_2$$

$$\sigma_C(E1 \setminus E_2) = \sigma_C(E_1) \setminus \sigma_C(E_2)$$

$\sigma_C(E_1 \otimes E_2) = \sigma_C(E_1) \otimes E_2$ se o conjunto de atributos de E_1 pertence a C.

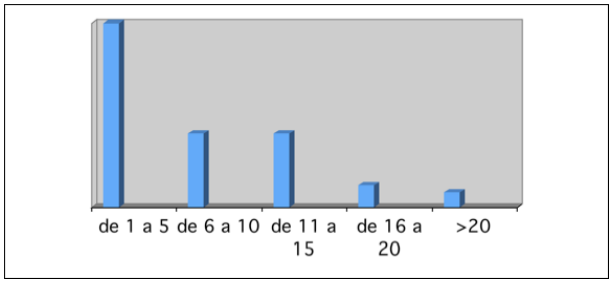

Figura 3.16: Histograma

$\sigma_C(E_1 \cap E_2) = \sigma_C(E_1) \cap \sigma_C(E_2)$

$\Pi_L(E_1 \bowtie E_2) = \Pi_L(\Pi_{(L \cup A_{E_2}) \cap A_{E_1}}(E_1) \bowtie \Pi_{(L \cup A_{E_1}) \cap A_{E_2}}(E_2))$

$\Pi_L(\sigma_C(E_1)) = \Pi_L(\sigma_C(\Pi_A(E_1)))$ onde A é o conjunto de atributos mencionados em C.

3.24.2 Histogramas

Os histogramas podem ser usados para avaliar o custo da selecção. Permitem analisar cardinalidades por intervalos.

Por exemplo, a seguinte tabela apresenta os médicos por anos:

Anos	Card
1-5	250
6-10	100
11-15	100
16-20	30
>20	20

O histograma é apresentado na figura 3.16.

3.25 Tuning

O tuning é um processo de acompanhamento da performance das interrogações de forma a escolher bons caminhos. O tuning pode sugerir a reescrita das interogações / código, a indexação de algumas tabelas e a eliminação de *full table scan*.