Escola Superior de Agricultura 'Luiz de Queiroz'
Universidade de São Paulo

BASES DE DADOS RELACIONAIS

uma introdução com aplicação florestal

Luiz Carlos Estraviz Rodriguez
Silvana Ribeiro Nobre

P I R A C I C A B A
São Paulo - Brasil
Agosto – 2014 *(re-edição)*

ÍNDICE

1. INTRODUÇÃO [1]

O objetivo principal deste curso é introduzir conceitos básicos para a gestão de bases de dados relacionais e a sua aplicação no desenvolvimento de bancos de dados florestais. Para que este objetivo seja alcançado no curso, foram planejadas quatro fases. Os conceitos são introduzidos através de exercícios práticos na linguagem SQL (*Structured Query Language*) para manipulação de bases de dados.

Pelkki (1992), citado por Falcão (1998), sugere três passos iniciais antes da implementação de uma base de dados eficiente: (i) planejamento da base de dados com alto nível de abstração; (ii) desenho da base de dados com nível detalhado de abstração; e (iii) implementação da base de dados em um sistema. Primeiramente, neste curso, destaca-se a importância da fase inicial de planejamento e desenho da base de dados. Para ilustrar esta fase, discutem-se alguns princípios gerais que devem nortear a criação de uma base de dados relacional.

Já na seção seguinte, é apresentado um modelo genérico para armazenamento de dados florestais. Este exemplo tem como base as pressuposições introduzidas na seção anterior e revela características relacionais. Em seguida, aplicam-se as sentenças SQL para a criação da estrutura exigida pelo modelo genérico proposto. As sentenças usadas formam o conjunto SQL de definição de dados (*SQL Data Definition Statements*). Os exercícios baseiam-se principalmente no uso dos comandos de criação e alteração de tabelas e índices.

Na etapa seguinte, discutem-se rapidamente os mecanismos de alimentação de bases de dados relacional. A inserção de dados é ilustrada a partir de exercícios com comandos de manipulação de dados (*SQL Data Manipulation Statements*). Os exercícios resolvidos nesta etapa envolvem o uso de comandos de inserção e atualização de dados. A última etapa trata da busca e pesquisa de informações em bases de dados relacionais. Inúmeras são as atividades que poderiam ser ilustradas nesta etapa do curso. Em ordem crescente de complexidade, serão desenvolvidos exercícios que mostrarão o grande número de possibilidades de busca que a linguagem SQL oferece.

---

[1] O material apresentado nesta apostila é resultado de uma compilação de outros trabalhos, principalmente de textos publicados por Falcão (1998), Greenspun (1999), Pelkki (1992) e Rodriguez (1999).

2. MODELAGEM DE BASES DE DADOS

As principais funções de um sistema gestor de informação permitem ao usuário:

(i) lidar adequadamente com transações;
(ii) encontrar rapidamente a informação procurada; e
(iii) inserir, atualizar, eliminar e buscar dados sem preocupação com detalhes como ordem e integridade do banco.

Para oferecer essas funções são idealizados modelos de bases de dados, que permitem a representação de entidades e relações entre estas. CODD (1982), citado por Falcão (1998), estabeleceu três componentes fundamentais como constituintes de um modelo de dados: estruturas permitidas; operadores para essas estruturas; e regras de integridade, que asseguram que as modificações dos dados não violam a integridade do sistema.

De fato, a necessidade de criar modelos construídos com base nestes componentes deu origem ao próprio modelo relacional de tratamento de dados que será discutido mais à frente. Antes, é importante introduzir o conjunto básico de características e propriedades desejáveis em uma base de dados moderna.

2.1. Características desejáveis de uma base de dados

EVEREST (1986) e FLEMMING e VON HALLE (1990), citados por Falcão (1998), apresentam as características de um bom modelo de dados:

1. Deve ser simples e não redundante,
2. Preciso na representação de fatos sobre os dados,
3. De fácil representação gráfica,
4. Com estrutura capaz de ser apresentada em uma simples descrição narrativa,
5. A estrutura do modelo deve ser independente de sua implementação, e
6. Deve incluir regras de validação, integridade e outras restrições.

Como a informação tem um ciclo de vida muito maior que as aplicações que a geram (APPLETON, 1993), é de se esperar que a estrutura de um modelo de dados contenha as mesmas relações semânticas existentes no mundo real (DURDING *et al.* 1977 e APPLETON, 1983, citados por Falcão, 1998).

2.2 Propriedades desejáveis de uma base de dados

Dentre as propriedades desejáveis para uma base de dados com as características acima citadas, destacam-se:

Ausência de redundância e inconsistência - Os dados de uma base não devem aparecer repetidos. Isto evita desperdício de espaço e riscos de introdução de inconsistências. A redundância pode levar naturalmente à inconsistência, quando a alteração da informação em uma parte do sistema não se reflete imediatamente na outra.

Dados são partilhados - Vários usuários devem poder acessar a base de dados simultaneamente, e deve ser permitido que diferentes aplicações acessem a informação sem a necessidade de criação de dados adicionais.

Padronização - O administrador da base de dados (ADB) deve assegurar que todos os padrões necessários (nomes de tabelas, campos, tipos de dados e organização) sejam aplicados na representação da informação. Esta propriedade é particularmente importante quando se pensa em troca de informação entre diferentes bases de dados. A existência de um padrão assegura que essas trocas de informação sejam realizadas coerentemente. É essencial também em grandes bases de dados utilizadas por vários analistas ou usuários. A opção pela padronização deve ser feita no momento do projeto da base, quando ainda não são conhecidos e vivenciados os transtornos do uso de uma base desenhada sem esse cuidado.

Permite a inclusão de restrições de segurança - Por vezes a informação disponível numa base de dados não deve ser acessada por todos os usuários do sistema. O ABD precisa ter condições de impor restrições de acesso para garantir que apenas os usuários certos tenham acesso à informação mais sensível. Da mesma forma, deve ser possível determinar níveis de acesso à informação (somente consulta, inclusive modificação etc.).

Integridade - Deve assegurar que os dados na base de dados sejam verdadeiros. Duas entradas que pretendam representar o mesmo fato ou atributo é um exemplo de falta de integridade (é evidente que isto só acontece se houver redundância no modelo de dados). Outro tipo de falta de integridade é o desrespeito a regras básicas de alimentação do banco: por exemplo, uma árvore aparecer com uma altura de 200 metros (ao invés de 20 m), ou uma determinada medição feita numa parcela que não existe.

Independência entre aplicativos e dados - Diz-se que uma aplicação é dependente dos dados quando é impossível mudar a estrutura de armazenamento dos dados, ou técnica de acesso, sem afetar a aplicação. É extremamente desejável que uma aplicação seja independente dos dados. Ao ABD deve ser permitida a modificação da estrutura de armazenamento sem que se precise refazer as aplicações já existentes. Por exemplo, a entrada de outros tipos de informação na base de dados não pode comprometer o desempenho de aplicações anteriores. A adição de novos tipos de informação deve ser invisível para as aplicações já existentes. Como ilustração, suponhamos uma tabela que contém apenas 4 campos: parcelas, data de medição, número da árvore e DAP. Suponhamos ainda que exista

uma aplicação que calcula a área basal para cada medição de cada parcela. A adição de um campo para altura de cada árvore, por exemplo, não deveria afetar a aplicação existente.

Na escolha e implementação de uma base de dados dever-se-á, assim, ter sempre em conta estas exigências que são fundamentais para a construção de um sistema eficiente, eficaz e duradouro.

2.3. Tipos de bases de dados

Os diferentes sistemas de bases de dados refletem uma evolução na forma de representação da informação. Tabelas simples, arquivos de acesso seqüencial, modelos relacionais, e modelos orientados por objetos são fases de uma evolução natural, causada tanto pelo aumento de capacidade computacional, como pelo aumento das exigências e demandas dos usuários. Descrevem-se a seguir os mais populares (Falcão, 1998):

Bases de dados baseadas em uma tabela simples

Este é o tipo mais simples de base de dados. Consiste numa tabela única com toda a informação armazenada em um único objeto. Este tipo de base de dados é apropriado apenas quando existe uma única entidade para a qual se deseja armazenar dados. Serve, por exemplo, para listas telefônicas. Estes sistemas são muito limitados, pois dificultam o crescimento dos mesmos. Por exemplo, para adicionar a informação *altura da copa* de parte das árvores listadas em uma tabela com *código da árvore* e *DAP*, temos duas hipóteses: ou adicionamos um novo registro, repetindo a informação que identifica a árvore e colocando a altura da copa no lugar do *DAP*, ou adicionamos um novo campo a todas as árvores. Nenhuma destas alternativas é desejável, pois a primeira causa maior dificuldade no tratamento da informação e a segunda pode provocar um considerável desperdício de espaço de armazenamento. Novos campos fazem com que apareçam registros com valores nulos, gastando desnecessariamente recursos. Mas, de fato, o principal problema que ocorre em sistemas deste gênero é o freqüente aparecimento de inconsistências e dados repetidos.

Modelo de Rede

Rapidamente se percebeu que os sistemas de tabelas simples eram muito ineficientes, e de difícil manutenção em caso de crescimento da base, ou da necessidade de armazenamento de diferentes tipos de informação. O primeiro modelo padronizado, largamente aceito e que apareceu para *mainframes*, foi o **modelo de Rede** desenvolvido pelo *Comitee for Data Systems Language* (CODASYL), também os criadores da linguagem COBOL - COmmon Business Oriented Language. Os sistemas CODASYL usam o termo *tipo de registro* em vez de *tabela*, mas são basicamente conceitos semelhantes. Os *tipos de registro* em CODASYL têm apontadores para outros registros em diferentes *tipos de registros*. Um apontador é um valor que especifica a localização em memória ou em um arquivo. O termo genérico que designa este tipo de entidade e relação é *listas ligadas* (*linked lists*). Os ponteiros ligam os registros numa estrutura organizada denominada *rede*. A grande vantagem deste sistema é

a boa performance que este tipo de organização tem quando se procuram os atributos que pertencem a um objeto específico. Como inconvenientes, temos: (i) caso se pretendam consultas mais genéricas, o tempo de processamento é muito elevado; e (ii) a gestão da base de dados é complexa e de difícil manutenção. De fato, para cada operação de modificação da base de dados, é necessária a atualização de registros e dos ponteiros a eles associados. Contudo, e apesar destes problemas, existem ainda em funcionamento bilhões de linhas escritas em COBOL gerindo sistemas CODASYL, o que prova a confiabilidade do sistema.

Modelo Hierárquico

O modelo hierárquico foi desenvolvido pela IBM para os seus *mainframes*. Foi o primeiro modelo de gestão de base de dados, e continua a ser usado em várias companhias no mundo todo. O primeiro produto a ser desenvolvido usando o modelo hierárquico foi o IMS (*Information Management System*), desenvolvido em conjunto pela IBM e *North American Aviation*. A equipe de desenvolvimento deste modelo usou a estrutura hierárquica, pois concordavam com o princípio filosófico de que todos os objetos da vida real são hierárquicos por natureza. Assim, este modelo, trata grupos repetidos usando uma estrutura semelhante a uma árvore invertida. A informação dos primeiros registros constituem os 'ramos' e a informação dos grupos repetidos são as 'folhas'. Este modelo à semelhança do CODASYL usa ponteiros para ligar os diferentes registros. A principal vantagem do modelo hierárquico é que as técnicas necessárias para encontrar registros relacionados são mais simples que os métodos empregados pelo modelo de rede.

Modelo Relacional

O Sistema Relacional de bases de dados foi criado por CODD (1970) nos laboratórios da IBM. A principal vantagem deste modelo é que não é necessário misturar apontadores e dados nas tabelas, pelo contrário, as tabelas estão ligadas por *chaves* com valores idênticos nas duas tabelas. O termo relacional se deve ao fato de 'relação' ser o equivalente matemático para tabela. Para a maioria dos casos esses dois termos podem ser tratados como sinônimos.

Resumidamente, um sistema relacional satisfaz duas propriedades: (i) a informação é percebida pelo usuário como tabelas e nada mais que tabelas, e (ii) os operadores à disposição do usuário são operadores que geram novas tabelas a partir de outras tabelas já existentes. Por exemplo, teremos operadores que extraem um conjunto de linhas de uma tabela, outros que extraem um conjunto de colunas, outros ainda sumarizam a informação de uma tabela em um formato tabular.

O modelo relacional assenta-se numa base teórica sólida. É graças a esta base teórica que os sistemas relacionais se comportam de uma forma perfeitamente definida, e os usuários podem prever com confiança o que o sistema fará numa determinada situação. Esta previsibilidade implica que as aplicações são fáceis de compreender, documentar, ensinar, aprender, usar e lembrar.

Uma base de dados relacional **é virtualmente independente da implementação**, podendo variar o sistema operacional ou o Sistema Gestor de Bases de Dados (SGBD). Isto se deve

ao fato das tabelas e todas as relações entre as diferentes tabelas estarem definidas de forma implícita nessas mesmas tabelas.

Modelo Orientado por Objetos

Quando se menciona que um sistema é *Orientado por Objetos (OO),* há sempre 3 características que lhe são inerentes: *encapsulação*, *hereditariedade* (*inheritance*) e *polimorfismo*. *Encapsulação* é o fato de a representação interna de um objeto poder ser invisível aos utilizadores finais e incluir no objeto tanto os dados como as funções que agem sobre esses dados. *A hereditariedade*, é uma forma de agrupamento das classes, juntando-as por atributos comuns, que são atributos de uma super-classe. Por exemplo, a classe *Planta*, pode derivar 3 outras, *Árvore*, *Arbusto* e *Erva*. Todas as 3 têm características comuns que são atributos também da classe *Planta*, dizendo-se assim que *herdam* os atributos da classe que as originou, mas tendo outros próprios que as diferenciam. *O polimorfismo* se refere à capacidade de um mesmo método poder ser aplicado a vários objetos ou uma mesma função a diferentes tipos de dados.

Uma das vantagens evidentes de uma abordagem orientada por objetos é o fato de se passar a trabalhar com as entidades e com os métodos próprios dessas entidades. Por exemplo, podemos aplicar o método *corte* a uma parcela modificando automaticamente os atributos susceptíveis de serem afetados por esta operação, como a rotação (no caso de uma parcela de eucalipto), a idade e o volume existente. A idéia geral é elevar o nível de abstração.

2.4. Chaves de indexação em sistemas relacionais

Chaves primárias

Uma **chave primária** é um atributo ou grupo de atributos que, de uma forma única, identifica um registro numa relação. Todos os atributos que não fazem parte da chave são dependentes funcionalmente desta. A regra geral é que todas as relações devem ter uma chave, e os valores dessa chave devem ser únicos. Este fato implica que os registros de uma relação devem ser únicos, ou seja, não deveriam existir duas linhas exatamente iguais. Entretanto, existem exceções, e por isso alguns SGBDs permitem a criação de tabelas sem chave primária.

Chaves estrangeiras

Chaves estrangeiras são atributos de relações que fazem referência a chaves primárias de outras relações.

Integridade referencial

Uma base de dados tem integridade referencial se esta não contém nenhuma chave estrangeira sem correspondência. Em outras palavras, se B se refere a A, então A tem que existir. Por exemplo, numa tabela de parcelas e medições, se a tabela de medições tem como

chave estrangeira o número da parcela, então qualquer número indicado nesta chave tem que ter correspondência com um registro existente na relação de parcelas.

Regras de chaves estrangeiras

Para as chaves estrangeiras, há duas grandes questões que necessitam ser respondidas: (i) (i) como apagar ou (ii) modificar os alvos das referências das chaves estrangeiras? O que acontece se um usuário tentar apagar o alvo da referência de uma chave estrangeira? Por exemplo, apagar uma parcela para a qual existe pelo menos uma medição.

Há duas opções que podem ser consideradas: (i) impedir o desencadear da operação (*restricted*) permitindo eliminações se não houver nenhuma medição nessa parcela; (ii) "cascatear" (*cascades*) a operação de eliminação para as tabelas associadas. Por exemplo, ao apagar uma parcela, as suas medições são também apagadas. Para ilustrar estas duas opções, supõe-se a existência de duas tabelas; uma de parcelas (PAR) e outra de medições (MED) dessas parcelas (Figura 2.1).

Tabela PAR	
ID_PAR	**Area**
1	400
2	400
3	500
4	900

Tabela MED		
ID_PAR	**DATA**	**ABASAL**
1	2/73	10.5
2	1/71	11.2
2	9/72	13.2
3	2/72	15.2

Figura 2.1 - Tabelas PAR e MED

Neste caso, ID_PAR em PAR é a chave primária, mas o mesmo campo em MED é uma chave estrangeira, que referencia as características da parcela em PAR. O que aconteceria se se apagasse a parcela 2? Com a opção *cascades*, a sua eliminação é propagada para a tabela MED, apagando todas as medições que tenham sido feitas nessa parcela. A opção *restricted* limita a possibilidade de eliminação de qualquer parcela enquanto existirem medições que a referenciem. Assim, para apagar uma parcela nestas condições teríamos primeiro que apagar todas as medições existentes nesta parcela e só então proceder à eliminação da parcela desejada.

O problema é o mesmo para uma atualização de um alvo de uma referência de uma chave estrangeira, com as mesmas duas opções a serem consideradas. Note-se ainda que os dois conceitos, chave estrangeira e integridade referencial são sempre definidos em termos um do outro, ou seja, não é possível explicar o conceito de "chave estrangeira" sem recorrer ao de integridade referencial.

2.5. Implementação prática de uma base de dados relacional

Esta seção baseia-se nas sugestões de JENNINGS (1994), citadas por Falcão (1998). O conceito de tabela é o que mais se aproxima do de relação, sendo as tabelas a forma mais generalizada de estruturação da informação. Uma tabela é um objeto que consiste em uma coleção de linhas (registros), que têm uma coleção idêntica de propriedades. Os valores associados às propriedades aparecem em colunas (campos). A terminologia planilha de cálculo é muito usada nas bases de dados que usam SQL para manipular informação em tabelas (Figura 2.2).

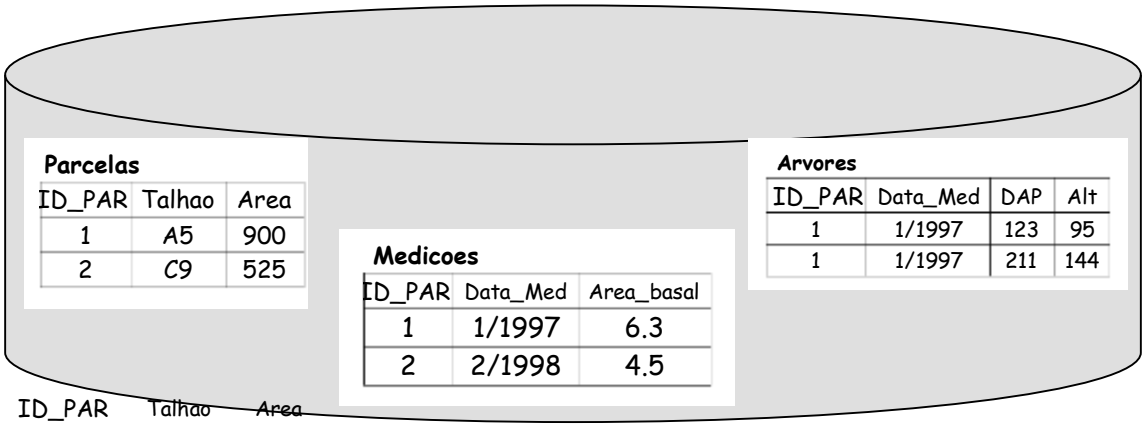
Atributos

	campo_1	**campo_2**	...	**campo_n**
Registro_1				
...				
Registro_N				

Figura 2.2 - Constituintes de uma tabela numa base de dados

Um ***registro*** é uma representação de um objeto ou de um fato do mundo real, tal como uma pessoa, uma árvore, uma parcela de inventário, ou um evento ocorrido em uma unidade de produção. Um ***campo*** descreve uma das características dos objetos ou fatos representados pelos registros. Corresponde às colunas das planilhas de cálculo, e é equivalente ao conceito de ***atributo***. A interseção de uma linha com uma coluna representa uma propriedade de um objeto real. É também designada ***célula***. A regra fundamental de todas as tabelas é que cada atributo possua apenas uma propriedade (*atomicidade*). Para ilustrar os pontos que se seguem, consideremos um conjunto de 3 tabelas relacionadas e estruturadas (Figura 2.3).

Figura 2.3 - Exemplo de uma base constituída por três tabelas



Parcelas

ID_PAR	Talhao	Area
1	A5	900
2	C9	525

Medicoes

ID_PAR	Data_Med	Area_basal
1	1/1997	6.3
2	2/1998	4.5

Arvores

ID_PAR	Data_Med	DAP	Alt
1	1/1997	123	95
1	1/1997	211	144

ID_PAR	Talhao	Area
1	A5	900
2	C9	525

Aqui ID_PAR é uma Chave Primária da Tabela de *Parcelas*. Assim, para cada Parcela existe um valor único que a identifica. Por outro lado, o mesmo campo na tabela de *Medicoes* é uma *chave estrangeira (foreign key)*, servindo para indicar que a Parcela 1 do talhão A5, onde foram obtidos DAP e Altura, foi medida em 1/1997 e apresenta área basal de 6.3 m2/ha. Mas ID_PAR não é a chave primária nesta tabela, porque, obviamente, existem repetições. A chave primária desta tabela é uma chave composta pelos campos ID_PAR e Data_Med, pois não pode existir nenhuma medição de parcelas no mesmo mês do mesmo ano.

Um outro aspecto a ser reforçado é que, à semelhança do conceito de *relação* anteriormente expresso, as tabelas também não tem uma ordem de registros definida. Não existe uma posição física de cada registro na base de dados. Depende da ordem como essa tabela é acessada. Geralmente os registros aparecem ordenadas de acordo com a chave primária, mas é uma ordem aparente que pode ser modificada eventualmente. Assim, não faz qualquer sentido falar de *número de registro*, quando nos referimos à posição de um objeto na base de dados.

2.6. O SQL

Como encontrar rapidamente a informação procurada? Sistemas Gestores de Informação lidam normalmente com inserções (INSERT), atualizações (UPDATE), e eliminação (DELETE)

de registros. Freqüentemente, entretanto, deseja-se também selecionar e extrair dados de dentro dos bancos (SELECT).

Historicamente, a primeira abordagem envolvia a utilização de arquivos texto para armazenamento de dados. Esta estratégia pode ser eficiente se os arquivos forem pequenos. Mas quando o volume de informações atinge dezenas ou centenas de milhares de dados, como em sistemas de inventário com dados sobre árvores individuais, os arquivos texto não são eficientes. Imagine também como seria possível selecionar árvores com uma certa idade, e diâmetro acima de um certo valor, simultaneamente com outros usuários querendo uma lista de árvores em determinados talhões por data de plantio. Seria fácil atender rapidamente e simultaneamente a esses usuários?

Ao longo do processo de evolução destes sistemas, consolidaram-se estratégias que se baseiam no uso de **índices**. Se forem mantidos índices que mapeiam as árvores por idade, diâmetro e talhão, por exemplo, o servidor não precisará mais percorrer todos os registros individualmente em busca da informação desejada. Isto implica, entretanto, na atualização desses índices durante os processos de inserção, atualização e eliminação de dados do banco.

Nestes casos, como interagir com o banco de dados eficientemente e de forma a manter a integridade do mesmo? A resposta foi dada por E. F. Codd em 1970, em um artigo intitulado *A relational model of data for large shared data banks* e publicado em junho desse ano no *Communications of the ACM*. Surgiam então os Sistemas Relacionais de Gestão de Bases de Dados (RDBMS, *relational database management systems*). Até essa data se grandes sistemas alterassem os seus índices ou o formato dos dados era muito provável que boa parte do sistema tivesse que ser re-escrito.

Para acessar a essas bases de dados, bem como para elaborá-los e mantê-los, criou-se a linguagem SQL *Structured Query Language* (linguagem estruturada de consulta). A *Structured Query Language, ou SQL,* como é designada hoje em dia, é a língua franca dos sistemas de gestão de bases de dados relacionais (JENNINGS, 1994). Para o programador, a maior inovação é a possibilidade de uso de uma linguagem de busca declarativa.

A primeira versão que E. F. Codd e os seus associados desenvolveram foi a SEQUEL (Structured English Query Language), que foi apenas usada em alguns protótipos iniciais. A segunda versão desta linguagem foi denominada SEQUEL/2, que logo se foi apelidada SQL. Vale lembrar que hoje em dia a SQL é uma linguagem *standartizada*, sendo a reformulação mais recente a definida pela Organização Internacional de Standards (ISO) e pela Comissão Internacional de Eletrotécnica (IEC) sob a referência ISO/IEC 9075:1992.

A versão de SQL empregada neste curso usa o padrão que a Microsoft disponibiliza no seu produto *Access*, que difere, em alguns pormenores, dos padrões internacionais. Assim, as sentenças SQL que serão apresentadas no curso poderão necessitar ligeiras correções quando usadas em outras plataformas. Para ver em maior detalhe as diferenças entre o MS Access SQL e o ANSI SQL-92, consultar as obras de referência da Microsoft (1995b - Professional Features) e JENNINGS (1994).

O SQL é o que se denomina uma linguagem de 4ª Geração (*4th Generation Language*, 4GL). As linguagens de 1ª geração eram os códigos de máquina. Informação em códigos binários com zeros e uns, que os primeiros computadores processavam. O *assembler* representa um nível mais evoluído em termos de abstração e possui uma sintaxe própria que facilita o entendimento do código escrito (linguagem de 2ªa geração). O FORTRAN foi a primeira linguagem de 3ª geração, ainda não totalmente estruturada, que foi seguida pouco anos depois pelo C, BASIC, e PASCAL, esta última já uma linguagem inteiramente estruturada do ponto de vista de orientação por procedimentos. Estas são linguagens onde as instruções são definidas por meio de seqüências de ações agrupadas em procedimentos.

As linguagens de 4ª geração, como o SQL, usam uma estratégia diferente, pois dizem ao computador *o que é que se pretende* e não detalham *o como agir para chegar aos resultados*. A maioria das linguagens de computação são processuais, isto é, permitem a detalhada definição do que fazer, passo a passo, criando procedimentos. Em SQL, o programador diz "Eu quero dados que satisfaçam ao seguinte critério" e cabe ao sistema RDBMS cumprir com essa missão da melhor forma possível.

O SQL emprega conceitos matemáticos da teoria dos conjuntos e é também considerada uma linguagem *orientada por conjuntos*. Comandos em SQL fazem sair conjuntos de dados em formato tabular, e os resultados são dependentes do conteúdo da base de dados. O número de elementos do conjunto de dados pode variar cada vez que executamos um comando, e pode variar freqüentemente num ambiente com vários usuários simultâneos.

É uma linguagem que pode ser extremamente complexa, e os padrões da sua *standartização* são naturalmente extensos. O documento que detalha a sua estruturação tem mais de 600 páginas. Neste curso cobriremos os conceitos e procedimentos mais básicos, procurando dar uma idéia da enorme potencialidade da linguagem através de exemplos.

O SQL permite fazer grande parte da gestão de um sistema relacional. Desde a criação de tabelas e estruturação de índices, à inserção de registros e realização de consultas de saída. O SQL está dividido em 3 grandes grupos de comandos, que por vezes se sobrepõem:

a) DDL - *Data Definition Language (Linguagem de definição de dados)*. É o conjunto de comandos que serve para criar, alterar e estruturar a base de dados (CREATE, DROP, ALTER).

b) DML - *Data Manipulation Language (Linguagem de manipulação de dados)*. São os comandos que permitem o acesso à informação disponível na base de dados, incluindo seleção, inserção, alteração, e eliminação de registros (SELECT, INSERT, UPDATE, DELETE).

c) DCL - *Data Control Language* (*Linguagem de controle de dados*). É o conjunto de comandos que controla o acesso à base de dados e estabelece os critérios de segurança.

Existem duas vantagens ao se usar um linguagem declarativa. A primeira é que as buscas independem da forma como os dados são representados (o RDBMS é livre para armazenar os

dados da forma que melhor lhe convier). A segunda tem a ver com o aumento de confiabilidade do software. É muito mais difícil criar pequenos erros (*bugs*) em uma busca SQL do que em um programa processual.

Um outro benefício das linguagens declarativas é que usuários menos treinados em programação são capazes de escrever programas bastante úteis. O programa *Access* permite o uso de sentenças SQL dos conjuntos DDL e DML. A segurança (comandos DCL), nestes casos, é feita internamente pelo próprio RDBMS (rotinas internas que constituem o cerne do *relational database management system* do Access).

Para uma visão mais detalhada da linguagem SQL sugerem-se os trabalhos de DATE e DARWEN (1993) ou, para uma abordagem mais abreviada, o capítulo 5 de JENNINGS (1993). Nas seções seguintes serão apresentados diversos exemplos de comandos SQL. Antes, entretanto, oferecemos para discussão um modelo genérico básico de gestão de bases de dados florestais.

3. UM MODELO BÁSICO DE BASE DE DADOS FLORESTAIS

O processo de decisão de um gestor de recursos florestais se baseia em um número finito de informações. Para que a gestão seja efetiva, é crucial o acesso a esses dados e precisam ser confiáveis, eficientes e eficazes os sistemas que os mantém. As bases de dados, quando estruturadas de acordo com abordagens relacionais, oferecem essas características e são recomendados para situações de média e alta complexidade. Este é o caso da gestão da informação em operações silviculturais e de manejo florestal. A recomendação do seu uso nestas áreas pode ser encontrada, por exemplo, em trabalhos como os de Alder (1995).

Fundamentalmente, a orientação básica é a de que as informações sejam organizadas em tabelas com linhas e colunas. Cada registro ocupa uma linha na tabela e se refere a um determinado objeto ou fato do mundo real, tal como uma fazenda, um talhão, uma árvore etc. Os campos de um registro descrevem atributos, que são características do objeto ou fato ao qual o registro se refere, e correspondem às colunas da tabela. Portanto, a interseção de uma linha com uma coluna representa uma única propriedade de um objeto ou evento real. Estas propriedades não devem aparecer repetidas em nenhuma outra instância na base de dados, oferecendo ao usuário da base a vantagem da não redundância e da atomicidade da informação (Falcão, 1998).

3.1 As entidades florestais do tipo objeto

O primeiro passo para a montagem de um modelo relacional é a identificação das entidades. É o que tentaremos fazer ao analisar uma primeira parte do conjunto florestal de informações normalmente conhecido como "cadastro florestal".

Em geral, a entidade que centraliza as informações é o ***talhão***. Os talhões apresentam-se agrupados freqüentemente por motivos econômicos, administrativos, legais etc. Devemos representar no banco de dados esses vários agrupamentos. Esses grupos de talhões podem ser ***fazendas***, ***regiões***, ***núcleos***, ***projetos***, ***áreas de gestão***, ***unidades de manejo*** ou outros ainda que porventura existam em aplicações reais. Os talhões também podem ser agrupados em mais de uma forma, isto é, podemos ter um agrupamento para fins legais em ***fazendas***, outro agrupamento para fins administrativos em ***projeto*** `PROJETO`, e ainda outro para fins de planejamento de longo prazo em ***unidade de manejo***.

Após a identificação das entidades, define-se a relação entre elas. Para talhões e grupos de talhões, temos a relação n:1 que quer dizer que vários talhões existem dentro de um grupo de talhões, ou que um grupo de talhões possui vários talhões.

Neste curso o `TALHAO` é constituinte dos seguintes grupos e sub-grupos de talhões: `REGIAO`, `DISTRITO` e `FAZENDA`. Estas quatro entidades, até agora identificadas, se relacionam entre

si da mesma forma e o próximo passo é representar essa idéia de forma gráfica como mostra a Figura 3.1.

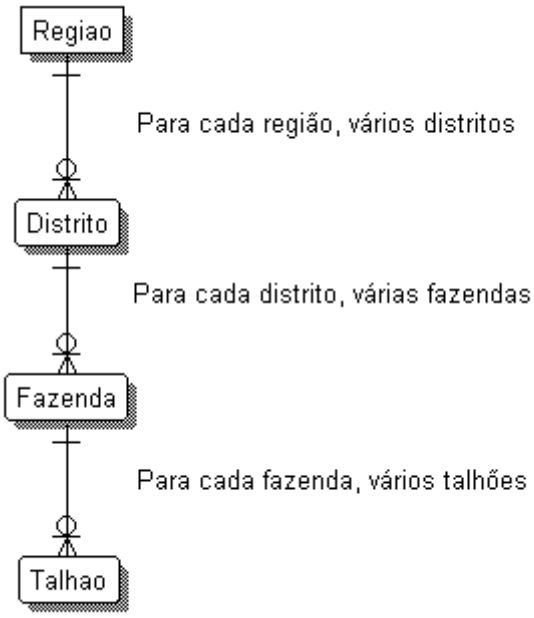


Figura 3.1 : Entidades TALHAO, REGIAO, DISTRITO e FAZENDA e seus relacionamentos

As quatro entidades citadas representam objetos existentes na realidade florestal. O próximo passo é identificar as características relevantes (os atributos) de cada entidade. O talhão poderia ser conceituado aqui como uma determinada área contígua de terra utilizada para a produção florestal. Os seus atributos básicos poderiam ser, então, a área, o índice de sítio (um indicador silvicultural de produtividadde), a distância da fábrica, o tipo de ***solo***, as condições de ***topografia***, o ***microclima*** prevalecente e qualquer outro atributo que permita caracterizar o talhão.

Alguns destes atributos de talhão podem representar outros objetos e, por isso, podem também ser entidades relacionadas com o talhão. É o caso de tipo de solo, classes de microclima e condições topográficas. É bom proceder com cuidado nesta fase, para evitar redundâncias. É importante observar que estas entidades representam objetos de características diferentes dos objetos representados pelas entidades TALHAO, FAZENDA, DISTRITO e REGIAO, sendo também relevantes para a correta caracterização do talhão sem redundâncias.

Visando aumentar a flexibilidade do modelo, não é aconselhável misturar os atributos das entidades tipo de solo, microclima e topografia como os atributos do talhão. Além de se

gastar recursos desnecessários, dificulta o cadastramento coerente dos talhões, a recuperação das informações, e o acréscimo no futuro de características que são únicas do solo, ou microclima ou topografia.

A Figura 3.2 mostra como essas entidades devem ser inseridas no modelo de dados e como devem se relacionar com a entidade TALHAO.

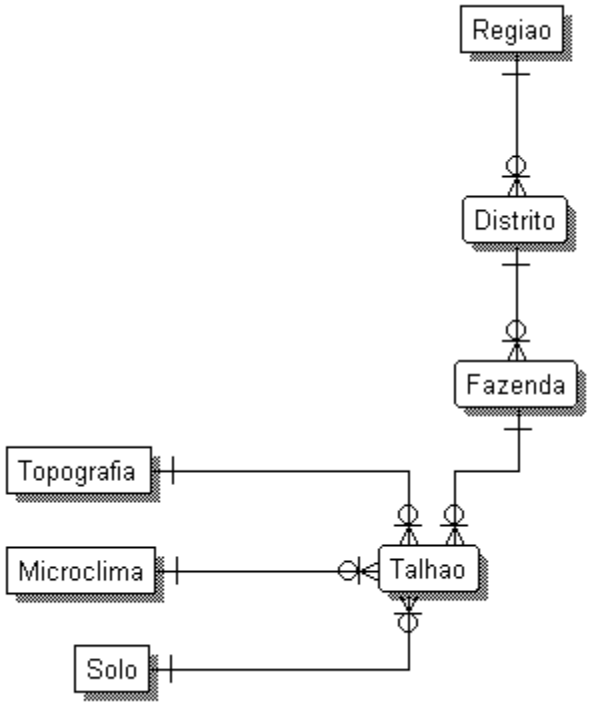


Figura 3.2 : Entidades SOLO, MICROCLIMA e TOPOGRAFIA

3.2 Entidades florestais do tipo evento

Existem também entidades que podem representar eventos associados aos objetos (entidades) já identificados. É o caso de ***Ciclo*** e ***rotação*** em sistemas florestais que envolvem o regime de talhadia simples (condução típica em povoamentos florestais que rebrotam após o corte). Sobre um talhão podem ocorrer vários ciclos florestais de manejo, sendo que a cada reforma do talhão dá-se início a um novo ciclo representando, portanto, um novo registro na entidade ***ciclo***. O mesmo é válido para rotação, pois quando se dá início a um ciclo cria-se também a primeira rotação. Os registros são lançados na entidade ***rotação*** quando do início das próximas rotações dentro de um mesmo ciclo.

Os atributos que podem caracterizar a entidade CICLO são: data do plantio, ***material genético*** plantado, espaçamento ou densidade de plantio e outros que se identifiquem especificamente com os ciclos de manejo conduzidos no talhão. A entidade material genético (MATGEN) é criada e se relaciona com ciclo da mesma forma que as entidades SOLO, MICROCLIMA e TOPOGRAFIA se relacionam com TALHAO.

Os atributos que podem caracterizar a entidade ROTACAO são: data de início da rotação, data de corte e intensidade de corte. No caso da primeira rotação de cada ciclo, a data de início da rotação é igual a data de plantio do ciclo, o que pode parecer uma redundância, mas nas rotações posteriores, esta data é igual à data da condução da brotação. Outros critérios para definição da data de início da rotação podem ser adotados. É importante observar, entretanto, que esta data é imprescindível para a determinação da idade futura dos povoamentos e deve ser alimentada com critério único para todo o banco.

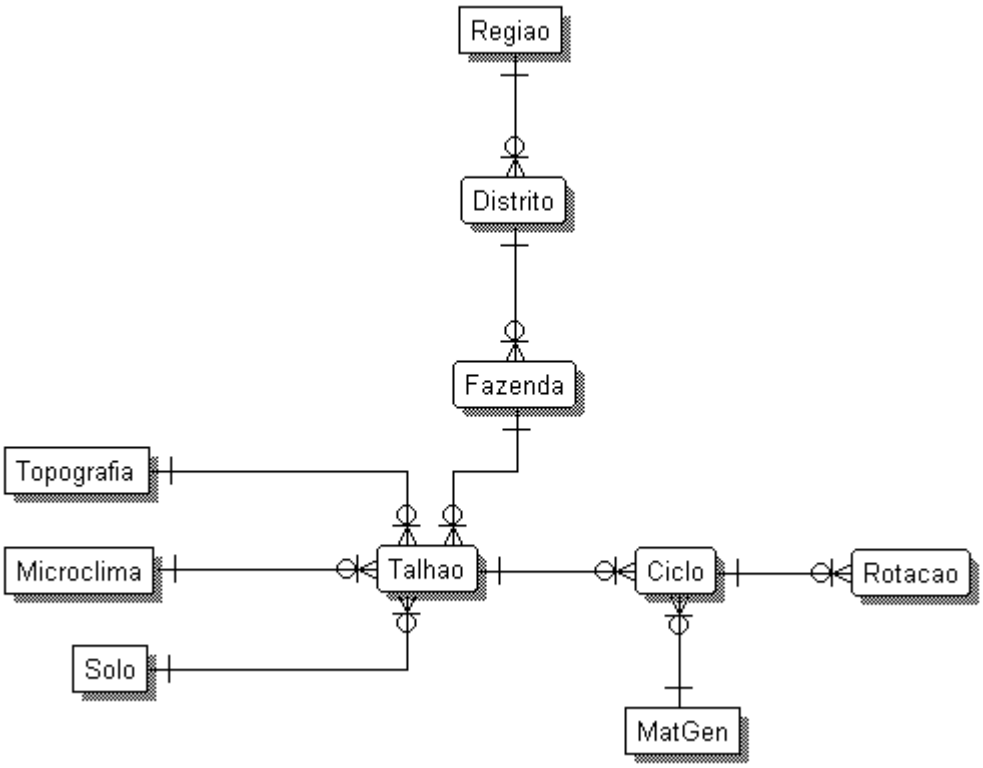


Figura 3.3: Entidades CICLO e ROTACAO

A Figura 3.3 mostra como as entidades CICLO e ROTACAO se relacionam com a entidade TALHAO. É importante enfatizar também que muitos outros detalhes poderiam enriquecer este modelo de dados. Poderia-se, por exemplo, optar pela criação de uma nova entidade para

espaçamento e ligá-la a entidade CICLO; oferecendo neste caso a mesma flexibilidade refletida pelo esquema de registro de vários materiais genéticos para cada ciclo, ou vários tipos de solo para cada talhão. Enfim, estas considerações permitem a inclusão de detalhes que particularizam e representam melhor a enorme diversidade da realidade que se deseja ver refletida no modelo.

A análise e identificação das entidades, e de seus relacionamentos, deve ser feita com a mesma minúcia para o restante das informações incluídas no base de dados. Centenas de outras entidades, em situações reais, são geralmente agregadas a essas compondo um grande banco.

Os dados de operações silviculturais planejados e realizados, poderiam ser entendidos como entidades do tipo evento relacionados a entidade ROTACAO. O consumo de insumos, de mão de obra, de horas-máquina para cada operação realizada, ocorrências de sinistros, dados meteorológicos, de levantamento de solos, detalhes dos diversos tipos de material genético e clonagem, produção e expedição de mudas, experimentação, colheitas e desbastes, dados de inventário e muitos outros objetos e eventos fazem parte do conjunto de informações necessárias à gestão dos recursos florestais.

A orientação básica é que os dados devem ser agrupados por assunto: dados de inventário, dados de clima, dados de experimentação, dados de produção de mudas etc. Existem algumas entidades auxiliares que se ligam a dados de mais de um grupo como Tipos de solo, Tipos de vegetação primária e outras semelhantes. Já outras entidades, como TALHAO, CICLO e ROTACAO, são centrais e fundamentais. Praticamente, todos os grupos de dados se relacionam a uma destas três entidades. A próxima seção detalha um grupo importante de informações em bancos florestais: o modelo básico para inventário. Esse módulo é apresentado, assim como a forma de relacionamento com o módulo central do modelo, isto é, com o cadastro florestal.

3.3 Entidades relacionadas ao Inventário

Dentre as entidades de inventário, a mais importante é a ***parcela***. A parcela de inventário florestal refere-se à unidade amostral onde são coletadas as medições. Seus atributos são: área, número de árvores, sítio e outros que se fizerem necessários para a sua caracterização mais precisa.

É importante que se faça um comentário sobre o atributo sítio. Este atributo já existe na entidade TALHAO, e haverá redundância se for também usado para caracterizar parcela, já que a localização da parcela é conhecida, e através dela pode-se chegar ao atributo sítio. Entretanto, se o atributo sítio da entidade PARCELA for uma característica muito específica devido à localização da parcela dentro do talhão, e o atributo sítio da entidade TALHAO significar o sitio predominante no talhão, não haverá redundância e será relevante mantê-los

separados. Este mesmo artifício pode ser usado para outras características como material genético, declividade e solo.

Identificam-se também neste módulo as entidades MEDICAO e ARVORE. São feitas em cada parcela várias medições. E, em cada medição de cada parcela, medem-se todas as árvores. Se conceituarmos a entidade MEDICAO como cada avaliação realizada em cada parcela, e a entidade ARVORE como a unidade alvo das avaliações em cada parcela, surge o relacionamento representado na figura 3.4.

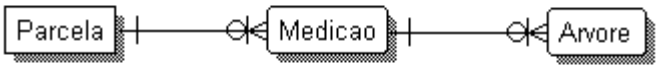


Figura 3.4: Entidades PARCELA, MEDICAO e ARVORE.

Pode ser relevante manter um controle das equipes de inventário responsáveis pelas medições, e para tanto cria-se esta entidade relacionando-a com a entidade MEDICAO. Ligada à entidade ARVORE cria-se a entidade NOTA para armazenar características, ocorrências e detalhes das árvores medidas. Muitos outros objetos e eventos poderiam enriquecer a modelagem dos dados do inventário, como tipos de inventário, variáveis medidas para diferentes tipos de inventário, estratos aos quais pertencem as parcelas, bases de estratificação, dados de cubagem etc.. Entretanto, a figura 3.5 mostra as entidades básicas que serão usadas nos exercícios e conceitos das seções seguintes.

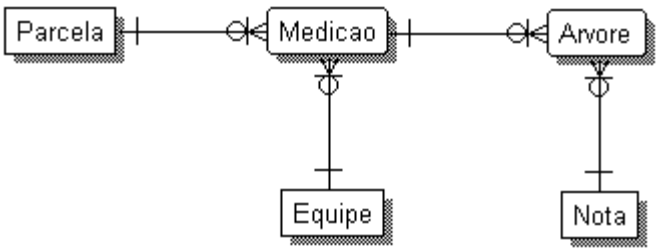


Figura 3.5: Modelo básico de inventário

Os dois módulos precisam agora ser integrados. Para isso, cria-se um vínculo entre os dois grupos que seja flexível, e que permita a independência dos aplicativos e consultas ao banco.

Sabe-se a localização de cada parcela, normalmente dentro de um talhão. Numa primeira análise optaríamos pela relação PARCELA e TALHAO, mas isso não seria correto. O mais apropriado é ligá-la à entidade ROTACAO, pois a parcela (unidade amostral do inventário) representa e acompanha especificamente o desenvolvimento da rotação, que se renova ao cabo de cada corte no talhão.

A integração dos módulos Cadastro e Inventário poderia se fazer através da relação "uma-para-várias" entre as entidades ROTACAO e PARCELA. Mas usaremos o artifício de criar uma entidade intermediária "um-para-um", para que o módulo Inventário fique totalmente independente do módulo Cadastro. Na prática, isto evita que os atributos do módulo cadastro tenham que ser repetido na entidade PARCELA, tornando muito mais flexível as aplicações de inventário. Ao mesmo tempo esta solução permite a localização perfeita da parcela de inventário e a busca e todos os dados de cadastro através da entidade ROTACAO. Este modelo é mostrado na figura 3.6. A relação entre PARCELA e INVECADA é lida assim: para cada ocorrência em INVECADA existe necessariamente uma correspondente em PARCELA; e para cada ocorrência em PARCELA pode existir no máximo uma ocorrência em INVECADA.

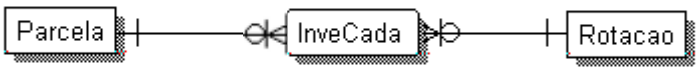


Figura 3.6: Cadastro e Inventário ligados através das entidades PARCELA e ROTAÇÃO

3.4 Lista de atributos

O próximo passo da modelagem de dados é construir a lista de atributos. Para cada uma das entidades identificadas lista-se com detalhes seus atributos. Os detalhes seriam o tipo de dado, tamanho, conjunto de valores possíveis, se podem ou não ser nulos e conceitos.

Os Quadros 3.1 e 3.2 mostram uma lista simplificada desses atributos. As informações cadastrais dos povoamentos florestais são mantidas nas tabelas apresentadas no Quadro 3.1. O Quadro 3.2 apresenta as tabelas que mantém os dados coletados em atividades de inventário florestal.

Este também é momento de se definir os atributos que farão parte da chave primária de cada uma das entidades. As entidades REGIAO, DISTRITO, FAZENDA, TOPOGRAFIA, MICROCLIMA, SOLO, MATGEN, e EQUIPE tem chaves primárias semelhantes. Pode-se

estabelecer para essas entidades um código numérico que pode ser auto-incremento caso o RDBMS no qual será implementado o banco ofereça este recurso.

Para a entidade TALHAO alguns comentários são importantes. O que identifica o talhão é o par de dados (código da fazenda, código do talhão). Isto é, não existe um par de dados repetido. Em outras palavras, não podem ser incluídos na entidade talhão dois talhões com o mesmo código na mesma fazenda. Pode-se impor que esse par de dados seja a chave primária da tabela, entretanto, como pode ocorrer a necessidade de renumeração dos talhões, e como essa entidade é central na base de dados isso pode ser desastroso. A chave primária dessa entidade é utilizada praticamente por todos os grupos de dados florestais, e pode-se chegar a milhões de registros como é o caso das operações florestais durante vários anos em milhares de talhões.

Cria-se, então, um identificador de talhão, Id_Talhao, como chave primária da entidade TALHAO e um índice único nos campos Cod_Fazenda e Cod_Talhao para garantir a unicidade de talhão. O Id_Talhao é propagado para as entidades CICLO, ROTACAO, INVECADA e para todas as entidades que se relacionem com talhão, ciclo ou rotação.

A chave primária para CICLO é uma chave composta dos atributos: Id_Talhao e Ciclo, sendo que o atributo Ciclo representa o número do ciclo em que se encontra a floresta plantada no talhão. A chave primária para ROTACAO é composta de Id_Talhao, Ciclo e Rotacao, sendo que o atributo Rotacao representa o número da rotação em que está a floresta plantada no talhão.

A chave primária de PARCELA, Id_Parcela, é o próprio número da parcela utilizado para a identificação usual de uma parcela de inventário. A chave de MEDICAO é semelhante a chave de CICLO, composta do Id_Parcela e NumMedicao, sendo que NumMedicao é um atributo que identifica o número da medição efetuada na parcela.

A chave primária de ARVORE deve identificar uma medição de cada árvore, portanto deve ser composta do identificador da medição (Id_Parcela e Medicao) e do identificador de cada árvore (ArvNumero).

A chave primária de NOTA é constituído pelo próprio código de avaliação usado durante a medição, e a descrição é o seu conceito. Os demais atributos, além da chave primária estão listados nos Quadros 3.2 e 3.2. Sobre esses atributos deve-se comentar que seus nomes devem ser representativos do seu conteúdo. Deve-se procurar dar nomes completos sem muitas abreviações, que não são mais necessárias com os recursos da informática moderna.

Quadro 3.1: Tabelas de dados cadastrais da base florestal

Tabela	Informação armazenada	Atributos
REGIAO	Regiões geográficas	Identificação (*Cod_Regiao*) e descrição (*Dsc_Regiao*) das regiões.
DISTRITO	Distritos regionais	Identificação (*Cod_Distrito*); descrição (Dsc_Distrito) dos distritos e região (*Cod_Regiao*) à qual pertence o distrito.
FAZENDA	Unidades administrativas de área contígua	Identificação (*Cod_Fazenda*); descrição (Dsc_Fazenda) das fazendas e distrito (*Cod_Distrito*) ao qual pertence a Fazenda.
TALHAO	Talhões florestais	Identificação (*Id_Talhao*) dos talhões; fazenda (*Cod_Fazenda*) à qual pertence o talhão; código (*Cod_Talhao*) do talhão; índice de sítio (*Sitio*) ou produtividade do talhão; topografia (*Cod_Topo*); microclima (*Cod_Micro*); tipo de solo (*Cod_Solo*); área (*Area*) e distância (*Distancia*) do talhão à fábrica.
TOPOGRAFIA	Tipos de declividade	Identificação (*Cod_Topo*) e descrição (*Dsc_Topo*) de acordo com a declividade.
MICROCLIMA	Tipos de microclima	Identificação (*Cod_Micro*) e descrição (*Dsc_Micro*) do microclima. Por exemplo, exposição do terreno face ao sol e predominância de ventos.
SOLO	Tipos de solos	Identificação (*Cod_Solo*) e descrição (*Dsc_Solo*) dos tipos de solo.
MATGEN	Identificação do material genético florestal	Identificação (*Cod_MatGen*) do material genético disponível para a formação dos povoamentos florestais; espécie (*Especie*) e descrição (*Dsc_MatGen*) da origem genética do material.
CICLO	Ciclos de manejo florestal	Identificação do Talhão (*Id_Talhao*); identificação do ciclo de produção florestal (*Ciclo*) em desenvolvimento no talhão; data de plantio (*DataPlantio*); densidade (*DensPlantio*) e material genético (*Cod_MatGen*) plantado no início do ciclo de manejo.
ROTACAO	Estágios florestais de manejo	Identificação do ciclo (*Id_Talhao e Ciclo*); rotação (*Rotacao*) ou fase do ciclo; data que marca o início da Rotação (*DataInicio*); data de corte (*DataCorte*) e intensidade de corte (*IntCorte*).

É interessante notar o artifício utilizado para vincular as tabelas do módulo Cadastro com as tabelas do módulo Inventário. O elo de ligação, a tabela `INVECADA`, possui apenas (i) a identificação da entidade rotação composta pelos campos Id_Talhao, Ciclo e Rotação e (ii) a identificação da entidade parcela representada pelo campo Id_Parcela. Desta forma, qualquer evento envolvendo atividades de inventário florestal estará sempre associado a uma determinada rotação, ciclo e talhão. A recuperação de dados, quer seja a partir de informações cadastrais ou de eventos de inventário, produzirá sempre resultados vinculados e íntegros.

Dentre outras vantagens, o desenho relacional apresentado (i) mantém os atributos das entidades *talhão*, *ciclo* e *rotação* organizados em tabelas distintas; (ii) os atributos de *parcelas*, *medições* e *árvores*; consideram a possibilidade de se alocarem múltiplas parcelas em um mesmo talhão; e (iii) mantém a integridade interna da base de dados através de relacionamentos simples que vinculam as tabelas através de códigos e identificadores únicos.

O resultado é uma base de dados flexível capaz de manter os eventos florestais ininterruptamente organizados em uma série histórica para todas as unidades de manejo.

Quadro 3.2: Tabelas para dados de inventário florestal

Tabela	Informação armazenada	Atributos
PARCELA	Unidades amostrais	Identificação das unidades amostrais (*Id_Parcela*) que são identificadas e vinculadas a uma determinada rotação florestal; área (*Area*) da parcela e índice de sítio (*Sitio*) ou produtividade da parcela.
MEDICAO	Levantamentos nas unidades amostrais	Identificação da parcela (*Id_Parcela*); número da medição (Num_*Medicao*) na parcela; data da medição (*DataMedicao*) e equipe (Cod_*Equipe*) responsável pelas medições.
ARVORE	Mensurações efetuadas nos indivíduos que constituem as unidades amostrais	Identificação da mensuração efetuada em uma árvore (*Id_Parcela* + *NumMedicao*); árvore medida (*NumArvore*); vara medida (*NumVara*); diâmetro à altura do peito (*DAP*); altura (*ALT*) e nota ou qualidade (*Cod_Nota*) do fuste.
EQUIPE	Equipes de inventário florestal	Identificação da equipe (*Cod_Equipe*) e nome ou descrição dos membros da equipe (*Dsc_Equipe)*.
NOTA	Códigos qualificativos das árvores	Identificação do código de qualificação da árvore (*Cod_Nota*) e descrição (*Dsc_Nota*) da nota.

Outros grupos de tabelas com, por exemplo, informações operacionais poderiam ser facilmente agregados à base através de um artifício semelhante ao proporcionado pela tabela `INVECADA`. Esses dados permitiriam, por exemplo, o armazenamento de custos e rendimentos operacionais, de volumes efetivamente explorados e o acompanhamento de indicadores de eficiência e produtividade. O objetivo, ao apresentar a estrutura sugerida nesse capítulo, é ilustrar a importância e a utilidade de bases de dados criadas com princípios relacionais na área florestal.

3.5 Implementação do modelo

O próximo passo é a criação física do banco em um RDBMS. A maioria dos sistemas gerenciadores de bancos de dados possuem interfaces de criação do banco de forma amigável e intuitiva, nas quais podem ser criadas as tabelas, os campos, os relacionamentos, os índices, e as restrições mais diversas para manter a integridade do banco.

Muitos detalhes aqui omitidos, dependendo dos recursos disponíveis no RDBMS, facilitam bastante esta implementação. Podem também ser usadas ferramentas de modelagem (ferramentas Case) que criticam o modelo, facilitam sua criação e manutenção, e geram um conjunto de sentenças SQL para a criação do banco. Para grandes bancos essas ferramentas

são essenciais e não podem ser dispensadas, pois permitem uma eficiente documentação e manutenção do sistema desde o início.

A Figura 3.7 apresenta a estrutura relacional básica, implementada em MS Access®, que será estudada nos próximas etapas e que foi sugerida por Rodriguez (1999). Nessa proposta observam-se os vínculos entre tabelas que conferem o relacionamento e a consistência desejados. Para composição dessa estrutura, Rodriguez (1999) baseou-se nas sugestões apresentadas por Alder (1995) e Falcão (1998).

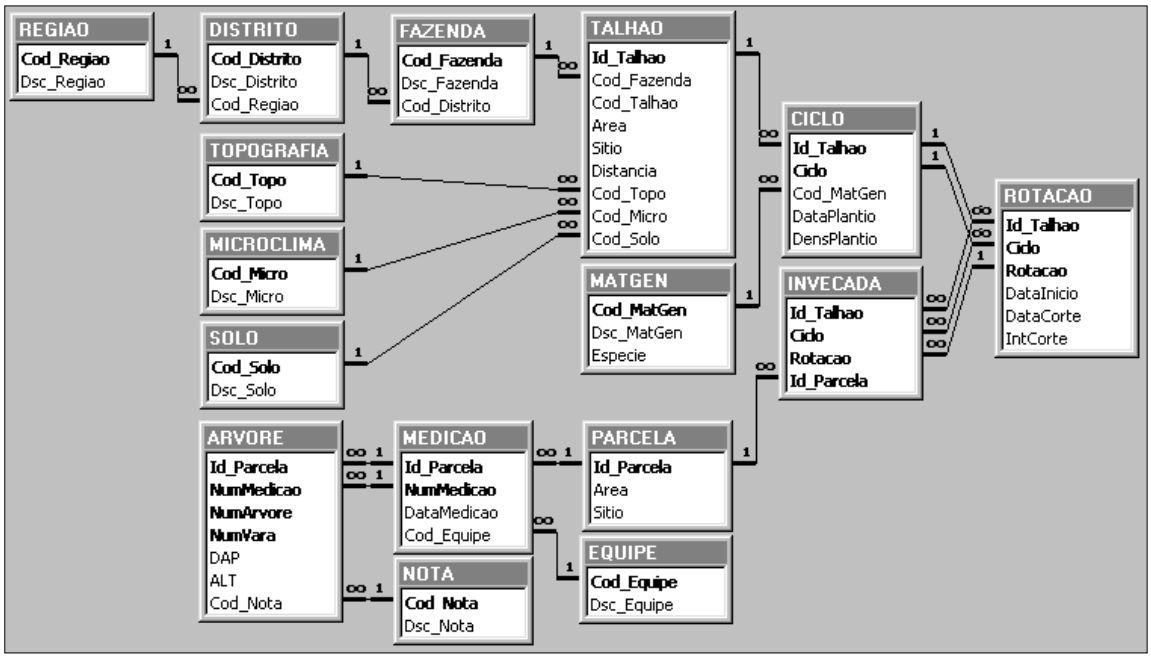


Figura 3.7 : Base florestal relacional integrando dados básicos de Cadastro e Inventário

4. DEFINIÇÃO E USO DE UMA BASE DE DADOS FLORESTAIS

O que é, afinal, uma base relacional de dados florestais? De fato, a resposta é bastante simples: é um conjunto de tabelas, vinculadas entre si, com dados florestais disponibilizados simultaneamente para vários usuários. Para criar a base, é preciso dizer ao RDBMS (*relational database management system*) quais tabelas compõem a base e quais colunas constituem cada tabela. Uma vez criada, serão rotineiras as atividades de alimentação e consulta à base de dados. Nesta seção são apresentadas as sentenças SQL necessárias para criação, alimentação e consulta de bases relacionais. Partiremos da estrutura básica relacional proposta na Figura 3.7 da página anterior.

4.1. Criação da estrutura física

No modelo relacional proposto na seção anterior a lista de fazendas é armazenada em uma tabela com três colunas: `Cod_Distrito`, `Cod_Fazenda` e `Dsc_Fazenda`. Cada registro na lista é uma linha nessa tabela.

O seguinte comando SQL poderia ser usado para criar a tabela `FAZENDA` com duas colunas:

```
CREATE TABLE FAZENDA (                                          4.1
    Cod_Fazenda AUTOINCREMENT,
    Dsc_Fazenda VARCHAR(50) );
```

O campo `Cod_Fazenda` é um campo do tipo "auto-numeração" que faz com que o Access numere automaticamente as fazendas. O campo `Dsc_Fazenda` é definido com um número variável de caracteres, sendo que 50 é a dimensão máxima. Se tivéssemos que eliminar a tabela `FAZENDA` da base, usaríamos o seguinte comando:

```
DROP TABLE FAZENDA;                                             4.2
```

Como pode ser observado na Figura 3.7, a tabela `FAZENDA` apresenta dois campos associados a chaves de relacionamento com outras tabelas. O campo `Cod_Fazenda`, por exemplo, aparece em negrito, para sinalizar que é a chave primária (PRIMARY KEY) da tabela, e a tabela `TALHAO` aponta para essa chave. Campos definidos como chave primária forçam o sistema a usar o seu conteúdo como um identificador único dos registros, que é o próprio conceito de chave primária. Em outras palavras, o sistema impedirá a inserção de um registro com *código de fazenda* duplicado.

Campos usados como chaves de relacionamento, que é o caso de `Cod_Fazenda` na tabela `TALHAO`, são automaticamente indexados. Ou seja, possuem seus próprios índices. Entretanto podemos definir qualquer tipo de índice para uma tabela, usando um ou mais campos, e este não ser necessariamente usado como chave de relacionamento. Vamos inicialmente supor que desejamos definir alguns índices para a tabela `FAZENDA`. Se a tabela

FAZENDA já existe, e precisamos criar um índice baseado no campo Dsc_Fazenda, com proibição de fazendas com descrições repetidas, usa-se a seguinte sentença:

```
CREATE UNIQUE INDEX indu_FAZENDA                                    4.3
    ON FAZENDA (Dsc_Fazenda) ;
```

Para apagarmos o índice, basta usar a seguinte sentença SQL:

```
DROP INDEX indu_FAZENDA                                             4.4
    ON FAZENDA ;
```

Se a tabela FAZENDA já existir, a chave primária baseada no campo Cod_Fazenda é definida através da seguinte sentença SQL:

```
CREATE INDEX indp_FAZENDA                                           4.5
    ON FAZENDA (Cod_Fazenda)
    WITH PRIMARY ;
```

Da mesma forma, para apagarmos o índice, bastaria usar a seguinte sentença SQL:

```
DROP INDEX indp_FAZENDA                                             4.6
    ON FAZENDA ;
```

Índices e chaves primárias, entretanto, podem ser definidas no próprio ato da criação da tabela. A tabela FAZENDA, com chave primária no campo Cod_Fazenda, poderia ser criada através do seguinte comando SQL (primeiro apague a tabela FAZENDA se já existir na base):

```
CREATE TABLE FAZENDA (                                              4.7
    Cod_Fazenda AUTOINCREMENT
        CONSTRAINT indp_FAZENDAS
        PRIMARY KEY,
    Dsc_Fazenda VARCHAR(50) );
```

Campos que fazem referência a campos definidos como chaves primárias em outras tabelas são chamadas de chaves externas (FOREIGN KEYs). Este é o caso do campo Cod_Distrito usado como chave externa para vincular a tabela FAZENDA com a tabela DISTRITO (vide Figura 3.7). Uma chave externa indica como as tabelas se relacionam - os dados nos campos de chave externa e de chave primária devem coincidir.

A sentença SQL 4.7 cria a tabela FAZENDAS com apenas dois campos e uma chave primária. Precisamos, portanto, adicionar o campo Cod_Distrito para vinculá-lo à chave primária da tabela DISTRITO. Quando criada permitirá a definição correta das restrições de integridade sugeridas na estrutura lógica apresentada na Figura 3.7. Esta proposta de fato envolve também um relacionamento entre as tabelas REGIAO e DISTRITO, de tal forma que possam existir vários distritos dentro de uma região, e várias fazendas dentro de um distrito. Para definirmos essas relações vamos criar primeiramente as tabelas REGIAO e DISTRITO com suas respectivas chaves primárias:

```
CREATE TABLE REGIAO (                                                    4.8
    Cod_Regiao AUTOINCREMENT
        CONSTRAINT indp_REGIAO
        PRIMARY KEY,
    Dsc_Regiao VARCHAR(50) );
```

```
 CREATE TABLE DISTRITO (                                                 4.9
    Cod_Distrito AUTOINCREMENT
        CONSTRAINT indp_DISTRITO
        PRIMARY KEY,
    Dsc_Distrito VARCHAR(50) );
```

Podemos agora definir as chaves externas que permitirão o estabelecimento dos vínculos desejados: `DISTRITO` vinculada com `REGIAO` através do campo `Cod_Regiao`; e `FAZENDA` vinculada com `DISTRITO` através do campo `Cod_Distrito`. Para estas definições faremos uso do comando ALTER TABLE, que incluirá nas tabelas as colunas desejadas como chaves externas:

```
ALTER TABLE FAZENDA                                                      4.10
    ADD Cod_Distrito INTEGER
        CONSTRAINT inde_FAZENDA
        REFERENCES DISTRITO ;
```

Note que um campo do tipo `INTEGER` faz referência a um campo do tipo `AUTOINCREMENT`, o campo `Cod_Distrito` na tabela `DISTRITO` é `AUTOINCREMENT` e na tabela `FAZENDA` ele é `INTEGER`. Isso ocorre porque o campo `AUTOINCREMENT` é um campo do tipo inteiro, sendo `AUTOINCREMENT` apenas na tabela onde é gerado automaticamente (`DISTRITO`):

```
ALTER TABLE DISTRITO                                                     4.11
    ADD Cod_Regiao INTEGER
        CONSTRAINT inde_DISTRITO
        REFERENCES REGIAO ;
```

O comando ALTER TABLE também pode ser usado para eliminar chaves e/ou índices. Por exemplo, os vínculos criados nas sentenças `4.10` e `4.11` poderiam ser apagados da seguinte forma:

```
ALTER TABLE FAZENDA                                                      4.12
   DROP CONSTRAINT inde_FAZENDA ;
```

```
ALTER TABLE DISTRITO                                                     4.13
   DROP CONSTRAINT inde_DISTRITO ;
```

A Figura 4.1 apresenta a estrutura lógica das relações definidas até o momento pelas sentenças `4.7` a `4.11`. Dispomos então dos comandos SQL necessários para criar a estrutura física do restante da base de dados florestal proposta na Figura 3.7.

Figura 4.1 - Estrutura física de relacionamento entre REGIAO, DISTRITO e FAZENDA

Listamos a seguir novas sentenças que complementam o trabalho de definição da base de dados florestais. Começaremos criando as tabelas TOPOGRAFIA, MICROCLIMA, SOLO e TALHAO. Note que TALHAO é deixada por último para que, durante a sua criação, possam ser definidas todas as chaves primárias e externas dessa tabela.

```
CREATE TABLE TOPOGRAFIA (                                          4.14
    Cod_Topo AUTOINCREMENT
        CONSTRAINT indp_TOPO
        PRIMARY KEY,
    Dsc_Topo VARCHAR(50) );
```

```
CREATE TABLE MICROCLIMA (                                          4.15
    Cod_Micro AUTOINCREMENT
        CONSTRAINT indp_MICRO
        PRIMARY KEY,
    Dsc_Micro VARCHAR(50) );
```

```
CREATE TABLE SOLO (                                                4.16
    Cod_Solo AUTOINCREMENT
        CONSTRAINT indp_SOLO
        PRIMARY KEY,
    Dsc_Solo VARCHAR (50) );
```

Para referenciar um campo auto-incremento, caso de Cod_Topo, Cod_Micro e Cod_Solo usa-se um campo INTEGER, como nos relacionamentos entre FAZENDA, DISTRITO e REGIAO. Desta vez, criamos todos os campos e as referências a outras tabelas em uma só sentença:

```
CREATE TABLE TALHAO (                                              4.17
    Id_Talhao AUTOINCREMENT
        CONSTRAINT  indp_TALHAO
        PRIMARY KEY,
    Cod_Fazenda INTEGER
        CONSTRAINT inde_TALFAZ
        REFERENCES FAZENDA,
    Cod_Talhao CHAR(5),
    Area NUMBER,
    Sitio NUMBER,
    Distancia NUMBER,
    Cod_Topo INTEGER
        CONSTRAINT inde_TALTOPO
        REFERENCES TOPOGRAFIA,
    Cod_Micro INTEGER
        CONSTRAINT inde_TALMICRO
        REFERENCES MICROCLIMA,
    Cod_Solo INTEGER
        CONSTRAINT inde_TALSOLO
        REFERENCES SOLO );
```

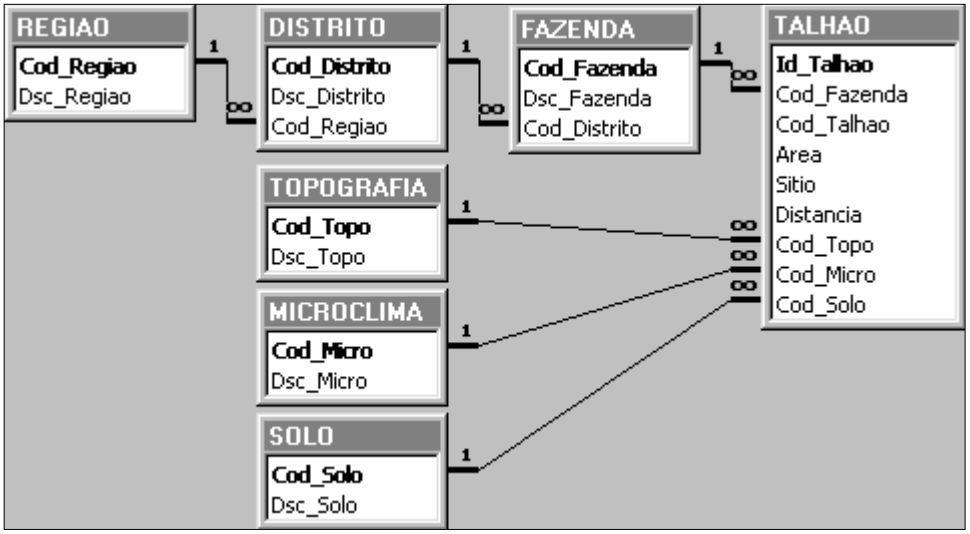


Figura 4.2 - Resultado da introdução de novas tabelas (`TOPOGRAFIA`, `MICROCLIMA`, `SOLO` e `TALHAO`) na base de dados florestais

A Figura 4.2 reflete as modificações na base introduzidas pelas sentenças `4.14` a `4.17`. Mas ainda falta garantir que não sejam incluídos talhões com o mesmo conjunto Fazenda + Código do talhão. Para tal precisamos criar um índice único nesses dois campos da tabela `TALHAO`. A sentença 4.18 mostra o comando `CREATE UNIQUE INDEX`.

```
CREATE UNIQUE INDEX indu_Talhao                                          4.18
    ON TALHAO (Cod_Fazenda, Cod_Talhao)
    WITH DISALLOW NULL ;
```

Para completar a criação das tabelas que armazenam informações cadastrais na base, definimos agora as tabelas `MATGEN`, `CICLO` e `ROTACAO`.

```
CREATE TABLE MATGEN (                                                    4.19
    Cod_MatGen AUTOINCREMENT
        CONSTRAINT indp_MATGEN
        PRIMARY KEY,
    Dsc_MatGen VARCHAR (50),
    Especie VARCHAR (50) );

CREATE TABLE CICLO (                                                     4.20
    Id_Talhao INTEGER
        CONSTRAINT inde_CICTAL
        REFERENCES TALHAO,
    Ciclo INTEGER ,
    Cod_MatGen INTEGER
        CONSTRAINT inde_CICMATGEN
        REFERENCES MATGEN,
    DataPlantio DATE,
    DensPlantio NUMBER,
    CONSTRAINT indp_CICLO
    PRIMARY KEY (Id_Talhao, Ciclo) );
```

Observe que a chave primária da tabela CICLO é composta de dois campos e essa informação é descrita na cláusula CONSTRAINT no final do comando CREATE TABLE. Isto também é feito para criar a tabela ROTACAO.

```
CREATE TABLE ROTACAO (                                                    4.21
    Id_Talhao INTEGER,
    Ciclo INTEGER,
    Rotacao INTEGER,
    DataInicio DATE,
    DataCorte DATE,
    IntCorte NUMBER,
    CONSTRAINT indp_ROTACAO
        PRIMARY KEY (Id_Talhao, Ciclo, Rotacao),
    CONSTRAINT inde_ROTCIC
        FOREIGN KEY (Id_Talhao, Ciclo)
        REFERENCES CICLO (Id_Talhao, Ciclo) );
```

Observe que a chave externa que referencia a tabela CICLO também é composta, e também é construída numa cláusula CONSTRAINT ao final do comando CREATE TABLE.

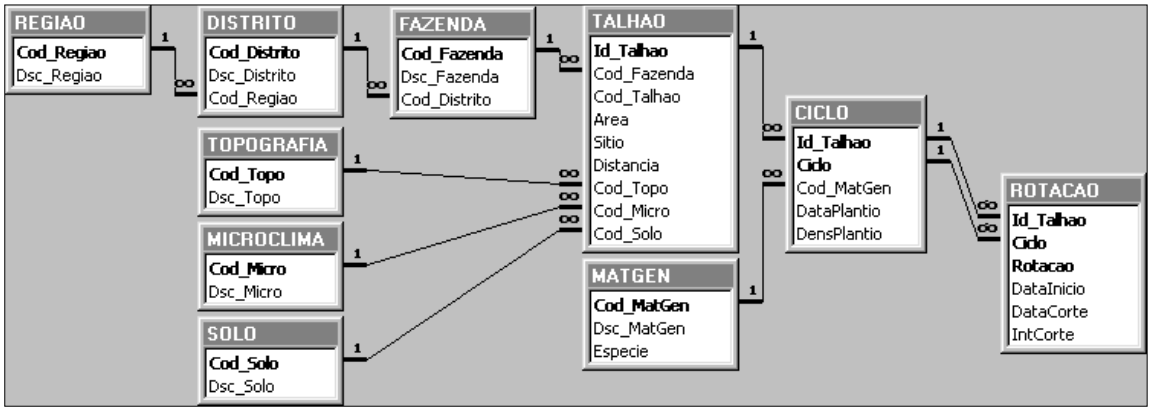


Figura 4.3 - Conjunto de tabelas que constitui o cadastro florestal na base de dados

As sentenças SQL implementadas até o momento permitiram gerar a estrutura física da base de dados florestal para armazenamento das informações cadastrais florestais básicas (Figura 4.3). O próximo conjunto de sentenças define um novo bloco na estrutura. Nesse bloco serão armazenadas informações básicas de inventário florestal, e será formado pelas tabelas: PARCELA, EQUIPE, NOTA, MEDICAO e ARVORE.

```
CREATE TABLE PARCELA (                                                    4.22
    Id_Parcela INTEGER
        CONSTRAINT indp_PARCELA
        PRIMARY KEY,
    Area NUMBER,
    Sitio INTEGER );

CREATE TABLE EQUIPE (                                                     4.23
    Cod_Equipe AUTOINCREMENT
        CONSTRAINT indp_EQUIPE
        PRIMARY KEY,
    Dsc_Equipe VARCHAR (50) );
```

```
CREATE TABLE NOTA (                                                    4.24
    Cod_Nota INTEGER
        CONSTRAINT indp_CODAVAL
        PRIMARY KEY ,
    Dsc_Nota VARCHAR (50) );
```

Nas tabelas `PARCELA` e `NOTA`, os códigos e chaves primárias não foram definidos como `AUTOINCREMENT`, porque o seu valor é conhecido e importante no contexto. Isto é, os códigos já existem e precisam ser informados e armazenados como são. O sistema não pode gerar um código automaticamente para eles.

```
CREATE TABLE MEDICAO (                                                 4.25
    Id_Parcela INTEGER
        CONSTRAINT inde_MEDPAR
        REFERENCES PARCELA,
    NumMedicao INTEGER ,
    DataMedicao DATE,
    Cod_Equipe INTEGER
        CONSTRAINT inde_MEDEQUI
        REFERENCES EQUIPE,
    CONSTRAINT indp_MEDICAO
    PRIMARY KEY (Id_Parcela, NumMedicao) );

CREATE TABLE ARVORE (                                                  4.26
    Id_Parcela INTEGER ,
    NumMedicao INTEGER ,
    NumArvore INTEGER ,
    NumVara INTEGER ,
    DAP NUMBER,
    ALT NUMBER,
    Cod_Nota INTEGER
        CONSTRAINT inde_ARVNOTA
        REFERENCES NOTA,
    CONSTRAINT indp_ARVORE
       PRIMARY KEY (Id_Parcela, NumMedicao, NumArvore, NumVara) ,
    CONSTRAINT inde_ARVMED
       FOREIGN KEY (Id_Parcela, NumMedicao)
       REFERENCES MEDICAO (Id_Parcela, NumMedicao) );
```

Observa-se na sentença `4.26` um exemplo de duas cláusulas `CONSTRAINT` para definição simultânea de múltiplas chaves. Este recurso foi utilizado, porque nesta tabela os campos `Id_Parcela`, `NumMedicao`, `NumArvore` compõem a chave primária da tabela `ARVORE`; e ao mesmo tempo os campos `Id_Parcela` e `NumMedicao` compõem a chave externa que referencia a tabela `MEDICAO`. O bloco independente de tabelas para armazenamento de dados de inventário florestal, e a respectiva estrutura de relacionamento, criado pelas sentenças `4.22` a `4.26`, pode ser observado na Figura 4.4.

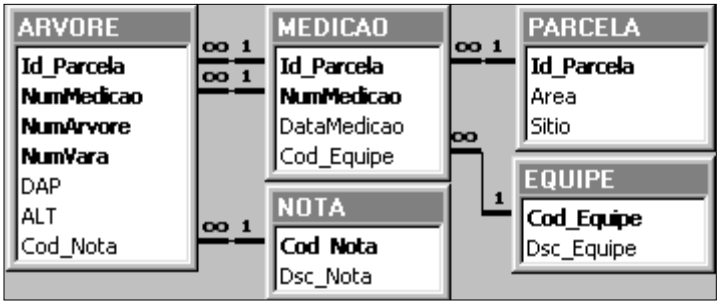


Figura 4.4 - Tabelas que compõe o módulo Inventário

Para completar a definição da estrutura proposta na Figura 3.7, falta apenas criar a tabela que vincula os dois módulos criados até o momento: o módulo de Inventário e o módulo de Cadastro. Será permitida a existência de duas ou mais parcelas de inventário em uma mesma rotação (por sua vez associadas ao ciclo e ao talhão), mas a uma parcela estará associada apenas uma rotação. Isto implica na ligação indireta entre parcela e rotação através da tabela `INVECADA`. A seguinte sentença SQL implementa essa estratégia.

```
CREATE TABLE INVECADA (                                                    4.27
    Id_Talhao INTEGER ,
    Ciclo INTEGER ,
    Rotacao INTEGER ,
    Id_Parcela INTEGER
        CONSTRAINT inde_AUXPARC
        REFERENCES PARCELA,
    CONSTRAINT indp_INVECADA
       PRIMARY KEY (Id_Talhao, Ciclo, Rotacao, Id_Parcela),
    CONSTRAINT inde_AUXROT
       FOREIGN KEY (Id_Talhao, Ciclo, Rotacao)
       REFERENCES ROTACAO (Id_Talhao, Ciclo, Rotacao) );
```

O resultado final é a estrutura apresentada na Figura 3.7. A próxima seção apresenta os comandos básicos para inserção de dados. Antes, entretanto, é conveniente resumir a sintaxe dos comandos SQL usados até agora.

CREATE TABLE *tabela*
(campo1 tipo [(*tamanho*)] [NOT NULL] [*índice1*]
[, *campo2 tipo* [(*tamanho*)] [NOT NULL] [índice2] [, ...]]
[, CONSTRAINT índicedemulticampos [, ...]])

onde:

tabela	Nome da tabela a ser criada.
campo	Nome do campo ou campos a serem criados na tabela.
tipo	Tipo de dados de campo na nova tabela.
tamanho	Tamanho do campo em caracteres (somente os campos Texto e Binário).
indice	Cláusula CONSTRAINT que define um índice de campo único. Veja CONSTRAINT logo mais.
indicedemulticampos	Cláusula CONSTRAINT que define um índice de campos múltiplos. Veja CONSTRAINT logo mais.

```
ALTER TABLE tabela
    {ADD
      {COLUMN campo tipo[(tamanho)] [NOT NULL] [CONSTRAINT indice]
      | CONSTRAINT indicedemulticampos}
    | DROP
      {COLUMN campo I CONSTRAINT nomedoindice} }
```

onde:

tabela	Nome da tabela a ser alterada.
campo	Nome do campo a ser adicionado ou excluído da tabela.
tipo	Tipo de dados de campo.
tamanho	Tamanho do campo em caracteres (somente os campos Texto e Binário).
indice	Índice para campo. Veja CONSTRAINT logo mais.
indicedemulticampos	Definição de um índice de campos múltiplos a ser adicionado à tabela. Veja CONSTRAINT logo mais.
nomedoindice	O nome do índice de campo múltiplo a ser removido.

```
CREATE [UNIQUE] INDEX indice ON tabela
    (campo [ASC|DESC][, campo [ASC|DESC], ...])[WITH { PRIMARY | DISALLOW
    NULL | IGNORE NULL }]
```

onde:

indice	Nome do índice a ser criado.
tabela	Nome da tabela existente que conterá o índice.
campo	Nome do campo ou campos a serem indexados. Para criar um índice de campo único, liste o nome do campo entre parênteses após o nome da tabela. Para criar um índice de campos múltiplos, liste o nome de cada campo a ser incluído no índice. Para criar índices descendentes, use a palavra reservada DESC; caso contrário, assume-se que os índices são ascendentes.

```
CONSTRAINT nome {PRIMARY KEY | UNIQUE | NOT NULL |
    REFERENCES tabelaexterna}
```

onde:

nome	O nome da restrição de campo único a ser criada.
tabelaexterna	Nome da tabela externa contendo um campo* com chave primária ao qual se fará o vinculo. Se a chave primária da tabela externa consistir em mais de um campo, deverá ser utilizada uma definição de restrição de campos múltiplos, listando todos os campos referenciais, o nome da tabela externa e os nomes dos campos referenciados na tabela externa, na mesma ordem em que os campos referenciais são listados. Se o campo ou campos referenciados forem a chave primária da tabela externa, não é necessário especificar os campos referenciados — por padrão, o mecanismo de banco de dados se comporta como se a chave primária da tabela externa fossem os campos referenciados.

```
DROP {TABLE tabela | INDEX índice ON tabela}
```

onde:

tabela	Nome da tabela a ser excluída ou a tabela a partir da qual um índice deve ser excluído.
indice	Nome do índice a ser excluído da tabela.

4.2. Alimentação e modificação dos dados de uma base

A inclusão e alteração de dados através de sentenças SQL em uma base relacional é implementada através dos comandos de inserção e atualização de dados (INSERT, DELETE e UPDATE). Estes são comandos do grupo SQL de manipulação de dados (*SQL Data Manipulation Statements*). Para conhecê-los melhor vamos começar a "*popular*" a base de dados florestais criada na seção anterior.

Os dados que estaremos introduzindo na base referem-se a uma propriedade florestal hipotética, denominada Fazenda Modelo. A proposta de modelagem da base exige que os dados cadastrais das propriedades florestais sejam considerados *atributos* da *entidade* FAZENDA. Por sua vez, as fazendas podem ser agrupadas em DISTRITO, que provavelmente estão associadas com REGIAO. Digamos que a empresa florestal tenha dividido a área de ação do grupo em regiões norte, sul, leste e oeste. As sentenças 4.28 e 4.29 nos permitem incluir na tabela REGIAO dois registros, as regiões " Sul" e "Leste", respectivamente. Não é necessário informar o código da região pois o definimos como AUTOINCREMENT, e será gerado automaticamente para cada nova região.

```
INSERT INTO REGIAO (Dsc_Regiao)                                    4.28
    VALUES ('Sul') ;

INSERT INTO REGIAO (Dsc_Regiao)                                    4.29
    VALUES ('Leste') ;
```

Este mecanismo de inclusão nos permite a introdução de registros individuais, sendo recomendada apenas quando o volume de registros é muito pequeno, e em situações que exigem total controle sobre cada inclusão. A eliminação de registros de uma tabela é extremamente simples, e também uma das mais "perigosas". Uma sentença simples do tipo DELETE * FROM [*tabela*]; apaga TODOS os registros de uma tabela e pode causar enormes danos. É sempre recomendável usar o comando de eliminação de registros em associação com a clausula WHERE, ganhando-se com isto um maior controle sobre os registro que se deseja eliminar. Por exemplo, a sentença 4.30 apaga da tabela REGIAO apenas o registro cuja descrição é "Sul".

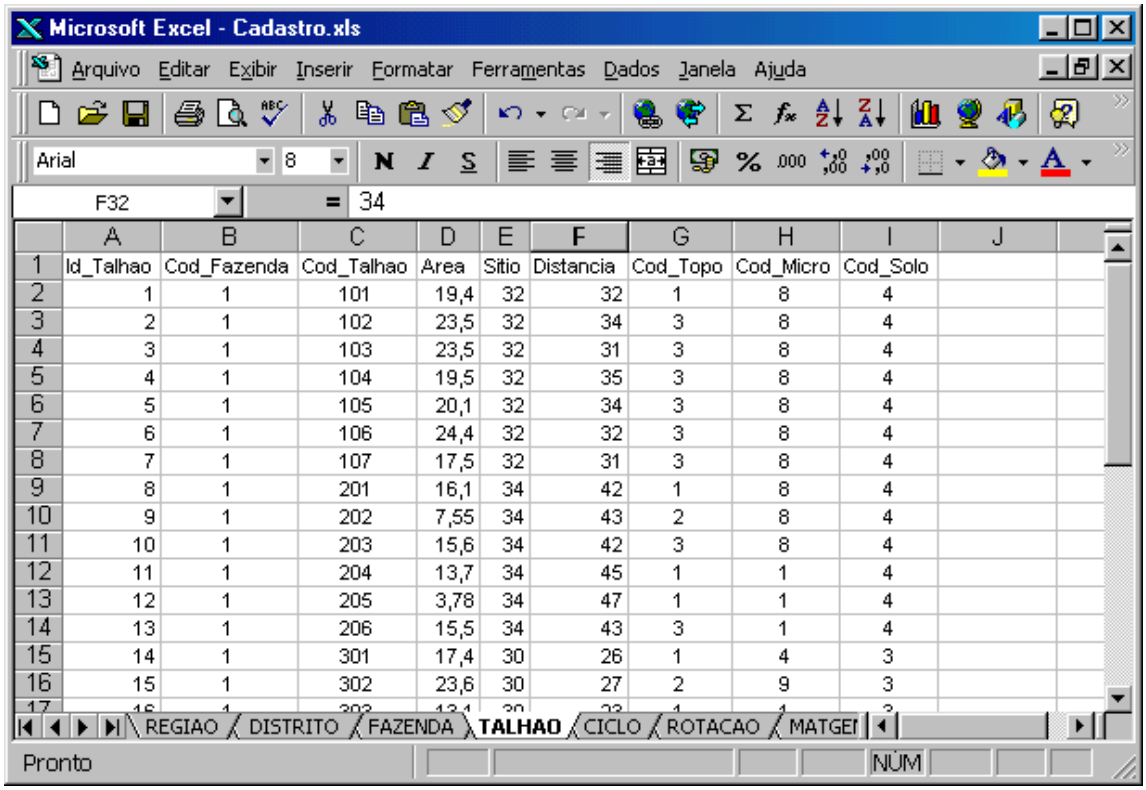
```
DELETE FROM REGIAO                                                 4.30
    WHERE Dsc_Regiao = 'Sul' ;
```

São muito comuns as ocasiões em que os dados de uma base precisam ser alterados, ou simplesmente atualizados. O comando UPDATE é normalmente empregado nestes casos. A sentença `4.31` mostra como exemplo a alteração da descrição do código da região Leste de "Leste" para "Regiao Leste".

```
UPDATE REGIAO                                                    4.31
   SET Dsc_Regiao = 'Regiao Leste'
   WHERE Dsc_Regiao = 'Leste' ;
```

Sentenças SQL semelhantes às definidas nos exemplos `4.28` e `4.29` permitem a inclusão de um registro de cada vez. Entretanto, seria extremamente tediosa a tarefa de alimentação de uma base de dados, se não existissem alternativas que permitissem a inclusão simultânea de vários registros de uma só vez. Os sistemas disponíveis hoje no mercado para gerenciamento de bases de dados oferecem mecanismos de importação que simplificam a inclusão de grandes volumes de informação.

Duas planilhas especialmente preparadas para este curso, guardam as informações que devem ser incluídas na nossa base florestal. Tratam-se das planilhas *Cadastro.xlsx* e *Inventario.xlsx* que podem ser importadas pelo MS Access através do menu "DADOS EXTERNOS" e "Excel" (Importar Planilha do Excel).



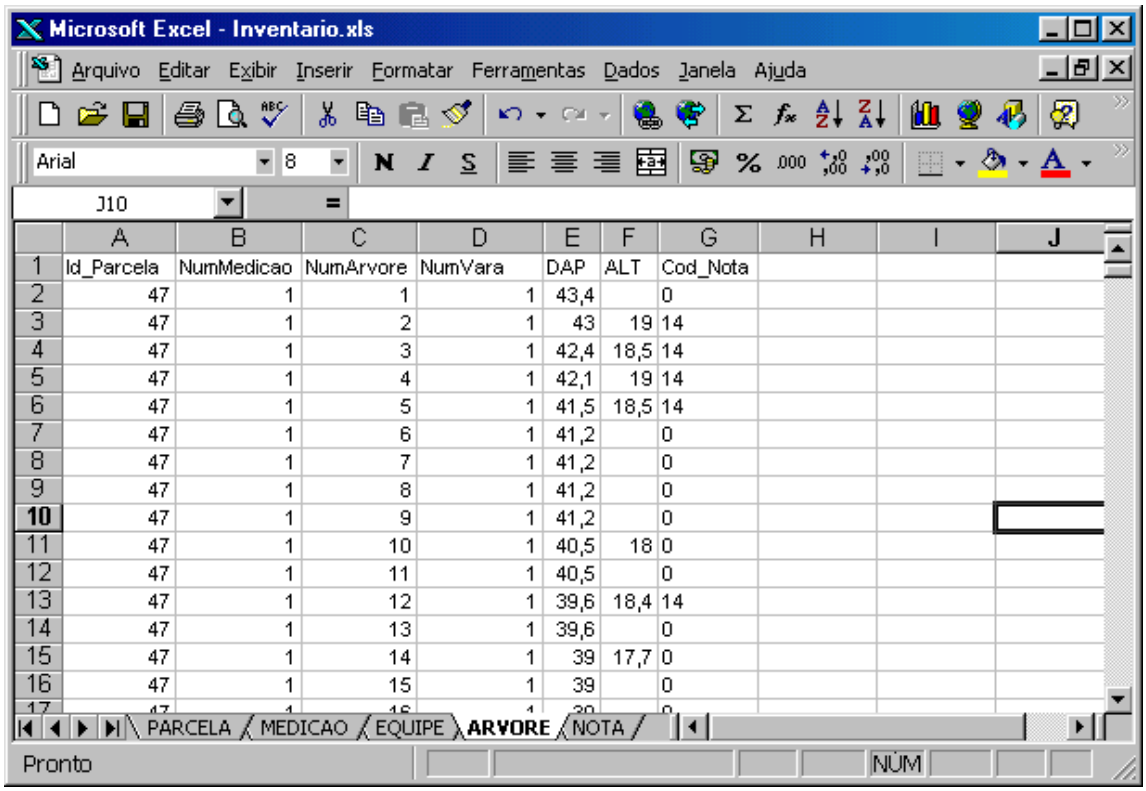
	A	B	C	D	E	F	G	H	I	J
1	Id_Talhao	Cod_Fazenda	Cod_Talhao	Area	Sitio	Distancia	Cod_Topo	Cod_Micro	Cod_Solo	
2	1	1	101	19,4	32	32	1	8	4	
3	2	1	102	23,5	32	34	3	8	4	
4	3	1	103	23,5	32	31	3	8	4	
5	4	1	104	19,5	32	35	3	8	4	
6	5	1	105	20,1	32	34	3	8	4	
7	6	1	106	24,4	32	32	3	8	4	
8	7	1	107	17,5	32	31	3	8	4	
9	8	1	201	16,1	34	42	1	8	4	
10	9	1	202	7,55	34	43	2	8	4	
11	10	1	203	15,6	34	42	3	8	4	
12	11	1	204	13,7	34	45	1	1	4	
13	12	1	205	3,78	34	47	1	1	4	
14	13	1	206	15,5	34	43	3	1	4	
15	14	1	301	17,4	30	26	1	4	3	
16	15	1	302	23,6	30	27	2	9	3	

Figura 4.5 - Detalhe da planilha Excel `TALHAO` de onde são importados dados

A planilha *Cadastro.xlsx* contém onze pastas correspondendo cada uma delas às onze tabelas que na base florestal organizam a informação cadastral: REGIAO, DISTRITO, FAZENDA, TALHAO, CICLO, ROTACAO, MATGEN, TOPOGRAFIA, MICROCLIMA, SOLO e INVECADA. A planilha *Inventario.xlsx* é formada por cinco pastas com informações que serão introduzidos nas tabelas destinadas à organização dos dados de inventário florestal: PARCELA, MEDICAO, EQUIPE, ARVORE e NOTA. A Figura 4.5, como ilustração, apresenta a aparência de uma das pastas (TALHAO) da planilha Cadastro.xls.

Essa possibilidade de usar uma planilha de cálculo para pré-processamento e consistência pode ser muito vantajosa. A melhoria da qualidade da informação pode, por exemplo, neste caso ser implementada de forma bastante flexível.

As tabelas que irão armazenar os dados de inventário constituem um grupo a parte, e são vinculadas ao módulo Cadastro pela tabela INVECADA. Desta forma, precisamos primeiro importar os dados para as tabelas de inventário. Depois faremos a importação da tabela INVECADA que cria o vínculo necessário entre os dois módulos Inventário e Cadastro. Neste caso, a informação será importada da planilha *Inventario.xlsx* de forma semelhante à realizada para os dados cadastrais.



	A	B	C	D	E	F	G
1	Id_Parcela	NumMedicao	NumArvore	NumVara	DAP	ALT	Cod_Nota
2	47	1	1	1	43,4		0
3	47	1	2	1	43	19	14
4	47	1	3	1	42,4	18,5	14
5	47	1	4	1	42,1	19	14
6	47	1	5	1	41,5	18,5	14
7	47	1	6	1	41,2		0
8	47	1	7	1	41,2		0
9	47	1	8	1	41,2		0
10	47	1	9	1	41,2		0
11	47	1	10	1	40,5	18	0
12	47	1	11	1	40,5		0
13	47	1	12	1	39,6	18,4	14
14	47	1	13	1	39,6		0
15	47	1	14	1	39	17,7	0
16	47	1	15	1	39		0

Figura 4.6 - Detalhe da planilha Excel ARVORE de onde são importados dados

A pasta ARVORE da planilha *Inventario.xlsx*, exibida na Figura 4.6, contém 7085 registros com medições de DAP, altura e código de avaliação da árvore. O número registrado para DAP

representa de fato CAP, circunferência à altura do peito, e não diâmetro. Estas informações refletem várias medições periódicas realizadas em parcelas permanentes de inventário florestal distribuídas na razão de pelo menos uma por talhão.

Para completarmos a inclusão de dados na base falta apenas introduzir os dados presentes na tabela INVECADA. Os campos dessa tabela fazem o vínculo necessário para integrar o módulo Cadastro com o módulo Inventário. Processo semelhante, por exemplo, poderia existir para outros módulos que agrupassem entidades e respectivos atributos relacionados com as unidades básicas de gestão florestal (talhões) definidas no cadastro. O acompanhamento e registro de atividades operacionais, por exemplo, que permitissem avaliações futuras de rendimento e custo, poderiam ser armazenadas em um novo módulo denominado *Operações*, por exemplo.

Um resumo da sintaxe das sentenças SQL usadas nesta seção é apresentada a seguir.

Inserção de registro único:

```
INSERT INTO destino
    [(campo1[, campo2[, ...]])]
  VALUES (valor1[, valor2[, ...])
```

Inserção de registros múltiplos:

```
INSERT INTO destino [IN bancodedadosexterno]
    [(campo1[, campo2[, ...]])]SELECT [origem.]campo1[, campo2[,
...]FROM expressãodetabela
```

onde:

destino	Nome da tabela ou consulta à qual acrescentar os registros.
Bancodedadosexterno	Caminho até um banco de dados externo. Para obter uma descrição do caminho, consulte a cláusula IN.
origem	Nome da tabela ou consulta a partir da qual os registros vão ser copiados.
campo	Nomes dos campos aos quais os dados serão acrescentados, se seguindo um argumento destino, ou os nomes dos campos a partir dos quais os dados serão obtidos, se seguindo um argumento origem.
Expressãodetabela	Nome da tabela ou das tabelas das quais os registros são inseridos. Este argumento pode ser um nome de tabela simples ou um composto resultante de uma operação INNER JOIN, LEFT JOIN ou RIGHT JOIN ou uma consulta salva.
valor	Valores a serem inseridos nos campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: *valor1* é inserido no *campo1* do novo registro, *valor2* no *campo2*, e assim por diante. Separar os valores com uma vírgula e colocar os campos de texto entre aspas (' ').

```
DELETE [tabela.*]
   FROM tabelaWHERE critérios
```

onde:

*tabela.**	Nome opcional da tabela da qual são excluídos registros.
tabela	Nome da tabela da qual são excluídos registros.
critérios	Expressão que determina os registros a ser excluídos.

```
UPDATE tabela
   SET novovalorWHERE critérios;
```

onde:

tabela	Tabela contendo os dados que serão modificados.
novovalor	Expressão que determina o valor a ser inserido em um campo específico dos registros atualizados.
critérios	Expressão que determina quais registros serão atualizados. Apenas os registros que satisfazem à expressão são atualizados.

4.3. Consultas

Os procedimentos de consulta são os mais utilizados dentre todos os recursos da SQL. Uma consulta SQL tem geralmente a forma `SELECT... FROM... WHERE...` e a sua utilidade reside no enorme potencial de exploração da base de dados oferecido por suas expressões de recuperação da informação. Depois da estruturação e alimentação de uma base, estas expressões irão justificar a própria existência da base e o valor dos dados nela contidos.

As variações e possibilidades de uso dos comandos SQL de busca são praticamente ilimitados. Explorar essas alternativas vai além do escopo deste curso. Nesta seção serão apresentadas apenas superficialmente algumas sentenças básicas que ilustram o potencial da linguagem SQL e as vantagens de se terem usado princípios relacionais quando da construção da base de dados.

Basicamente, o comando SELECT indica quais são as colunas da base de dados escolhidas durante a consulta. A cláusula FROM indica quais tabelas serão usadas durante a consulta. E a clausula WHERE irá restringir os registros selecionados por meio de condições lógicas.

Por exemplo, para retirarmos da tabela `MEDICAO` uma lista das expedições e respectivas equipes implementadas após 01/01/1990, é suficiente dizer:

```
SELECT Cod_Equipe , DataMedicao                                    4.32
  FROM MEDICAO
 WHERE DataMedicao > #JAN/01/1999# ;
```

O uso do símbolo "#" delimitando a data no formato MM/DD/AAAA (sistema americano) garante a portabilidade da sentença entre plataformas e sistemas operacionais diferentes.

Isto garante que a data seja lida corretamente, qualquer que seja o idioma e padrão local de configuração do sistema. O resultado da sentença `4.32` é uma lista de duas colunas com as medições realizadas após o dia primeiro de janeiro de 1999 e as respectivas equipes.

Mas várias medições foram feitas no mesmo dia, e quiséssemos saber em que dias as equipes trabalharam? A opção `DISTINCT` acrescentada ao `SELECT` faz com que o resultado não apresente linhas iguais e o resultado será o desejado.

```
SELECT DISTINCT Cod_Equipe, DataMedicao                                    4.33
  FROM MEDICAO
 WHERE DataMedicao > #JAN/01/1999# ;
```

Talvez, neste caso, fosse também interessante usar o SQL para agregar a informação e permitir, por exemplo, conhecer o número de expedições realizadas:

```
SELECT Count(*) AS [Medições após 01/01/99]                                4.34
  FROM MEDICAO
 WHERE DataMedicao > #JAN/01/1999# ;
```

Ou, melhor ainda, listar os totais agrupados por equipe:

```
SELECT Cod_Equipe AS [Equipe],                                             4.35
       Count(*) AS [Medições após 01/01/99]
  FROM MEDICAO
 WHERE DataMedicao > #JAN/01/1999#
 GROUP BY Cod_Equipe ;
```

Notam-se nesses casos o uso da função `count(*)` e da clausula `AS`. A clausula `AS` é utilizada para definir o novo nome do respectivo campo que aparecerá na tabela de resultados. Já a função `count(*)` conta o número de registros (inclusive nulos) e faz parte do conjunto SQL de funções de agregação (*aggregate functions*). Outros exemplos de funções de agregação são `Sum` (soma), `Avg` (média aritmética), `StDev` e `StDevP` (desvios padrão da amostra e da população), `Var` e `VarP` (variâncias da amostra e da população), e `Min` e `Max` (valores mínimo e máximo). Se usado na tabela `ARVORE`, por exemplo, o comando `Count(DAP)` apresentaria como resultado o número de registros com o campo `DAP` não nulo, e `Avg(DAP)` apresentaria a média aritmética desses valores.

Muitos dos dados em nossa base, entretanto, encontram-se dispersos em tabelas relacionadas. O poder das consultas reside na capacidade de agrupar esses dados dispersos. Ao tentar recuperar dados dispersos em diversas tabelas, o importante é certificar-se que as tabelas contendo os dados de interesse estão vinculadas entre si, direta ou indiretamente. Desta forma, se utilizarmos o comando `SELECT` na sua versão simples (`SELECT... FROM...`) para recuperação de dados em duas tabelas distintas associadas entre si, teremos como resultado o "*produto cruzado*" ou "*produto cartesiano*" das duas tabelas. Sendo assim, se por exemplo cada tabela contiver 5 registros, o resultado da pesquisa será uma lista com 25 registros (5X5).

Voltando para o nosso exemplo, temos os seguintes conteúdos nas tabelas `REGIAO` e `DISTRITO`:

Tabela REGIAO

Cod_Regiao	Dsc_Regiao
2	Regiao Leste
3	Regiao Norte
4	Regiao Oeste

Tabela DISTRITO

Cod_Distrito	Dsc_Distrito	Cod_Regiao
1	Rancho Fundo	3
2	Verde Vale	4

Observam-se 3 registros na tabela REGIAO e 2 registros na tabela DISTRITO. Vamos agora observar o *"produto cruzado ou cartesiano"* dessas tabelas obtido com a seguinte sentença:

```
SELECT * FROM REGIAO, DISTRITO ;                                    4.36
```

Resultado da sentença SQL 4.36

REGIOES.Cod_Regiao	Dsc_Regiao	Cod_Distrito	Dsc_Distrito	DISTRITOS.Cod_Regiao
2	Regiao Leste	1	Rancho Fundo	3
2	Regiao Leste	2	Verde Vale	4
3	Regiao Norte	1	Rancho Fundo	3
3	Regiao Norte	2	Verde Vale	4
4	Regiao Oeste	1	Rancho Fundo	3
4	Regiao Oeste	2	Verde Vale	4

Como podemos observar na tabela DISTRITO, existem apenas dois distritos registrados. Um associa o distrito "Rancho Fundo" com o código de região "3" e outro associa o distrito "Verde Vale" com o código de região "4". Como proceder para obter o nome da região de cada distrito, sendo que esses nomes aparecem definidos na tabela REGIAO. O SQL oferece duas estratégias principais de busca quando os dados encontram-se em tabelas separadas: uma baseada na clausula WHERE e outra na clausula JOIN. Vamos ver como poderiam ser definidas essas sentenças SQL:

```
SELECT Dsc_Regiao, Dsc_Distrito                                     4.37
  FROM REGIAO, DISTRITO
 WHERE REGIAO.Cod_Regiao = DISTRITO.Cod_Regiao ;
```

```
SELECT Dsc_Regiao, Dsc_Distrito                                     4.38
  FROM REGIAO INNER JOIN DISTRITO
    ON REGIAO.Cod_Regiao = DISTRITO.Cod_Regiao ;
```

Resultado da sentença SQL 4.37 ou 4.38

Dsc_Regiao	Dsc_Distrito
Regiao Norte	Rancho Fundo
Regiao Oeste	Vale Verde

Como seria de ser esperar, os resultados das sentenças 4.37 e 4.38 são iguais. Dois registros apenas indicando os distritos registrados em cada região.

Mas se a questão fosse listar as regiões cm seus distritos, mesmo que as regiões não tenham distritos cadastrados teríamos que utilizar a opção OUTER JOIN, assim:

```
SELECT Dsc_Regiao, Dsc_Distrito                                     4.39
  FROM REGIAO LEFT OUTER JOIN DISTRITO
    ON REGIAO.Cod_Regiao = DISTRITO.Cod_Regiao ;
```

Obtemos nesse caso todas as regiões e seus distritos. Como não existe distrito cadastrado para a região Leste, a coluna de distrito aparece nula para essa linha. Poderíamos, portanto, obter uma lista só das regiões que não possuem distritos cadastrados:

```
SELECT Dsc_Regiao, Dsc_Distrito                                    4.40
  FROM REGIAO LEFT OUTER JOIN DISTRITO
    ON REGIAO.Cod_Regiao = DISTRITO.Cod_Regiao
 WHERE Dsc_Distrito IS NULL ;
```

Vamos analisar um outro exemplo. Podemos aproveitar que na nossa base de dados as tabelas `MEDICAO` e `ARVORE` estão vinculadas pelos campos `Id_Parcela e NumMedicao`, e listar as árvores medidas durante o ano de 1995. A sentença, sem o uso da clausula `INNER JOIN`, poderia ser definida assim:

```
SELECT                                                             4.41
    Medicao.DataMedicao,
    Arvore.Id_Parcela,
    NumArvore, DAP, ALT, Cod_Nota
FROM
    MEDICAO, ARVORE
WHERE
    MEDICAO.Id_Parcela  = ARVORE.Id_Parcela AND
    MEDICAO.NumMedicao  = ARVORE.NumMedicao AND
    DataMedicao        >= #JAN/01/1995#     AND
    DataMedicao        <= #DEC/12/1995#
ORDER BY
    DataMedicao ASC ;
```

O mesmo resultado poderia ser obtido com o uso da clausula `INNER JOIN`.

```
SELECT                                                             4.42
    DataMedicao,
    Arvore.Id_Parcela,
    NumArvore, DAP, ALT, Cod_Nota
FROM
    MEDICAO INNER JOIN ARVORE
ON
  ( MEDICAO.Id_Parcela = ARVORE.Id_Parcela AND
    MEDICAO.NumMedicao = ARVORE.NumMedicao )
WHERE
    DataMedicao >= #JAN/01/1995# AND
    DataMedicao <= #DEC/12/1995#
ORDER BY
    DataMedicao DESC ;
```

Nota-se o uso da clausula `ORDER BY` para ordenar os resultados em ordem ascendente ou descendente (`ASC` e `DESC`, respectivamente). Podemos aproveitar a sentença `4.41`, alterando-a ligeiramente, para obter algumas estatísticas interessantes com as funções de agregação apresentadas há pouco. A sentença `4.43` conta o número de árvores para as quais foram medidos CAP e altura em 1995, e calcula médias, desvios e valores máximos e mínimos.

```
SELECT                                                                    4.43
    Count(DAP) AS [Número de CAPs],
    Min(DAP)   AS [CAP mínimo],
    Avg(DAP)   AS [Média dos CAPs],
    Max(DAP)   AS [CAP máximo],
    StDev(DAP) AS [Desvio dos CAPs],
    Count(ALT) AS [Número de Alturas],
    Min(ALT)   AS [Altura mínima],
    Avg(ALT)   AS [Média das Alturas],
    Max(ALT)   AS [Altura máxima],
    StDev(ALT) AS [Desvio das Alturas]
FROM
    MEDICAO, ARVORE
WHERE
    MEDICAO.Id_Parcela  = ARVORE.Id_Parcela AND
    MEDICAO.NumMedicao  = ARVORE.NumMedicao AND
    DataMedicao        >= #JAN/01/1995#     AND
    DataMedicao        <= #DEC/12/1995# ;
```

Outra clausula útil é a `GROUP BY` que nos permite aplicar a ação do `SELECT` a grupos de registros. Poderíamos, por exemplo, pedir que as funções agregadas da sentença `4.43` fossem aplicadas às árvores medidas em 1995, mas desta vez agrupadas por parcela e data de medição. A sentença SQL ficaria assim:

```
SELECT                                                                    4.44
    MEDICAO.Id_Parcela  AS [Parcela],
    MEDICAO.DataMedicao AS [Medição],
    Count(DAP) AS [Número de CAPs],
    Min(DAP)   AS [CAP mínimo],
    Avg(DAP)   AS [Média dos CAPs],
    Max(DAP)   AS [CAP máximo],
    StDev(DAP) AS [Desvio dos CAPs],
    Count(ALT) AS [Número de Alturas],
    Min(ALT)   AS [Altura mínima],
    Avg(ALT)   AS [Média das Alturas],
    Max(ALT)   AS [Altura máxima],
    StDev(ALT) AS [Desvio das Alturas]
FROM
    MEDICAO, ARVORE
WHERE
    MEDICAO.Id_Parcela   = ARVORE.Id_Parcela AND
    MEDICAO.NumMedicao   = ARVORE.NumMedicao AND
    MEDICAO.DataMedicao >= #JAN/01/1995#     AND
    MEDICAO.DataMedicao <= #DEC/12/1995#
GROUP BY
    MEDICAO.Id_Parcela, MEDICAO.DataMedicao ;
```

Para uma maior precisão nas definições, nota-se na sentença `4.44` a cuidadosa identificação dos campos através do uso da sintaxe *`Tabela.Campo`*. Esse cuidado deve ser tomado sempre que existir um campo com o mesmo nome nas tabelas vinculadas. Observa-se também a inclusão dos campos usados para agrupar os resultados logo no início da sentença `SELECT`.

Um outro exemplo interessante envolve o cálculo do CAP médio de todas as medições por parcela. Sabendo-se que foram feitas cinco medições seqüenciais por parcela, é de se esperar

que esses CAPs sejam crescentes conforme as medições se sucederam em cada parcela. Uma forma de checar se isto realmente aconteceu é através do seguinte comando:

```
SELECT                                                                    4.45
    MEDICAO.Id_Parcela   AS [Parcela],
    MEDICAO.DataMedicao  AS [Medição],
    Count(ARVORE.DAP)    AS [Número de árvores],
    Avg(ARVORE.DAP)      AS [CAP médio]
FROM
    MEDICAO, ARVORE
WHERE
    MEDICAO.Id_Parcela = ARVORE.Id_Parcela AND
    MEDICAO.NumMedicao = ARVORE.NumMedicao
GROUP BY
    MEDICAO.Id_Parcela, MEDICAO.DataMedicao
ORDER BY
    MEDICAO.Id_Parcela, MEDICAO.DataMedicao ;
```

Para terminarmos esta introdução, apresentamos mais alguns exemplos de cálculos simples com sentenças SQL, envolvendo dados do cadastro, do inventário e ambos.

Cálculo do DAP médio das medições realizadas na parcela 3

```
SELECT                                                                    4.46
    MEDICAO.DataMedicao       AS [Medições na parcela 3],
    Count(ARVORE.DAP)         AS [Número de árvores],
    Avg(ARVORE.DAP)           AS [CAP médio],
    Avg(ARVORE.DAP/3.1415927) AS [DAP médio]
FROM
    MEDICAO, ARVORE
WHERE
    MEDICAO.Id_Parcela = ARVORE.Id_Parcela AND
    MEDICAO.NumMedicao = ARVORE.NumMedicao AND
    MEDICAO.Id_Parcela = 3
GROUP BY
    MEDICAO.Id_Parcela, MEDICAO.DataMedicao
ORDER BY
    MEDICAO.Id_Parcela, MEDICAO.DataMedicao ;
```

Cálculo do DAP médio das medições realizadas em 1995 por tipo de solo

```
SELECT                                                                    4.47
    SOLO.Dsc_Solo             AS [Solo],
    Avg(ARVORE.DAP/3.1415927) AS [DAP médio medido em 1995]
FROM
    SOLO, TALHAO, INVECADA, MEDICAO, ARVORE
WHERE
    SOLO.Cod_Solo        = TALHAO.Cod_Solo    AND
    TALHAO.Id_Talhao     = INVECADA.Id_Talhao AND
    INVECADA.Id_Parcela  = MEDICAO.Id_Parcela AND
    MEDICAO.ID_Parcela   = ARVORE.Id_Parcela  AND
    MEDICAO.NumMedicao   = ARVORE.NumMedicao  AND
    MEDICAO.DataMedicao >= #JAN/01/1995#      AND
    MEDICAO.DataMedicao <= #DEC/31/1995#
GROUP BY
    SOLO.Dsc_Solo ;
```

Observe que a tabela MEDICAO não é relacionada diretamente com a tabela INVECADA e mesmo assim ligamos as duas nesta sentença. Isso pode ser feito. Na verdade estamos fazendo operações sobre a tabela ARVORE e MEDICAO. Mas não queremos obter as médias por parcela, e sim pelo tipo de solo de cada parcela. As ligações que fazemos da tabela MEDICAO com a tabela SOLO seguem o caminho mais curto possível para obter a informação desejada.

Cálculo em três etapas da Área Basal de parcelas de inventário

1. Primeiramente, criamos uma tabela TEMPORARIA contendo parcelas (Parcela), medições (Medicao), número da árvore medida (Arvore) e área basal da árvore (ABarv).

```
SELECT                                                                    4.48
    ARVORE.Id_Parcela  AS [Parcela],
    ARVORE.NumMedicao  AS [Medicao],
    ARVORE.NumArvore   AS [Arvore],
    Sum( (ARVORE.DAP * ARVORE.DAP) / (4*31415.927) ) AS [ABarv]
INTO
    TEMPORARIA
FROM
    ARVORE
GROUP BY
    ARVORE.Id_Parcela,
    ARVORE.NumMedicao,
    ARVORE.NumArvore
ORDER BY
    ARVORE.Id_Parcela,
    ARVORE.NumMedicao,
    ARVORE.NumArvore ;
```

Para ver o resultado podemos usar a seguinte sentença SQL:

```
SELECT * FROM TEMPORARIA                                                  4.49
    ORDER BY Parcela, Medicao, Arvore ;
```

2. Agora, somamos ABarv por parcela para obter a área basal da parcela, e multiplicamos ABarv por 10000/(área da parcela) para refletir a área basal por hectare. O SQL abaixo mostra os resultados apenas para as parcelas de número igual ou menor que 10.

```
SELECT                                                                    4.50
    TEMPORARIA.Parcela,
    TEMPORARIA.Medicao,
    Sum(TEMPORARIA.ABarv) AS [Área Basal da Parcela],
    Sum(TEMPORARIA.ABarv)*10000/PARCELA.Area AS [Área Basal por ha]
FROM
    TEMPORARIA, PARCELA
WHERE
    TEMPORARIA.Parcela  = PARCELA.Id_Parcela AND
    TEMPORARIA.Parcela <= 10
GROUP BY
    TEMPORARIA.Parcela,
    TEMPORARIA.Medicao,
    PARCELA.Area ;
```

3. Finalmente, apagamos da base a tabela TEMPORARIA.

```
DROP TABLE TEMPORARIA ;                                                    4.51
```

Consistência do CAP medido em 1995 (procura CAPs menores do que no ano anterior)

```
SELECT                                                                     4.52
    A1995.Id_Parcela,
    A1995.NumArvore,
    A1995.NumVara,
    A1994.NumMedicao AS [Medição 1994],
    A1994.DAP        AS [DAP em 1994],
    A1995.NumMedicao AS [Medição 1995],
    A1995.DAP        AS [DAP em 1995]
FROM
    ARVORE  A1995, MEDICAO M1995,
    ARVORE  A1994, MEDICAO M1994
WHERE
    A1995.Id_Parcela        = M1995.Id_Parcela AND
    A1995.NumMedicao        = M1995.NumMedicao AND
    Year(M1995.DataMedicao) = 1995             AND
    A1994.Id_Parcela        = M1994.Id_Parcela AND
    A1994.NumMedicao        = M1994.NumMedicao AND
    Year(M1994.DataMedicao) = 1994             AND
    A1995.Id_Parcela        = A1994.Id_Parcela AND
    A1995.NumArvore         = A1994.NumArvore  AND
    A1995.NumVara           = A1994.NumVara    AND
    A1995.DAP              <= A1994.DAP ;
```

Repare que utilizamos as tabelas `ARVORE` e `MEDICAO` duas vezes. É como se tivéssemos dois conjuntos de dados, um composto pelas tabelas `ARVORE` e `MEDICAO` ligadas e filtradas para o ano de 1995, outro para o ano de 1994. E ligamos esses dois conjuntos de dados fazendo a parcela, a árvore e a vara serem iguais.

Identificação do último ciclo de cada talhão

```
SELECT                                                                     4.53
    T.Cod_Fazenda,
    T.Cod_Talhao,
    C.Ciclo,
    C.Cod_MatGen,
    C.DataPlantio,
    C.DensPlantio
FROM
    TALHAO T, CICLO C
WHERE
    T.Id_Talhao   = C.Id_Talhao AND
    C.DataPlantio = (
    SELECT Max(M.DataPlantio)
        FROM  CICLO M
        WHERE M.Id_Talhao = T.Id_Talhao );
```

Aqui o recurso usado é chamado de *sub-select*. O resultado da sentença dentro do parêntesis é a maior data de plantio de um determinado talhão. Mais uma vez usamos o artifício de criar dois conjuntos de dados com uma mesma tabela. Neste caso, chamamos a tabela `CICLO` uma vez de "`C`", e a outra vez de "`M`".

Idade da rotação atual por talhão

```
SELECT                                                                    4.54
    T.Cod_Fazenda,
    T.Cod_Talhao,
    R.Ciclo,
    R.Rotacao,
    R.DataInicio,
    2000 - Year(R.DataInicio) AS [Idade]
FROM
    TALHAO T, ROTACAO R
WHERE
    T.Id_Talhao  = R.Id_Talhao AND
    R.DataInicio = (
    SELECT Max(M.DataInicio)
        FROM  ROTACAO M
        WHERE M.Id_Talhao = T.Id_Talhao)
ORDER BY 6 ;
```

Observe que a última linha pede para que o resultado seja ordenada pela sexta coluna, que expressa o resultado de uma fórmula.

Parcelas que não foram medidas em 1995:

```
SELECT                                                                    4.55
    Id_Parcela
FROM
    PARCELA
WHERE
    Id_Parcela NOT IN (
    SELECT Id_Parcela
        FROM MEDICAO
        WHERE Year(DataMedicao)= 1995 );
```

O artifício do *sub-select* também pode retornar um conjunto de dados. Na sentença 4.56 estamos verificando se uma determinada parcela não pertence ao conjunto de parcelas medidas em 1995.

Detectando problemas de medição (atribuindo DAP ou Altura a árvore morta ou falha)

```
SELECT                                                                    4.57
    ARVORE.Id_Parcela,
    ARVORE.NumMedicao,
    ARVORE.NumArvore,
    ARVORE.NumVara,
    ARVORE.DAP,
    ARVORE.ALT,
    NOTA.Dsc_Nota
FROM
    ARVORE, NOTA
WHERE
    ARVORE.Cod_Nota = NOTA.Cod_Nota AND
    ARVORE.Cod_Nota IN (4,15)       AND
    (ARVORE.DAP > 0 OR ARVORE.ALT > 0);
```

Mais uma vez utilizamos na cláusula `WHERE` o recurso de testar se um determinado valor pertence ou não a um conjunto de dados. Este conjunto de dados pode ser o resultado de um `SELECT` como na sentença `4.57`, ou um conjunto fixo de dados como na sentença `4.72`.

Detectando problemas no banco (talhões com área muito maior que a média)

```
SELECT                                                                    4.58
    FAZENDA.Dsc_Fazenda AS [Fazenda],
    TALHAO.Cod_Talhao   AS [Talhão],
    TALHAO.Area         AS [Área do talhão],
    (SELECT Avg(Area) FROM TALHAO) AS [Área média dos talhões]
FROM
    FAZENDA, TALHAO
WHERE
    FAZENDA.Cod_Fazenda = TALHAO.Cod_Fazenda          AND
    TALHAO.Area > (1.5 * (SELECT Avg(Area) FROM TALHAO));
```

Este caso é um exemplo de *sub-select* dentro de `SELECT` para retornar uma coluna na tabela de resultados.

Detectando problemas de medição (parcelas com CAP médio maior que 50 em 1995)

```
SELECT                                                                    4.59
    ARVORE.Id_Parcela AS [Parcela],
    ARVORE.NumMedicao AS [Medição],
    Avg(ARVORE.DAP)   AS [CAP Médio]
FROM
    ARVORE, MEDICAO
WHERE
    ARVORE.Id_Parcela = MEDICAO.Id_Parcela AND
    ARVORE.NumMedicao = MEDICAO.NumMedicao AND
    Year(MEDICAO.DataMedicao) = 1995
GROUP BY
    ARVORE.Id_Parcela,
    ARVORE.NumMedicao
HAVING
    Avg(DAP) > 50 ;
```

As funções agregadas pode sem usadas para filtrar os resultados numa cláusula `HAVING`. A condição que colocamos na cláusula `HAVING` é diferente das condições da cláusula `WHERE`. Uma condição colocada na cláusula `WHERE` filtra os dados que participam dos cálculos, mas a condição da cláusula `HAVING` filtra apenas o resultado a ser mostrado.

O objetivo deste curso foi introduzir os principais conceitos da abordagem relacional para a gestão da informação com base em uma proposta de relacionamento de dados florestais. Para isto fomos ilustrando o uso da linguagem SQL através do processo de criação, alimentação e consulta de uma proposta inicial de modelagem dos dados florestais. É muito grande o número de exercícios que poderiam ainda ser elaborados para explorar por completo o potencial da linguagem SQL e da estrutura lógica apresentada. Esperamos, entretanto, ter passado ao leitor uma boa e clara noção das principais características e vantagens que justificam a utilização de bases de dados relacionais. Para finalizar, apresentamos nas próximas páginas a sintaxe dos principais comandos usados nesta seção.

```
SELECT [atributo]
   { * | tabela.* |
   [tabela.]campo1 [AS alias1]
   [, [tabela.]campo2 [AS alias2]
   [, ...]]}
FROM expressãodetabela [, ...] [IN bancodedadosexterno][WHERE... ][GROUP
BY... ][HAVING... ][ORDER BY... ]
```

onde:

atributo	Um dos seguintes atributos: DISTINCTROW, DISTINCT, ALL, ou TOP. Utilize o atributo para restringir o número de registros retornados. Se nenhum for especificado, o padrão será ALL.
*	Especifica que todos os campos da tabela ou tabelas especificadas estão selecionados.
tabela	Nome da tabela contendo os campos a partir dos quais os registros são selecionados.

campo Nomes dos campos contendo os dados de interesse. Quando listados mais de um campo, serão recuperados na ordem listada.

alias Nomes a serem utilizados como cabeçalhos de coluna em lugar dos nomes originais da coluna na tabela.

Expressãodetabela Nome da tabela ou tabelas contendo os dados de interesse.

Bancodedadosexterno Nome do banco de dados que contém as tabelas em *expressãodetabela*, se não estiverem no banco de dados atual.

```
[TABLE] consulta1
  UNION [ALL] [TABLE] consulta2
  [UNION [ALL] [TABLE] consultan [ ... ]]
```

onde:

consulta Instrução SELECT, o nome de uma consulta armazenada ou o nome de uma tabela armazenada precedida da palavra-chave TABLE.

```
FROM tabela1
  INNER JOIN tabela2 ON tabela1.campo1 opercomp tabela2.campo2
```

onde:

tabela Nomes das tabelas das quais os registros são combinados.

campo **N**omes dos campos que são associados. Se não forem numéricos, os campos deverão ser do mesmo tipo de dados e conter o mesmo tipo de dados, mas não precisarão ter o mesmo nome.

opercomp Qualquer operador de comparação relacional: "=," "<," ">," "<=," ">=," ou "<>."

BIBLIOGRAFIA

ALDER, D. (1995) Growth modeling for mixed tropical forests. Oxford: Oxford Forestry Institute. Tropical Forestry Papers, 30. 231 p.

APPLETON D. S. (1983) Data Driven Prototyping. Datamation. 29(11): 259-268.

CODD, E.F. (1970) A relational Model of Data for Large Shared Data Banks. Communications of the Association for Computing Machinery (CACM) 13 No 6. Junho de 1970.

CODD, E.F. (1982) Relational Database: A practical foundation for productivity. Communications of the Association for Computing Machinery (CACM) 25 No 2. Fevereiro de 1982.

DATE, C. J e DARWEN, H.(1993) A guide to the SQL standard (3rd edition). Addison and Wesley. Reading. Massachussets, 830 p.

DURDING, B. M., Curtis A Becker e John D COULD (1977) Data Organization. Human Factors, 19(1):1-14.

FALCÃO, A.O.C.A. (1998) Estruturação e Implementação de Uma Base de Dados de Informação Biométrica Florestal. Dissertação de Mestrado. ISA - Universidade Técnica de Lisboa. 130 p.

FLEMING e VON HALLE (1990) An Overview of Logical Modeling. Data Resources Management 1(1):5-15

GREENSPUN, P. (1999) SQL for Web Nerds. [Documento publicado na World Wide Web. http://www.photo.net/sql/index.html].

JENNINGS, Roger (1994) Database Developpers Guide With Visual Basic 3.0. SAMS. Indianapolis. 1133 p.

PELKKI, Mathew H. (1992) Developing Conceptual Information System Models For Natural Resources. Ph.D. Thesis. University of Minnesota. 215 p.

RODRIGUEZ, L.C.E. (1998) Técnicas quantitativas para a gestão de florestas plantadas. Tese de Livre-Docência. ESALQ - Universidade de São Paulo.120 p.