Capítulo 3

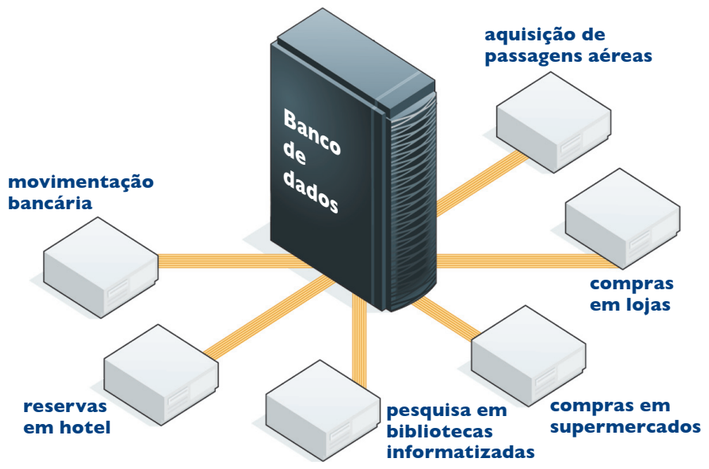
Banco de dados

- Evolução dos sistemas de computação
- Conceitos e terminologia
- Abordagem relacional
- Administração e gerenciamento

Segundo Ramez Elmasri, autor de vários livros sobre informática, o primeiro Sistema Gerenciador de Banco de Dados (SGBD) surgiu no final de 1960, com base nos primitivos sistemas de arquivos disponíveis na época, incapazes de controlar o acesso concorrente por vários usuários ou processos. Assim, os SGBDs evoluíram de sistemas de arquivos para armazenamento em disco, quando foram criadas novas estruturas de dados com o objetivo de armazenar informações. Os SGBDs evoluíram ao longo do tempo, com a introdução de diferentes formas de representação, ou modelos de dados, para descrever a estrutura das informações neles contidas.

Os bancos de dados se tornaram um componente essencial e indispensável em nosso dia a dia. Geralmente não nos damos conta disso, mas sem eles não conseguiríamos mais fazer atividades corriqueiras como depósitos, saques ou quaisquer outras formas de movimentação bancária, reservas em hotel, pesquisa em bibliotecas informatizadas, compras em supermercados ou lojas, aquisição de passagens aéreas. Tais atividades são exemplos clássicos de aplicações que utilizam bancos de dados para manipular textos, valores, imagens etc (figura 37).

Figura 37
Banco de dados.



3.1. Evolução dos sistemas de computação

Os últimos anos têm sido pródigos, em termos da tremenda evolução observada nos sistemas computacionais. O avanço nas diversas versões dos sistemas operacionais permite que haja cada vez mais e mais recursos disponíveis e novas metodologias de acesso, que ampliam a velocidade e a forma de usá-los. Entre as novidades estão os bancos de dados pós-relacionais, também chamados de bancos de dados orientados a objetos. Tudo isso tem um grande impacto, também, na construção de sistemas de aplicação. Vale, portanto, analisar as metodologias empregadas na implantação de sistemas comerciais baseados em computadores.

3.1.1. Abordagem tradicional

Segundo o professor e pesquisador Abraham Silberschatz (1999), a **abordagem tradicional** é baseada em técnicas não estruturadas, utilizando a intuição natural do analista de sistemas, bem como em sua capacidade criativa e vivência no ambiente do sistema, ou seja, em seu conhecimento de negócios. A principal característica dessa abordagem é a orientação intuitiva desses profissionais para desenhar o sistema, com base em procedimentos executados e descritos pelo usuário em sua forma pura, sem orientação a dados e otimizações organizacionais. Nesse período utiliza-se uma mistura de linguagens da terceira geração e de linguagens procedurais, com necessidade de declaração de dados e identificação física da localização da informação. Essa abordagem não prioriza a integração da informação, mas sim a solução de problemas setorizados, ou seja, pode gerar retrabalho, pois muitas vezes haverá os mesmos dados em diversos sistemas na organização.

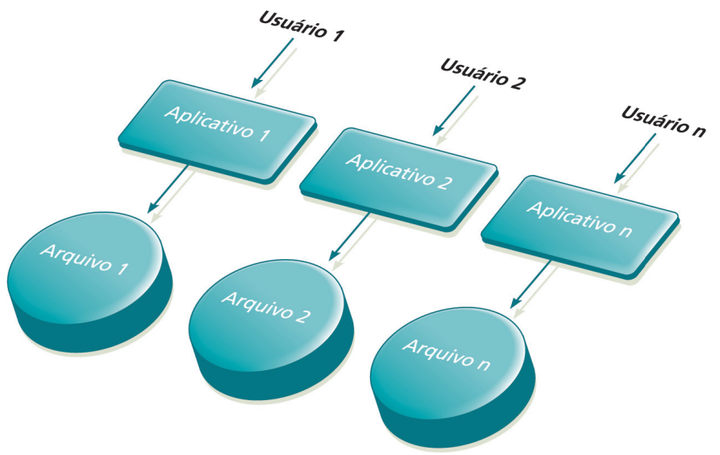
As principais características da abordagem tradicional são: cada aplicação tem seus próprios arquivos; os dados são repetitivos; inconsistência; subordinação de programas a arquivos; manutenção difícil e cara; o analista é o "dono" do sistema; falta de integridade e de segurança.

Assim, sob a abordagem tradicional desenvolvem-se sistemas e aplicações com arquivos de propriedade da aplicação e de seus programas, em função da utilização de linguagens de terceira geração, sem se levar em consideração a repetição dos dados em outros sistemas.

Como consequência, temos a redundância de informações, a qual provoca total incoerência e muitas vezes inconsistência de dados. Os sistemas são elaborados pensando-se na distribuição física dos arquivos (por exemplo, em que pasta ou diretório esse arquivo está fisicamente no HD), criando-se, assim, uma dependência dos arquivos pelos sistemas. Segundo Elmasri (1999), isso significa que qualquer alteração na base de dados implica a adaptação de grande quantidade de programas e, portanto, significa enorme esforço de programação. É bom frisar que, sempre que houver grandes alterações, temos uma forte tendência de esquecer alguns procedimentos, ao realizar alguma modificação, o que pode resultar em erros que muitas vezes demoramos para perceber, pois o sistema poderá continuar gravando as informações na base de dados anterior.

Caso haja muitos programas desenvolvidos dessa maneira, o sistema de tecnologia da informação da organização é ineficiente, pois se trata de uma abordagem antiga e inadequada ao atual estágio de desenvolvimento de sistemas. Esse tipo de abordagem pode ser visualizada da seguinte forma: cada usuário utiliza um aplicativo (ou um conjunto de aplicativos), o qual, por sua vez, usa um arquivo de dados que muitas vezes não é acessado por outro usuário, como sugere a ilustração a seguir (figura 38).

Figura 38
A abordagem tradicional: para cada usuário, um ou mais aplicativos.



3.1.2. Abordagem de sistemas integrados

Elmasri (1999) diz ainda que, por causa das deficiências da abordagem anterior, criou-se, por volta de 1980, uma nova, a de **sistemas integrados**, a qual pretendia resolver as questões de redundância e integridade de informações, trazendo uma visão corporativa de sistemas. Ela se vale da percepção de que existem conjuntos de arquivos de dados de interesse comum às várias áreas de uma organização, ou seja, de que não precisamos redigitar informações que já foram digitadas por outros usuários de outros departamentos ou usuários de outros sistemas. A integração entre os sistemas mantém um grau acentuado de unicidade da informação dentro dos sistemas da organização. Assim, tal abordagem eliminou a redundância de dados que caracteriza a anterior. Aqui, os sistemas passam a utilizar uma única base de informações, sem a repetição de arquivos por sistema.

As principais características da abordagem de sistemas integrados são: pouco adaptável; as alterações comprometem vários sistemas; aplicações estáticas com o tempo; problemas em um sistema paralisam atividades de outros; programas ainda subordinados a arquivos; visão de usuário inexistente; integridade e segurança são fracas.

Os sistemas desenvolvidos dessa maneira iniciaram a integração no universo corporativo, ficando dependentes uns dos outros por utilizarem a mesma base de dados. Assim, alterações em qualquer arquivo de dados implicavam modificações em programas de mais de um dos sistemas que acessavam a mesma base, em virtude de utilizar linguagens de alta dependência de dados (figura 39). Com o crescimento das aplicações, essa abordagem também se tornou ineficiente. Qualquer alteração na base de dados afetava um número ainda maior de sistemas e programas. Com tantas dificuldades relacionadas ao volume de alterações, as aplicações acabaram por se tornar praticamente estáticas, levando os departamentos de tecnologia da informação à estagnação e a serem considerados ineficientes pelos gestores das organizações.

3.1.3. Abordagem de bancos de dados

Segundo Elmasri (1999), embora haja diferenças até certo ponto significativas entre as duas abordagens apresentadas anteriormente, ambas têm em comum uma característica determinante: a ênfase dada aos processos, ao desenvolver

uma aplicação. É por isso que surgem as dificuldades de manutenção, ocasionadas parcialmente pelo forte acoplamento dos programas aos arquivos. A abordagem de banco de dados, ainda considerada a mais adequada, surgiu da necessidade de desenvolvimento de sistemas cada vez mais complexos e flexíveis, integrando os dados de toda a organização e não mais apenas os sistemas departamentais. Sob esse prisma, a ênfase no desenvolvimento de uma aplicação não é mais sobre os processos, e sim sobre os dados. Podemos dizer que, para a abordagem tradicional e de sistemas integrados, na expressão "processamento de dados" a parte mais importante é o processamento, enquanto na abordagem de banco de dados a palavra fundamental é dados.

Segundo Elmasri (2002), há duas preocupações principais, quando trabalhar com essa abordagem. A primeira, em um nível mais conceitual, consiste na definição de um modelo de dados em termos da organização como um todo, abordando seus diversos setores. É preciso retratar todo o interrelacionamento entre as diversas áreas e departamentos que compõem o negócio, bem como todas as necessidades de informação de cada um desses setores. Somente com base nisso poderemos garantir que as aplicações desenvolvidas individualmente terão o comportamento desejado quanto à integração com outros sistemas e, também, que os dados serão os mais confiáveis.

A segunda preocupação diz respeito à forma de implementação física dos dados, em relação aos programas que irão manipular. Aqui deveremos levar em consideração as linguagens de programação disponíveis e qual delas será adotada, tendo sempre o cuidado de selecionar a melhor para o tipo de sistema computacional que estamos desenvolvendo. É necessário que os programas tenham uma visão de alto nível dos dados, o mais independente possível dos fatores físicos ligados à forma de armazenamento desses dados. Assim, nos Sistemas Gerenciadores de Banco de Dados (SGDB), há uma separação entre o local onde se armazena o banco de dados, bem como a definição física dos dados, e aquele no

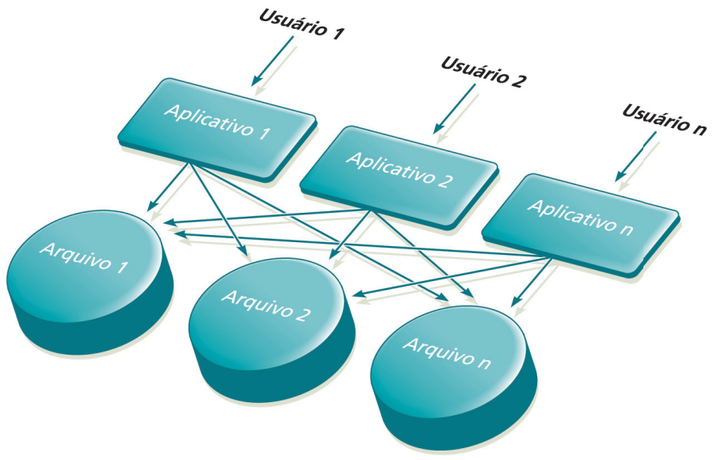


Figura 39
Abordagem de sistemas integrados: dependência.

qual eles são realmente utilizados. Daí se originam, inclusive, os conceitos de Linguagem de Definição dos Dados e de Linguagem de Manipulação dos Dados. Passa a existir, assim, um repositório onde são armazenados esses elementos de mais baixo nível, de forma que fiquem externos ao código do sistema. Esse repositório apresenta-se na forma de um dicionário de dados com aparência e transparência irrelevantes para o analista de sistemas e os programadores.

VANTAGENS

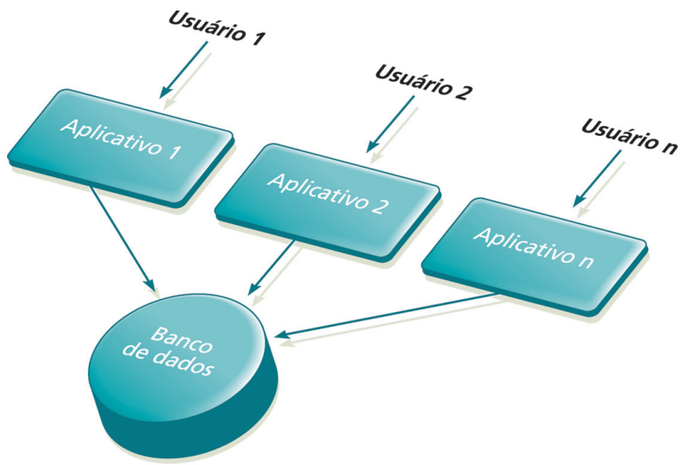
- Desenvolvimento flexível e produtivo
- Integração dos dados em nível empresarial
- Transparência dos dados às aplicações
- Maior controle sobre a integridade dos dados
- Maiores controles de segurança dos dados

REQUISITOS

- Aquisição e utilização de um SGBD
- Utilização de dicionário de dados
- Centralização da definição de dados
- Centralização do projeto das bases de dados

Também devemos avaliar com muita atenção qual arquitetura de hardware e software será a mais adequada ao nosso projeto. A abordagem de banco de dados está ilustrada na figura 40. Trabalhar com ela, no desenvolvimento de sistemas, exige dos profissionais de tecnologia da informação comprometimento com alguns princípios.

Figura 40
Abordagem de banco de dados.



3.2. Conceitos e terminologia

Para trabalhar essa abordagem é preciso conhecer os conceitos e as terminologias relacionadas a banco de dados, justamente os temas abordados neste capítulo.

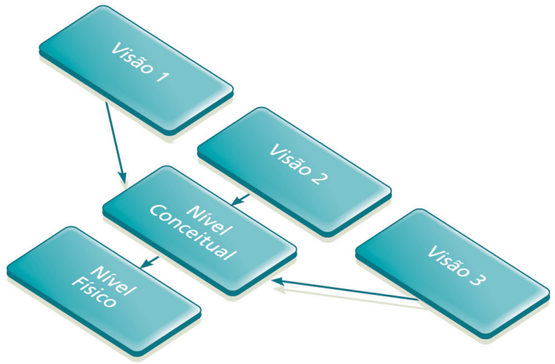
3.2.1. Abstração de dados

De acordo com Elmasri (2002), a abstração de dados é a capacidade de modelar as características, variáveis e constantes, de forma a desprezar os dados que não interessam (veja quadro *Os níveis de abstração*). Um SGBD é composto por uma coleção de arquivos interrelacionados e de um conjunto de programas que permitem aos usuários acessar e modificar os arquivos. Sua proposta maior é prover os usuários de uma visão abstrata dos dados. Ou seja, o sistema omite alguns detalhes de como as informações são armazenadas e mantidas, mas, para que possa ser utilizado em projetos bastante complexos, esses dados devem ser recuperados de maneira eficiente. Como nem sempre os usuários de bancos de dados são treinados em informática, a complexidade fica encapsulada em diversos níveis de abstração, para simplificar a interação desses usuários com o sistema (DATE, 2000). Há três níveis de abstração e seu interrelacionamento é ilustrado na figura 41.

OS NÍVEIS DE ABSTRAÇÃO

- **Nível físico:** é o mais baixo, no qual se descreve como os dados são armazenados, onde essas complexas estruturas são descritas detalhadamente. Assim, os registros – de um cliente, de um fornecedor, de uma conta bancária – são descritos como um bloco de posições consecutivas de armazenamento, como por exemplo string ou byte.
- **Nível conceitual:** é o próximo nível de abstração, em que se descreve quais dados são armazenados e as relações existentes entre eles. Aqui, o banco de dados é descrito em termos de um pequeno número de estruturas simples. É utilizado pelos administradores de banco de dados, que podem decidir quais informações devem ser mantidas. No nível conceitual, os registros – de um cliente, de um fornecedor, de uma conta bancária – se interrelacionam.
- **Nível visual:** é o mais alto de abstração e descreve apenas a parte do banco de dados. Muitos usuários não estão interessados em todas as informações existentes. Sua definição serve para simplificar a interação deles com o sistema, que pode fornecer várias visões diferentes para o mesmo banco de dados. Por exemplo: cada categoria de funcionário pode visualizar determinado grupo de informações.

Figura 41
Interrelacionamento entre os níveis de abstração.



Henry Korth é autor de vários livros clássicos da informática.

3.2.2. Instâncias e esquemas

Os bancos de dados se alteram, ao longo do tempo, à medida que informações são inseridas ou deletadas. O conjunto de dados armazenados em determinado momento é conhecido como instância do banco de dados. Já o projeto do banco de dados é chamado de esquema de banco de dados (DATE, 2000).

Segundo **Korth** (1995) o conceito de um esquema de banco de dados corresponde à noção de definição de tipo na linguagem de programação. Uma variável de um dado tipo tem um valor particular em um determinado instante de tempo. Assim, o conceito de valor de uma variável na linguagem de programação corresponde ao conceito de uma instância de um esquema de banco de dados. Ainda de acordo com Korth (1995), os sistemas de bancos de dados possuem diversos esquemas, subdivididos de acordo com os níveis de abstração descritos anteriormente. No nível mais baixo está o esquema físico, no intermediário o esquema conceitual e no mais alto, um subesquema. Em geral, os sistemas de bancos de dados suportam um esquema físico, um esquema conceitual e diversos subesquemas.

• Independência física: é a habilidade de modificar o esquema físico sem que seja preciso modificar os programas. Alterações no nível físico podem ser necessárias somente para uma ocasional melhoria de performance.
• Independência lógica: é a possibilidade de modificar o esquema conceitual sem que isso demande modificação nos programas. É necessária apenas quando a estrutura lógica do banco de dados é alterada. Por exemplo, se forem criadas mais colunas em uma determinada tabela do banco de dados. A independência lógica é mais difícil de alcançar do que a independência física, porém os programas são bastante dependentes da estrutura lógica dos dados que acessam

3.2.3. Independência de dados

Independência de dados é a habilidade de modificar a definição de um esquema em um nível, sem afetar sua definição de esquema em um outro nível, mais alto. Essa independência constitui o principal objetivo a ser alcançado no desenvolvimento de um banco de dados, pois permitirá a expansão das atividades da empresa, facilitando o desenvolvimento de novos sistemas. Tal independência consiste na capacidade de favorecer que haja evolução na descrição de dados, como a criação de uma nova estrutura lógica decorrente de uma nova aplicação, ou inclusão de um dado novo numa estrutura existente, sem que os sistemas ou aplicações (em última análise, os programas) tenham de ser alterados. Essa capacidade deve se estender aos casos de mudança na organização dos arquivos, em consequência de degradação e perda de eficiência (DATE, 2000). De acordo com esse conceito, os programas interagem com a visão lógica do banco de dados. A localização física do dado é totalmente transparente.

3.2.4. Linguagem de definição de dados

Um esquema de banco de dados é especificado por um conjunto de definições, por meio da chamada linguagem de definição de dados, ou DDL (do inglês, Data Definition Language). A compilação de comandos DDL é um conjunto de tabelas armazenadas em um arquivo chamado dicionário (ou diretório) de dados. Trata-se de um arquivo que contém metadados (dados sobre dados) e é sempre consultado antes que haja qualquer leitura ou modificação pelo banco de dados. Os comandos DDL servem para definir esquemas, remover relações, criar índices e modificar esquemas de relação.

3.2.5. Linguagem de manipulação de dados

A DML (do inglês Data Manipulation Language) decorre do fato de os níveis de abstração não se aplicarem somente à definição ou à estruturação de dados, mas também à sua manipulação. Isso tem uma série de significados, no âmbito do banco de dados: como **recuperamos a informação armazenada**, como inserimos novas informações, como deletamos as informações, como modificar dados armazenados.

Uma consulta (também chamada de query) é um comando que requisita o resgate de uma informação, com base na álgebra relacional e no cálculo relacional de tupla.

3.2.6. Usuários de banco de dados

Já sabemos que o objetivo central de um banco de dados é prover uma solução para armazenamento e busca de informações por seus usuários. Estes, os que irão interagir com o banco de dados para realizar essas operações, podem ser classificados em cinco tipos, de acordo com o tipo de interação (DALTON, 2008). Confira quais são:

- **Profissionais de sistemas informatizados:** aqueles que desenvolvem aplicativos desenvolvidos para realizar a interação com os bancos de dados.
- **Programadores ou desenvolvedores:** são os que fazem a interação com os sistemas, utilizando-se das DML, que são incluídas nos programas, denominados programas de aplicação.
- **Usuários avançados:** interagem com o sistema sem escrever programas. Como têm conhecimentos de linguagem de consulta, realizam chamadas específicas nessa linguagem para atender a suas demandas.
- **Usuários leigos:** só interagem com os sistemas desenvolvidos pelos programadores.
- **Administradores de banco de dados (DBA, do inglês Data Base Administrator):** são os responsáveis pela criação, manutenção e bom funcionamento do banco de dados. Avaliam sua performance com frequência e, sempre que necessário, tratam de aprimorá-la.

3.3. Abordagem relacional

Em 1969, o pesquisador da IBM Edgar Frank Codd, já mencionado no capítulo 2, começou a pesquisar uma maneira melhor de organizar dados e introduziu o conceito de banco de dados relacional, que é organizado em tabelas simples. Conforme Elmasri (2002), Codd criou um modelo que permite aos projetistas dividir suas bases de dados em tabelas separadas, mas relacionadas, de modo

a otimizar a eficiência da execução, mantendo a mesma aparência externa do banco de dados original para os usuários. Por isso é considerado o pai do banco de dados relacional. A maioria dos bancos de dados é relacional. Baseia-se no princípio de que as informações em uma base de dados podem ser consideradas como relações matemáticas e são representadas de maneira uniforme. Os objetos manipulados pelos usuários são as linhas (tuplas, na terminologia original) e as colunas das tabelas. Para fazer isso o usuário conta com um conjunto de operadores e funções de alto nível, denominados álgebra relacional.

3.3.1. Características principais

As principais características da abordagem relacional baseiam-se em uma série de fatores (KORTH, 1995), definidos a seguir.

- **Estrutura de dados tabular:** os dados são representados em forma de tabela, que é denominada relação, constituída de linhas ou tuplas, colunas ou atributos e, finalmente, tipos de dados que podem aparecer em uma coluna específica, chamada domínios. Assim, não existe o conceito de item de grupo, em um banco de dados relacional. Todos os itens de uma tabela são elementares.

- **Álgebra relacional:** a manipulação das tabelas é realizada por operadores por meio dos quais podemos acessar dados de diversas maneiras, ou seja, é um conjunto de operações e relações. Cada operação usa uma ou mais relações como seus operandos e produz outra relação como resultado. Dessa forma, a recuperação das informações em um SGBD relacional se faz em conjuntos de registros que comporão uma nova relação.

- **Dicionário de dados ativo e integrado:** um banco de dados que mantém as descrições dos objetos existentes no sistema da empresa deve ser ativo, no sentido de estar sempre disponível para utilização, e integrado, na medida em que englobe todas as informações sobre o sistema, permitindo realizar referências cruzadas nos dados.

- **Consistência automática de campos:** o próprio sistema tem a funcionalidade de validar os dados nos quesitos de tamanho, formato, tipo e integridade.

- **Referência cruzada de dados:** permite relacionar linhas e colunas, ou seja, valores de campos com linhas;

- **Integridade dos dados:** para garantir a qualidade das informações do banco de dados, devemos nos preocupar com quatro categorias de integridade dos dados:
  1. **De entidade:** definimos uma linha como exclusiva de determinada tabela, ou seja, definimos uma **chave primária (Primary Key)** de uma tabela.
  2. **Integridade de domínio:** valida entradas de uma coluna específica, podendo restringir suas relações em linhas, quando estas são incluídas ou excluídas. São as chaves estrangeiras (**Foreign Key**).
  3. **Integridade referencial:** preserva as relações definidas entre tabelas, permitindo consistência em todas elas.
  4. **Integridade definida pelo usuário:** o usuário define suas regras de negócio, quando não se enquadra nas outras categorias de integridade apresentadas.

A chave primária (Primary Key) serve para identificar cada registro numa tabela. Pode ser um atributo ou uma combinação de atributos.

A chave estrangeira (Foreign Key) pode ser uma coluna ou combinação de colunas usada para estabelecer links entre duas tabelas.

3.3.2. Princípios da orientação

O conceito de visões de dados tem a ver com a forma como o usuário os vê. Existe a possibilidade de criar visões de subconjuntos dos dados colocados em um SGBD. Os usuários podem ter acesso a parte do modelo, independentemente da forma como estão contidos no banco de dados. Tomemos como exemplo um banco de dados com clientes e pedidos. Podemos montar uma visão de informações que mostre os clientes com seus respectivos pedidos, sem que isso afete as estruturas do modelo de dados implementado ou apresentar uma visão com informações cadastrais de clientes sem dados financeiros, por exemplo. As visões são formadas conforme a necessidade do usuário. Para definir essas visões é preciso atentar para dois fatores: a transparência de dados (como e onde estão os dados torna-se secundário para o usuário, irrelevante) e os elos implícitos (colunas comuns nas tabelas, sem restrição a tipo de relacionamento).

3.3.3. As doze regras de Edgar F. Codd

As bases da abordagem relacional, como sabemos, foram lançadas em 1970 por Edgar F. Codd. Na época o pesquisador estabeleceu um conjunto de doze regras para um banco de dados realmente relacional. Segundo Date (2000), as normas de Codd, que vamos conhecer a seguir, discutem a fidelidade de um SGBD ao modelo relacional.

Regra número 1 - Representação de valores em tabelas

Toda informação, em um Banco de Dados Relacional, é apresentada em nível lógico por valores em tabelas.
Os nomes das tabelas, colunas e domínios são representados por séries de caracteres que, por sua vez, devem também ser guardadas em tabelas, as quais

BENEFÍCIOS DA ABORDAGEM

- Independência dos dados
- Visões múltiplas de dados
- Melhor comunicação entre o departamento de informática e os usuários
- Redução acentuada do desenvolvimento de aplicações e do tempo gasto em manutenção de sistemas
- Melhoria na segurança de dados
- Mais agilidade gerencial de informações ligadas ao processo decisório da empresa

formam o catálogo do sistema, o dicionário de dados. Portanto, o princípio de que a informação deve estar sob a forma de tabela é extensiva ao dicionário de dados. As tabelas são bidimensionais, o que equivale a dizer que não há colunas repetitivas (Cláusula Occurs, por exemplo, não existe).

Regra número 2 - Acesso garantido

Todo e cada dado num Banco de Dados Relacional tem a garantia de ser logicamente acessível, recorrendo-se a uma combinação do nome da tabela, um valor de chave primária e o nome da coluna.
Isso significa que a ordem das linhas, assim como a das colunas, é irrelevante. Para obter uma informação específica, ou seja, o valor de uma coluna em uma linha de uma tabela, basta informar o nome da tabela, o valor de uma chave primária (no caso, da linha a ser recuperada) e o nome da coluna da tabela em questão. Da mesma forma, para saber quais ocorrências de uma tabela têm um determinado valor em uma coluna, basta informar o nome da tabela e o nome da coluna, que o conteúdo poderá ser comparado. Conclusão: a regra especifica que, em um banco de dados efetivamente relacional, não existe informação inacessível.

Regra número 3 - Tratamento sistemático de valores nulos

Valores nulos são suportados em um SGBD relacional nulo para representar exatamente a informação perdida ou inexistente, inaplicável.
Para mais bem entender essa regra, vale a pena definir o que é valor nulo: é aquele utilizado para representar uma informação perdida ou desconhecida. Conceitualmente, um valor é inexistente. A inexistência de conteúdo em um campo equivale a dizer que seu valor é nulo. Mas atenção: jamais confunda zero(s) ou espaço(s) em branco com strings vazios.

Imagine um pacote de supermercado de papel, desdobrado. Sem tocá-lo ou olhar em seu interior, você não pode saber se está vazio ou contém algum produto. Portanto, nesse instante, para você, o conteúdo do pacote é NULO. É uma informação desconhecida, inexistente. Valores nulos, portanto, são suportados por um SGBD para representar exatamente a informação perdida ou inexistente, inaplicável. Para suportar a integridade de identidade é preciso especificar “não são permitidos valores nulos” em cada coluna da chave primária e em qualquer outra coluna que se considerar como uma restrição de integridade. Se levarmos em conta que um atributo tem de, obrigatoriamente, possuir conteúdo, estaremos especificando que seu valor não pode ser nulo. Assim, o SGBD precisa reconhecer a informação nula para poder restringi-la ou aceitá-la, se necessário.

Regra número 4 - Catálogo relacional ativo baseado no modelo relacional

A descrição do banco de dados é representada em nível lógico, da mesma maneira que seus dados.
Esta regra exige que um SGBD relacional tenha uma mesma linguagem para acesso e definição dos dados no dicionário e para a manipulação dos dados do banco e do dicionário. Por exemplo: um comando para adicionar informações em uma tabela da aplicação deve ser o mesmo para acrescentar informações no dicionário de dados.

Regra número 5 - Sublinguagem detalhada e dados

Um sistema de banco de dados relacional deve ter uma linguagem cujas instruções, com sintaxe bem definida, suportem a definição de dados, a definição de visão de dados, a manipulação de dados, as restrições de integridade, as autorizações e os limites de transação.

Instruções
- **Definição dos dados:** criar ou adicionar tabelas no dicionário de dados.
- **Definição de visão de dados:** fazer operações relacionais de junção, criar visões de partes do modelo de dados.
- **Manipulação de dados:** permitir criar, alterar, consultar e deletar conjuntos de dados.
- **Restrições de integridade:** possibilitar que sua sublinguagem controle as restrições específicas de uma tabela ou de valores aceitáveis para uma determinada coluna.
- **Autorizações:** definir limites de acesso a tabelas por usuário, inclusive em nível de coluna.
- **Limites de transação:** tornar viável a delimitação de uma transação lógica de modificação do banco de dados, o que fornece um alto nível de segurança da integridade de informações no banco de dados.

Regra número 6 - Atualização de visão

Todas as visões de dados, que são teoricamente atualizáveis, são também atualizáveis pelo sistema.
Atualizar significa mais do que simplesmente modificar a informação; abrange também a inclusão e/ou exclusão de dados. Por exemplo, se definimos uma visão de dados como um subconjunto horizontal de uma tabela, deve ser possível adicionar dados a essa visão. Consideremos uma tabela de funcionários e que haja visões de funcionários técnicos, funcionárias secretárias e funcionários administrativos. Atualizar uma dessas visões quer dizer, entre outras possibilidades, adicionar dados para a inclusão de uma secretária ou de um técnico. Outro exemplo seria criar uma visão de uma tabela de médicos por especialidade, digamos cem médicos em uma tabela, dos quais trinta são especializados em pediatria. Ao criarmos a visão Pediatras, uma seleção de linhas, que conterá somente os médicos com esta especialidade, cria-se no dicionário de dados um sinônimo de médico, denominado Pediatra, que tem como condição para esse role (literalmente, papel) o valor da coluna especialidade ser igual a pediatria. Esta visão deve ter a possibilidade de ser deletada. Se comandarmos DELETAR PEDIATRIA ou ALTERAR um pediatra específico, tais atualizações devem ser realizadas diretamente na tabela originária da visão.

Regra número 7 - Atualização de alto nível

No banco relacional, a linguagem de alto nível se aplica não somente à

consulta de dados, mas também à inclusão, alteração e exclusão de dados.
Isso quer dizer que as operações realizadas sobre a base de dados tratam conjuntos de informações (tabelas), não registro por registro. Isso quer dizer que a linguagem de alto nível não é procedural, uma característica das linguagens denominadas de quarta geração.

Regra número 8 - Independência de dados físicos

Os programas de aplicação permanecem inalterados sempre que quaisquer modificações são feitas nas representações de memória ou nos métodos de acesso.
Essa regra nos dá uma diretriz essencial quanto aos programas de aplicação: esses programas lidam somente com dados lógicos. Se houver qualquer modificação quanto ao local de armazenamento físico dos dados ou, ainda, uma mudança qualquer de índices de acesso, o programa de aplicação, logicamente estável, não sofrerá nenhuma paralisação nem precisará de alterações. Por exemplo, se desenvolvermos uma aplicação em um diretório de dados e, posteriormente, em consequência de uma migração ou expansão de hardware, essa base de dados for transferida para um outro diretório de dados, as aplicações desenvolvidas originalmente para tal base não serão afetadas, já que a definição de localização dos dados deve ser externa à linguagem de manipulação de dados.

Regra número 9 - Independência de dados lógicos

Os programas de aplicação também permanecem inalterados quando são feitos nas tabelas, mudanças de qualquer tipo para preservar a informação.
Aqui a regra implica, por exemplo, que a junção de duas tabelas não afeta os programas, assim como a criação de novos campos e as operações de seleção que dividem uma tabela por linhas ou colunas, desde que se preservem as chaves primárias. Se, por exemplo, criarmos um novo relacionamento entre duas entidades ou modificarmos a condição de relacionamento entre elas, em hipótese nenhuma isso afetará as aplicações existentes.

Regra número 10 - Independência de integridade

As restrições de integridade específicas de determinado banco de dados relacional devem ser definíveis na sublinguagem e armazenáveis no catálogo do sistema e não nos programas de aplicação.
Restrições de domínio e regras para validação de alguns ou todos os atributos também devem ser armazenáveis no dicionário de dados. Definições, por exemplo, do tipo de caractere de um campo – se numérico, se inteiro ou de tamanho variável – devem estar especificadas nele. Já quanto ao controle de sua validade (consistência), deve haver a possibilidade de ser realizado pela sublinguagem do SGBD, sem que, para isso, haja necessidade de que o programa de aplicação o controle. Da mesma forma, a exigência da existência de um campo deve ser informada no dicionário e também deve ser administrada por este, ficando os programas de aplicação desobrigados de tais controles.

Regra número 11 - Independência de distribuição

Um SGBD relacional tem independência de distribuição quando sua sublinguagem permite que os programas de aplicação permaneçam inalterados enquanto os dados são distribuídos, tanto na inicialização de um sistema quanto na ocorrência de uma redistribuição.
Se um sistema, por sua implementação, tiver arquivos redistribuídos em meios ou máquinas diferentes, a sublinguagem do SGBD fará o controle e a ligação com os programas de aplicação, sem que seja necessário realizar nenhuma modificação neles.

Regra número 12 - Não subversão

Se um SGBD relacional tem uma linguagem de baixo nível ou recursos de linguagem que permitam processamento em baixo nível, essa linguagem não pode ser usada para subverter ou ignorar as regras de integridade ou restrições expressas na linguagem relacional de alto nível.
Isso quer dizer que, se processarmos registro a registro, isso não poderá implicar a burla de restrições específicas dos objetos do banco de dados. Tal regra pode ser considerada extensiva à utilização de outras linguagens capazes de interagir com o banco de dados. Por exemplo, a utilização de uma linguagem de baixo nível não poderá adicionar um registro sem os atributos definidos, no dicionário, como obrigatórios para uma determinada tabela.

3.3.4. Chaves e índices

O modelo relacional emprega o conceito diferenciado entre chaves e índices.

Um índice é um recurso físico que visa otimizar a recuperação de uma informação por meio do método de acesso. Seu objetivo está relacionado com o desempenho de um sistema, ou seja, concebidos para aumentar a velocidade de recuperação dos dados, os índices devem ser criados para os campos (ou colunas) pesquisados com mais frequência. Como em tudo o que desenvolvemos, é preciso analisar os prós e os contras, já que a criação de índices pode gerar algumas desvantagens, como o tempo que se leva para construí-los; o espaço em disco utilizado para armazená-los; a demora maior para as operações de modificação no banco de dados, pois todas as mudanças têm de ser realizadas nos dados e nos índices. Esses problemas podem ser minimizados ou até mitigados com o uso de regras para a criação de **campos indexados**.

A chave designa o conceito de item de busca, ou seja, um dado que será empregado nas consultas à base de dados. É um conceito lógico da aplicação.

Podemos dizer que um campo é chave se puder ser utilizado para recuperar linhas de uma tabela. Geralmente, todos os campos de uma tabela são chaves, mas o modelo relacional preocupa-se com a definição das principais chaves de uma tabela. Assim, implementa os conceitos de chave primária e chave estrangeira, que se relacionam com duas restrições de integridade determinadas por Codd, quanto ao banco de dados. Vejamos, então, quais são os tipos de chave em um modelo relacional.

• Colunas que devem ser indexadas: chave primária; as que frequentemente são utilizadas em junção (chave estrangeira); as frequentemente pesquisadas em faixas de valores; as geralmente recuperadas de forma classificada.
• Colunas que não devem ser indexadas: as que raramente são referenciadas em uma consulta; as que contêm poucos valores únicos; as definidas com os tipos de dados text, image ou bit; quando o desempenho das atualizações é mais importante que o desempenho das consultas.

1) Chave primária

O conceito de chave primária está ligado à própria concepção do modelo relacional. Os dados estão organizados sob a forma de tabelas bidimensionais, com linhas e colunas, e o princípio nos conduz a termos uma forma de identificar uma única linha da tabela por meio de um identificador único em valor. Trata-se de um conceito fundamental para entendermos o funcionamento de um banco de dados. Quando definimos um campo como chave primária, estamos informando ao banco de dados que não pode existir mais algum registro com o mesmo valor especificado como chave primária, ou seja, os valores das chaves primárias devem ser únicos. Toda tabela relacional deve possuir esse identificador unívoco, denominado chave primária. Assim, uma tabela relacional contém um campo ou conjunto de campos concatenados, permitindo que dela se identifique uma e somente uma ocorrência. Mas em uma mesma tabela podem existir mais de um campo ou coluna com essa propriedade. A estas denominamos chaves candidatas. Entre essas candidatas a chave primária, temos de escolher uma para ser definida e utilizada como tal, deixando as demais como chaves alternativas.

A chave primária também pode ser formada pela combinação de mais de um campo, pois há casos em que não se pode atribuir essa função a um único campo, já que este pode conter dados repetidos. Nessa situação, podemos combinar dois ou mais campos para formar nossa chave primária. Quando a chave primária é composta, devemos ficar atentos ao desempenho das consultas, que é inversamente proporcional ao tamanho da chave primária. Quanto maior o tamanho da chave primária, maior será o tempo de retorno de uma consulta.

2) Chaves candidatas

Uma tabela também pode conter alternativas de identificador único, pois várias colunas ou concatenações diferentes de colunas podem ter essa propriedade e, portanto, são candidatas a chave primária. Mas, como somente uma será escolhida como chave primária, as restantes passam a ser consideradas chaves alternativas.

3) Chaves secundárias

O termo chave sempre se refere a um elemento de busca por meio do qual podemos recuperar uma informação ou um conjunto de informações. E, nas tabelas do banco de dados relacional, há colunas ou conjuntos de colunas concatenadas que nos permitem fazer não uma identificação única, mas de um grupo de linhas de uma tabela. Chamamos tais colunas ou conjuntos de colunas concatenadas de chaves secundárias. É possível que existam N linhas em uma tabela com o mesmo valor de chave secundária.

4) Chaves estrangeiras

Um dos conceitos de importância vital no contexto do modelo relacional é o das chaves estrangeiras. Para mais bem compreender do que se trata, vale analisar as duas tabelas a seguir (Funcionario e Departamento). Criar índices para chaves estrangeiras aumenta a velocidade na execução das junções e também facilita

a realização dos comandos **ORDER BY** e **GROUP BY**. Quando dizemos que duas tabelas possuem colunas comuns, devemos observar que, provavelmente, em uma da tabelas a coluna é uma chave primária. Na outra tabela, a coluna comum será caracterizada, então, como o que chamamos de chave estrangeira. É, na realidade, uma referência lógica de uma tabela à outra.

Conclui-se, então, que havendo uma coluna Ca em uma tabela A e uma coluna Cb, com idêntico domínio, em uma tabela B, a qual foi definida como chave primária de B, então a coluna Ca é uma chave estrangeira em A, já que a tabela A contém uma outra coluna distinta de Ca, que é sua chave primária. Não há limitação para o número de chaves estrangeiras em uma tabela. Veja nas tabelas abaixo exemplos de referência lógica por chave estrangeira.

O comando Order By especifica um ou mais elementos que serão usados para classificar um conjunto de resultados e facilitar a pesquisa. O Group By serve para agrupar dados recuperados conforme critérios pré-definidos.

FUNCIONARIO

Matricula	Nome	Depto
1234	Wilson Oliveira	25
5678	Lucas Sirtori	15
9191	Andréa Oliveira	10
9287	Amanda Cristina	10
9388	Priscila Oliveira	25

DEPARTAMENTO

Numero	Nome
10	Contabilidade
25	Sistemas
15	RH

Na tabela Funcionario temos o atributo Matricula_Funcionario como chave primária e, na tabela Departamento, a chave primária é o atributo Numero_Departamento. Assim, número do departamento em Funcionario é uma chave estrangeira, uma referência lógica que estabelece o relacionamento entre as duas entidades.

3.3.4.1. Regras de integridade

As regras de integridade existem para evitar que uma determinada coluna não tenha uma relação correspondente. Em função dos conceitos de chave primária e chave estrangeira, Codd elaborou as duas regras de integridade de dados do modelo relacional, a de integridade e a de integridade referencial.

Regra de integridade de identidade

Esta restrição se refere aos valores das chaves primárias. Se a chave primária, por definição, identifica uma e somente uma ocorrência de uma tabela, então não poderá ter valor NULO porque nulo não identifica nada, representando apenas a informação desconhecida.

Regra de integridade referencial

Se determinada tabela A tem uma chave estrangeira que é chave primária de uma tabela B, então ela deve:

- **Ser igual a um valor de chave primária existente na tabela B;**
- **Ser totalmente nula. Isso significa que uma ligação lógica está desativada.**

Conclui-se que não podemos ter um valor em uma chave estrangeira de uma tabela e que esse valor não existe como chave primária em alguma ocorrência da tabela referenciada. As regras de integridade do modelo relacional representam a garantia de que as tabelas guardam informações compatíveis. São, portanto, de extrema importância para a confiabilidade das informações do banco de dados.

A manutenção de valor nulo para uma chave estrangeira significa que não existe, para aquela ocorrência, ligação lógica com a tabela referenciada.

Utilizamos a integridade referencial para garantir a integridade das informações entre as tabelas relacionadas em um banco de dados, evitando assim inconsistências e repetições desnecessárias.

3.3.4.2. Regras de conversão do modelo E-R para o modelo relacional

Um modelo Entidade e Relacionamento pode ser convertido para um modelo Relacional de banco de dados. É preciso, porém, levar em conta as seguintes regras para cada elemento do sistema:

- **Entidades:** toda entidade torna-se uma tabela carregando todos os seus atributos. Cada atributo vira um campo da tabela. A chave primária e as chaves candidatas são projetadas para não permitir ocorrências múltiplas nem admitir nulos.

- **Relacionamento 1:N:** a tabela cuja conectividade é N carrega o identificador da tabela cuja conectividade é 1 (chave estrangeira).

- **Relacionamento 1:N:** (envolvendo autorrelacionamento): a chave primária da entidade é incluída na própria entidade como chave estrangeira, gerando uma estrutura de acesso a partir dessa chave estrangeira.

- **Relacionamento 1:1:** as tabelas envolvidas nesse relacionamento carregarão o identificador da outra (uma ou outra ou ambas) conforme a conveniência do projeto (de acordo com o acesso a essas tabelas).

- **Relacionamento 1:1:** (envolvendo autorrelacionamento): chave primária da entidade é incluída na própria entidade (chave estrangeira) e cria-se uma estrutura de acesso para ela.

- **Relacionamento N:N:** o relacionamento torna-se uma tabela, carregando os identificadores das tabelas que ele relaciona e os atributos do relacionamento (se houver).

3.4. Administração e gerenciamento

Um banco de dados (ver definição de Sistema Gerenciador de Banco de Dados – SGDB – no capítulo 2.5.10) é um conjunto de objetos SQL, como tabelas, funções e triggers, e tem de ser bem administrado para que se possa tirar de seus recursos o máximo proveito e com a melhor performance possível para cada tipo de negócio. Por isso há profissionais capacitados especificamente para essa função, conhecidos no mercado como Data Base Administrator (administrador de banco de dados) ou mesmo pela sigla DBA. Tais profissionais têm de conhecer profundamente as ferramentas de administração do banco de dados para utilizá-las de maneira eficiente, pois são eles que desenvolvem e administram estratégias, procedimentos, práticas e planos para disponibilizar documentação, além de compartilhar informações e dados corporativos necessários, no momento certo e às pessoas certas, com integridade e privacidade. Veja quais são as principais funções de um DBA:

• Definir o conteúdo de informações do banco de dados

Por meio de levantamentos de informações, o profissional deve decidir que informações devem ser mantidas no banco de dados da empresa.

• Definir a estrutura de armazenamento e a estratégia de acesso

Aqui o DBA deve indicar como os dados serão armazenados e quais serão os acessos mais frequentes.

• Promover a ligação com os usuários

Isso quer dizer que é sua função garantir a disponibilidade dos dados de que os usuários precisam, preparando-os ou os auxiliando na montagem do nível de visões.

• Definir os controles de segurança e integridade

Ou seja, determinar quem tem acesso a que porções do banco de dados e criar mecanismos que evitem inconsistências na base de dados.

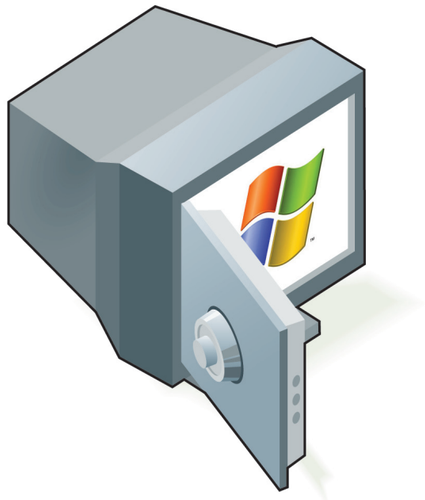
• Definir a estratégia de backup e recuperação

É função do DBA ainda especificar como e quando serão feitos os backup e desenvolver uma estratégia para recuperar informações em caso de danos ao banco de dados.

• Monitorar o desempenho e atender às necessidades de modificações

O DBA deve organizar o sistema de modo a obter o melhor desempenho possível para a empresa.

Figura 42
Segurança em banco de dados.



3.4.1. Segurança em banco de dados

Quando falamos de segurança em banco de dados, precisamos ter em mente um conceito muito simples: o usuário deve acessar somente os dados estritamente necessários para realizar seu trabalho – não podemos lhe fornecer nenhuma outra permissão. Por exemplo, se a função do usuário é apenas realizar consultas, não devemos permitir que possa também alterar, excluir ou inserir dados no sistema, mas tão somente que faça consultas (figura 42).

3.4.1.1. Segurança no sistema operacional

Antes de falar sobre a segurança em banco de dados, vamos abordar o tema em relação ao sistema operacional. Vamos saber, por exemplo, como e porque se deve fazer o logon no Windows ou qualquer outro sistema operacional. E também porque no Windows a conta do usuário é usada como se fosse sua identidade, já que é por meio da conta de logon desse sistema que conseguimos identificar os usuários conectados e carregar configurações personalizadas para cada um deles. Além disso, aprenderemos a criar e a administrar contas de usuários e de grupos de usuários para aumentar a segurança de nossos dados.

É muito importante que estudemos as contas de usuários e grupos de usuários, pois elas formam a base da estrutura de segurança no Windows. Para esse sistema operacional é fundamental identificar o usuário que está tentando acessar determinado recurso, como uma pasta ou impressora compartilhada. Uma vez identificado o usuário, o Windows pode determinar, com base nas permissões de acesso, qual é seu nível de acesso e se ele tem ou não permissão para acessar o recurso em questão. A identificação do usuário é feita por meio do logon, pois, para conectar, ele tem de informar ao sistema qual é sua conta de logon e digitar a respectiva senha. Feito o logon, o usuário está identificado para o Windows.

O conceito de logon está diretamente ligado ao princípio da autenticidade, pois, antes de liberar o acesso a um recurso, o Windows precisa identificar o usuário que está tentando fazer o acesso, ou seja, precisa autentificar o usuário.

3.4.1.1.1. Definição de contas de usuários

Uma conta de usuário é sua identidade para o Windows. Em outras palavras: é a maneira do programa identificar cada usuário. A partir do momento em que é possível identifica-lo por meio do logon, o sistema também consegue manter um ambiente personalizado para ele, bem como um conjunto de permissões que definem o que ele pode e não pode fazer, a que recursos tem ou não acesso e em que nível.

Se você está trabalhando em um computador isoladamente ou em uma pequena rede para a qual não foi definido um domínio e os servidores rodam o Windows 2000 Server ou Windows Server 2003 e o Active Directory, as contas de usuários são gravadas no próprio computador. Assim, cada computador terá sua lista própria de contas de usuário, que são chamadas contas locais de usuário, tradução para local user account. Os usuários de tais contas só podem fazer o logon no computador em que estas foram criadas e acessar os recursos unicamente dessa máquina. As contas locais são criadas em uma base de dados chamada base de dados local de segurança (do inglês local security database). Ao fazer o logon, o Windows compara o nome do usuário e a senha fornecida com os dados da base de segurança local. Se os dados forem reconhecidos, o logon se completa. Caso contrário, é negado e o sistema emite uma mensagem de erro.

Já em redes de grande porte, baseadas em servidores Windows Server 2003, normalmente se criam domínios. Em um domínio existe apenas uma lista de contas de usuários e grupos de usuários, a qual é compartilhada por todos os computadores que o integram. A lista de usuários é mantida nos servidores da rede, nos chamados Controladores de Domínios, ou DCs (do inglês Comain Controllers). Quando um usuário faz o logon, por meio de uma conta da lista, através da rede, o Windows verifica se ele forneceu nome e senha válidos para o domínio e só libera a conexão em caso afirmativo. Contas de domínio são armazenadas na base de segurança dos servidores do domínio em questão, a qual é conhecida como SAM no NT Server 4.0 e como Active Directory no Windows 2000 Server e no Windows Server 2003.

Há dois tipos de conta de usuário disponíveis no seu computador: administrador do computador e limitada. Existe também uma conta de convidado (guest) para usuários que não têm conta configurada no equipamento. Vejamos agora quais são as características de cada tipo de conta.

- **Conta administrador**

  Destina-se ao usuário que pode alterar o sistema, instalar programas e acessar todos os arquivos armazenados. Este, portanto, terá permissão sobre todos os recursos do computador, inclusive as contas de todos os outros usuários. Sua única restrição é alterar o tipo de sua própria conta para conta limitada, a menos que o computador contenha um outro usuário com conta de administrador. Tal restrição visa assegurar que haja sempre pelo menos

PERMISSÕES AO USUÁRIO ADMINISTRADOR

• Criar e excluir contas de usuário no computador.
• Criar senhas para as contas dos outros usuários no computador.
• Alterar nomes, imagens, senhas e tipos de contas dos outros usuários.

um usuário com uma conta de administrador, ou seja alguém com permissão para instalar novos programas, configurar o Windows e alterar as contas de usuários.

- Conta limitada

Como o próprio nome sugere, destina-se ao usuário com restrições de permissão para usar os recursos do computador, como alterar a maioria das configurações ou excluir arquivos importantes. Esse usuário basicamente só pode acessar programas já instalados e alterar a imagem de sua própria conta, além de criar, alterar ou excluir sua própria senha.

3.4.1.2. Permissões para banco de dados

As informações são o principal ativo das corporações. Assim, devemos ter muito cuidado ao definir o acesso a seus bancos de dados, fazendo permissões exclusivamente a pessoas que devem e podem acessá-los e na medida exata de suas necessidades de informação para o trabalho.

Agora que já sabemos um pouco sobre como funciona a segurança em sistemas operacionais, vamos conhecer a estrutura de segurança em **bancos de dados**, que é bem similar, a ponto até de herdar algumas de suas características. Vejamos, primeiramente, alguns dos conceitos de permissões em banco de dados.

Podemos definir, por exemplo, as seguintes permissões para um banco de dados:

- **Create Table (Criar Tabela).**
- **Create View (Criar Consulta).**
- **Create SP (Criar Stored Procedure).**
- **Create Default (Criar Default).**
- **Create Rule (Criar Regra),**
- **Create Function (Criar Função).**
- **Backup DB (Backup do Banco de Dados).**
- **Backup Log (Backup do Log de transações).**

RESTRIÇÕES AO USUÁRIO DE CONTA LIMITADA

- Instalar software ou hardware (drivers)
- Alterar o nome ou o tipo de sua própria conta (essa atribuição é exclusiva do usuário com conta de administrador).

Para atribuir as permissões, utilizamos o comando GRANT, com a seguinte sintaxe:

```
GRANT { ALL | statement [ ,...n ] }

TO security_account [ ,...n ]
```

Agora vamos exemplificar a utilização do comando GRANT.

Exemplo 1

Garantir para o login SERVIDOR\wilson a permissão de criar novos bancos de dados:

```
USE Master

GRANT CREATE DATABASE TO [SERVIDOR\wilson]
```

Usamos, primeiro, o comando USE Master, para tornar o Banco de Dados Master o banco atual, pois isto é condição para que o sistema permita a criação de novos bancos de dados. Na prática: ao criar um banco de dados, o usuário precisa gravar os dados nas tabelas do Banco de Dados Master, que concentra as informações sobre todo o conteúdo de uma instância do **SQL** Server.

Structured Quary Language (Linguagem de Consulta Estruturada) criada no início da década de 1970, na IBM.

Quando o login for um usuário do domínio do Windows, o nome deve vir entre colchetes, como no exemplo: [SERVIDOR\wilson]

Podemos atribuir mais do que uma permissão ao mesmo tempo, para um ou mais logins ou usuários.

Exemplo 2

Atribuir as permissões CREATE TABLE, CREATE RULE e CREATE VIEW, para o usuário SERVIDOR\wilson no banco de dados Exemplo1.

```
USE Exemplo1

GRANT CREATE TABLE, CREATE RULE, CREATE VIEW

TO [SERVIDOR\wilson]
```

Exemplo 3

Atribuir as permissões CREATE TABLE, CREATE RULE e CREATE VIEW, para os usuários SERVIDOR\grupo1 e SERVIDOR \grupo2, no Banco de Dados Exemplo1.

```
USE Exemplo1

GRANT CREATE TABLE, CREATE RULE, CREATE VIEW

TO [SERVIDOR \grupo1], [SERVIDOR \grupo2]
```

Abaixo, listamos as principais permissões de banco de dados que o comando GRANT pode atribuir. Lembre-se de que para atribuirmos todas as permissões, empregamos a palavra ALL (todos).

CREATE DATABASE: Banco de Dados master.

CREATE DEFAULT: todos os bancos de dados.

CREATE PROCEDURE: todos os bancos de dados.

CREATE RULE: todos os bancos de dados.

CREATE TABLE: todos os bancos de dados.

CREATE VIEW: todos os bancos de dados.

BACKUP DATABASE: todos os bancos de dados.

BACKUP LOG: todos os bancos de dados.

Já as principais permissões de objetos do banco de dados são estas:

SELECT: Tabelas, views e colunas.

INSERT: Tabelas e views.

DELETE: Tabelas e views.

UPDATE: Tabelas, views e colunas.

EXECUTE: Stored Procedures.

Exemplo

Atribuir todas as permissões para o usuário SERVIDOR\andrea, no banco de dados Exemplo1.

```
USE Exemplo1
GRANT ALL
TO [SERVIDOR\andrea]
```

Já para retirar as permissões de banco de dados, devemos recorrer ao comando REVOKE. Saiba que podemos retirar mais do que uma permissão ao mesmo tempo, para um ou mais logins, como mostramos nos exemplos 2 e 3.

Mas, antes, confira a sintaxe para o comando REVOKE:

```
REVOKE { ALL | statement [ ,...n ] }

FROM security_account [ ,...n ]
```

Exemplo 1
Retirar a permissão de criar novos Bancos de Dados, atribuída para o login SERVIDOR\wilson, anteriormente.

```
USE MASTER

REVOKE CREATE DATABASE TO [SERVIDOR\wilson]
```

Exemplo 2
Retirar as permissões CREATE TABLE, CREATE RULE e CREATE VIEW, atribuídas para o usuário SERVIDOR\wilson no Banco de Dados Exemplo1.

```
USE Exemplo1
REVOKE CREATE TABLE, CREATE RULE, CREATE VIEW
TO [SERVIDOR\wilson]
```

Exemplo 3
Retirar as permissões CREATE TABLE, CREATE RULE e CREATE VIEW, atribuídas para os usuários SERVIDOR\grupo1 e SERVIDOR\grupo2, no Banco de Dados Exemplo1.

```
USE Exemplo1

REVOKE CREATE TABLE, CREATE RULE, CREATE VIEW

TO [SERVIDOR\grupo1], [SERVIDOR\grupo2]
```

Exemplo 4
Retirar todas as permissões atribuídas ao usuário SERVIDOR\andrea, no Banco de Dados Exemplo1.

```
USE Exemplo1

REVOKE ALL

TO [SERVIDOR\andrea]
```

3.4.1.3. Permissões a objetos do banco de dados

As permissões a objetos aplicam-se aos diversos elementos de um banco de dados. Em uma tabela, por exemplo, podemos ter permissão de leitura dos dados (SELECT), adição de novos registros (INSERT), exclusão de registros (DELETE) e assim por diante. As permissões a objetos são as que mais diretamente se relacionam com as atividades diárias dos usuários.

Se determinado usuário utiliza uma aplicação que acessa o banco de dados no servidor SQL Server e deve ter permissão somente de leitura de dados de uma determinada tabela, o incluiremos em um role com permissão SELECT na tabela e pronto, ele apenas poderá consultar os dados. A seguir confira quais são as principais permissões a objetos:

SELECT: Tabelas, views e colunas.

INSERT: Tabelas e views.

DELETE: Tabelas e views.

UPDATE: Tabelas, views e colunas.

EXECUTE: Stored procedures.

REFERENCES: Tabelas e colunas.

Também, para atribuir permissões de objetos do banco de dados devemos utilizar o comando GRANT. Nesse caso, a sintaxe do comando GRANT é essa:

```
GRANT
{ ALL [ PRIVILEGES ] | permission [ ,...n ] }
{
[ ( column [ ,...n ] ) ] ON { table | view }
| ON { table | view } [ ( column [ ,...n ] ) ]
| ON { stored_procedure | extended_procedure }
| ON { user_defined_function }
}
TO security_account [ ,...n ]
[ WITH GRANT OPTION ]
[ AS { group | role } ]
```


Como a sintaxe completa não é nada simples, vamos recorrer a alguns exemplos para ilustrar a utilização do comando GRANT. Lembre-se de que podemos atribuir mais do que uma permissão ao mesmo tempo, para um ou mais logins ou usuários, como mostra o exemplo 2.

Exemplo 1
Para garantir ao usuário SERVIDOR\wilson a permissão de selecionar novos registros e atualizar os registros existentes, na tabela Clientes do Banco de Dados Exemplo1:

```
Use Exemplo1
GRANT SELECT, UPDATE ON Clientes
TO [SERVIDOR\wilson]
```

Quando o login for um usuário do Windows, o nome deve vir entre colchetes, como no exemplo:

```
[SERVIDOR\wilson]
```

Exemplo 2
Garantir para os usuários SERVIDOR\wilson e SERVIDOR\andrea a permissão de selecionar novos registros, atualizá-los e excluí-los, na tabela Clientes do Banco de Dados Exemplo1:

```
Use Exemplo1
GRANT SELECT, UPDATE, DELETE ON Clientes
TO [SERVIDOR\wilson], [SERVIDOR\andrea]
```

Exemplo 3
Atribuir todas as permissões para o usuário SERVIDOR\andrea, na tabela Clientes do Banco de Dados Exemplo1.

```
USE Exemplo1
GRANT ALL ON Clientes
TO [SERVIDOR\andrea]
```

Observe que, mais uma vez, utilizamos a palavra ALL, para indicar todas as permissões. Para retirar as permissões de objetos do banco de dados utilizamos REVOKE. Neste caso, a sintaxe do comando REVOKE é a seguinte:

```
REVOKE [ GRANT OPTION FOR ]
{ ALL [ PRIVILEGES ] | permission [ ,...n ] }
{
[ ( column [ ,...n ] ) ] ON { table | view }
| ON { table | view } [ ( column [ ,...n ] ) ]
| ON { stored_procedure | extended_procedure }
| ON { user_defined_function }
}
{ TO | FROM }
security_account [ ,...n ]
[ CASCADE ]
[ AS { group | role } ]
```

Agora, acompanhe nos exemplos abaixo a utilização do comando REVOKE. Nos Exemplos 2 e 3 mostramos que se pode retirar mais do que uma permissão ao mesmo tempo, para um ou mais usuários.

Exemplo 1
Retirar a permissão UPDATE, atribuída para o usuário SERVIDOR\wilson, anteriormente.

```
USE Exemplo1

REVOKE UPDATE ON Clientes

TO [SERVIDOR\wilson]
```

Exemplo 2
Retirar as permissões SELECT, UPDATE E DELETE, atribuídas para o usuário SERVIDOR\wilson na tabela Clientes do Banco de Dados Exemplo1.

```
USE Exemplo1

REVOKE SELECT, UPDATE, DELETE ON Clientes

TO [SERVIDOR\wilson]
```

Exemplo 3
Retirar todas as permissões atribuídas ao usuário SERVIDOR\andrea, na tabela Clientes do Banco de Dados Exemplo1.

```
USE Exemplo1

REVOKE ALL ON Clientes

TO [SERVIDOR\andrea]
```

Observe que novamente recorremos à palavra ALL para indicar todas as permissões. Já para negar as permissões de objetos do banco de dados utilizamos o comando DENY. Veja qual é a sintaxe do comando DENY:

```
DENY
{ ALL [ PRIVILEGES ] | permission [ ,...n ] }
{
[ ( column [ ,...n ] ) ] ON { table | view }
| ON { table | view } [ ( column [ ,...n ] ) ]
| ON { stored_procedure | extended_procedure }
| ON { user_defined_function }
}
TO security_account [ ,...n ]
[ CASCADE ]
```

Compreenda melhor essa relativamente complexa sintaxe por meio de três exemplos. Os de números 2 e 3 mostram que podemos negar mais do que uma permissão ao mesmo tempo, para um ou mais usuários.

Exemplo 1
Negar permissão UPDATE, para o usuário SERVIDOR\user1, na tabela Clientes, do Banco de Dados Exemplo1.

```
USE Exemplo1
DENY UPDATE ON Clientes

TO [SERVIDOR\wilson]
```

Exemplo 2
Negar as permissões SELECT, UPDATE E DELETE, para o usuário SERVIDOR\wilson, na tabela Clientes do Banco de Dados Exemplo1.

```
USE Exemplo1

DENY SELECT, UPDATE, DELETE ON Clientes

TO [SERVIDOR\wilson]
```

Exemplo 3
Negar todas as permissões atribuídas ao usuário SERVIDOR\andrea, na tabela Clientes do Banco de Dados Exemplo1.

```
USE Exemplo1

DENY ALL ON Clientes

TO [SERVIDOR\andrea]
```

O Maintenance Plan Wizard (assistente de plano de manutenção) elabora um plano de manutenção que o Agente Microsoft SQL Server pode executar regularmente. Tais como tarefas de administração de banco de dados, backups, verificações de integridade de banco de dados, atualização de estatísticas.

3.4.2. Plano de manutenção de banco de dados

A partir de uma estratégia adequada de backup e restore, um plano de manutenção define uma série de tarefas que devem ser executadas no banco de dados para garantir o seu bom funcionamento e desempenho, bem como a disponibilidade das informações. No plano de manutenção podemos prever tanto tarefas de backup para evitar a perda de informações, quanto de verificação da integridade dos objetos do banco de dados.

É possível criar um plano de manutenção manualmente, mas a maneira mais fácil e prática é fazê-lo por meio do assistente **Maintenance Plan Wizard** (plano de manutenção assistente). Esse programa é capaz de criar e agendar a execução periódica de vários jobs. Cada job realizará uma determinada tarefa que definirmos.

Agora, veremos como se cria um plano de manutenção pelo Wizard. Acompanhe as 24 etapas do passo a passo.

1. Abra o SQL Server Management Studio (Iniciar -> Programas -> Microsoft SQL Server -> SQL Server Management Studio).

2. Na janela Object Explorer, localize a instância SERVIDOR\SQL e dê um clique no sinal +, a seu lado, para o computador exibir as opções disponíveis.

3. Entre as opções que surgirem na tela, clique no sinal +, ao lado da opção Management, para que sejam mostradas novas opções.

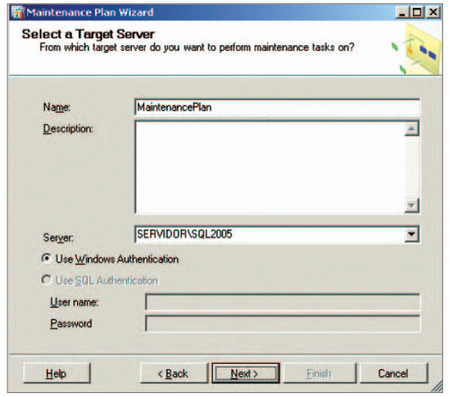


Figura 43
A tela inicial do assistente.

4. Das opções exibidas agora, clique, com o botão direito do mouse, em Maintenance Plans. No menu seguinte, clique em Maintenance Plan Wizard.

5. Surgirá agora a tela inicial do assistente, com uma mensagem informando sobre o que se pode fazer com o programa. Clique então no botão Next, seguindo para a próxima etapa. Aparecerá na tela uma página como a da figura 43.

6. Nessa etapa você definirá em qual instância do SQL Server será criado o plano de manutenção. Por padrão, a instância dentro da qual estava a opção Management -> Maintenance Plans, na qual você clicou com o botão direito do mouse, já vem selecionada. Vamos aceitar a sugestão, pois é exatamente nesta instância que queremos criar nosso plano de manutenção. Clique no botão Next, seguindo para a etapa seguinte do assistente.

7. Nessa fase você definirá quais tarefas integrarão o plano. Por padrão, todas as tarefas já vêm selecionadas. São elas: Chek database integrity, Shrink database, Defragment index(es), Re-index, Update statistics, History cleanup, Launxh SQL Server agent job, Backup database (full), Backup database (Differential) e Backup database (Transaction Log). Mantenha todas as opções marcadas, com exceção de Backup database (Differential). Clique então em Next.

8. Agora você definirá a ordem de execução das tarefas. Para alterar a ordem, marque uma determinada tarefa e depois clique no botão Move up, para movê-la para cima na lista, ou Move down, para movê-la para baixo na lista. Não vamos alterar a ordem sugerida. Assim, clique novamente em Next, e passemos à fase seguinte.

9. Abra a lista Select one or more. Aparecerão diversas opções. Nesta etapa podemos definir se o plano de manutenção incluirá todos os Bancos de Dados da Instância (All databases) ou apenas os do sistema (All system databases – master, model, and msdb). E ainda se incluirá todos os Bancos de Dados do Usuário (All user databases – all databases other than master, model, and msdb) ou apenas os selecionados (These specific databases). A opção These

TAREFAS QUE OS JOBS CRIADOS PELO DATABASE MAINTENANCE PLAN WIZARD PODEM EXECUTAR:

- **Reorganizar os dados nas páginas de dados e índices, por meio da reconstrução dos índices com um novo valor para o parâmetro Fill Factor. Isto garante melhor distribuição dos dados, com melhoria no desempenho.**
- **Compactar o banco de dados, removendo páginas de dados vazias.**
- **Atualizar uma série de informações (Indexes Statistics) sobre os índices do banco de dados.**
- **Fazer a verificação interna na consistência dos dados para certificar-se de que esses não estão corrompidos ou em estado inconsistente.**
- **Agendar o backup do banco de dados e do log de transações.**

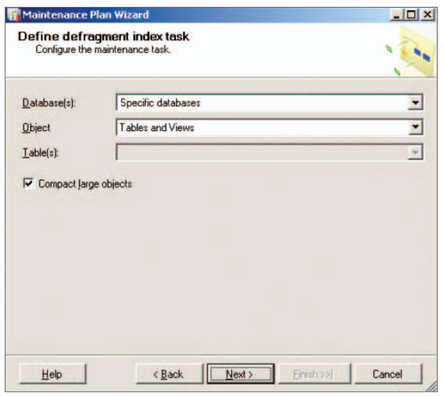
specific databases já vem marcada por padrão. Na lista de bancos de dados selecionamos os que queremos incluir no plano de manutenção.

10. Clique em OK para fechar a lista de opções e certifique-se de que marcou a opção Include indexes, para garantir que a otimização dos índices do banco de dados AdventureWorks seja incluída no plano. Agora clique em Avançar para chegar à fase seguinte do assistente.

11. Nessa etapa, definiremos para quais bancos de dados serão criadas tarefas de otimização. Abra a lista Select one or more, que mostrará diversas opções já descritas anteriormente. A opção These specific databases já vem selecionada por padrão. Na lista de bancos de dados podemos selecionar os que desejamos incluir no plano de manutenção. Em nosso exemplo, incluiremos apenas o Banco de Dados AdventureWorks. Certifique-se, assim, de que este esteja selecionado. Clique em OK para fechar a lista de opções. Ao selecionar um banco de dados, serão habilitadas as porções de otimização e recuperação de espaço. Aceite as configurações sugeridas e clique no botão Next para seguir para a próxima etapa do assistente.

12. Agora podemos definir uma série de configurações a respeito da otimização dos índices e das páginas de dados do banco de dados. Nesta fase você tem três listas de opções. Na primeira poderá ver quais bancos de dados serão incluídos no plano de manutenção, para terem os índices de suas tabelas e views desfragmentados. Abra a primeira lista (Database(s):) e selecione somente o banco de dados AdventureWorks. Na segunda lista (Object), selecione a opção Table (para otimizar somente os índices das tabelas), a opção View (para otimizar somente os índices das views) ou a opção Tables and Views (para otimizar tanto os índices das tabelas quanto das views). Selecione agora Tables and Views. Automaticamente, serão selecionadas todas as tabelas e views. Se você selecionar uma opção individual, como por exemplo Table, na terceira lista, poderá marcar somente determinadas tabelas, para otimização de índices. Com as opções selecionadas, sua janela deve ficar semelhante à que mostramos na figura 44.

Figura 44
As opções tabelas e views.



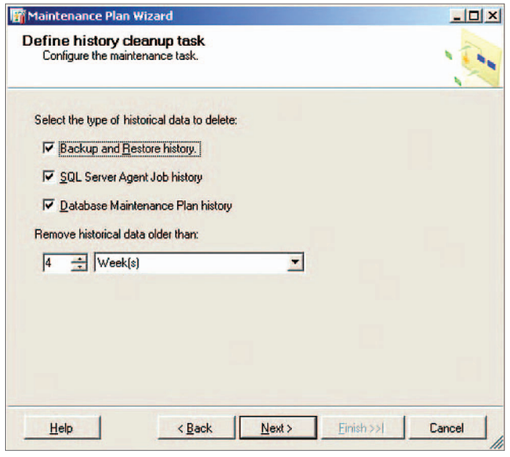


Figura 45
Aceitando as configurações padrão.

13. Clique agora no botão Next para seguir à próxima etapa do assistente.

14. Nessa fase vamos escolher os bancos de dados em que serão criadas tarefas, as quais, ao serem executadas, recriarão os índices. As três primeiras listas funcionam de modo idêntico às três listas apresentadas na figura 45. Na primeira lista você marcará um ou mais bancos de dados, na segunda selecionará as opções **Table**, **Views** ou **Tables** and **Views** e, na terceira, os objetos, individualmente, dependendo de qual opção foi selecionada na segunda lista.

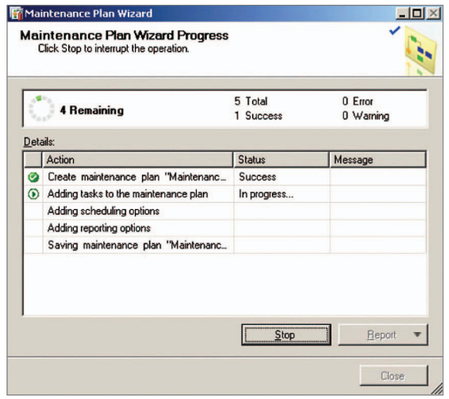
Esta tela tem uma série de opções avançadas, que somente um DBA experiente deverá alterar. Portanto, somente modifique essas opções se souber exatamente o que está fazendo.

Para o nosso exemplo, selecione, na primeira lista, o banco de dados AdventureWorks e, na segunda, a opção Tables and Views. Mantenha as demais opções inalteradas. E clique em Next para seguir adiante.

15. Nessa fase, você selecionará para quais bancos de dados serão criadas tarefas para atualizar as estatísticas das tabelas e views. As três primeiras listas funcionam exatamente como as três mostradas na figura 46. Marque, na primeira lista, um ou mais bancos de dados. Na segunda, selecione as opções Table, Views ou Tables and Views e, na terceira lista, os objetos, individualmente, dependendo de qual opção foi selecionada na segunda lista. Para nosso exemplo, escolha, na primeira lista, o banco de dados AdventureWorks e, na segunda, a opção Tables and Views. Mantenha as demais opções inalteradas e clique no botão Next para passar à fase seguinte do assistente.

16. Nessa etapa você selecionará quais opções de históricos serão limpas quando as tarefas do plano de manutenção (específicas para limpeza dos históricos) forem executadas e a periodicidade da execução. Você pode marcar as opções Backup and Restory history, SQL Server Agent Job history e Database Maintenance Plan History. Por padrão, as três opções já vêm selecionadas. Na parte de baixo da janela, você definirá quais dados devem ser excluídos do histórico. Por padrão também, já vem selecionada a opção 4 Week(s), significando que serão criadas tarefas, no plano de manutenção, para excluir

Figura 46
Finalizando a criação do plano de manutenção.



dados que tenham sido gravados mais de quatro semanas antes nos históricos selecionados. Aceitaremos as configurações padrão. Agora clique em Next, seguindo adiante.

17. Nessa etapa serão exibidos os jobs já existentes, criados anteriormente. Você pode marcar um ou mais jobs para serem executados, também como parte do plano de manutenção. No nosso exemplo, incluiremos todos os jobs já existentes. Portanto, certifique-se de que todos foram selecionados e clique em Next.

18. Agora você definirá para quais bancos de dados serão criadas tarefas de backup, como parte do plano de execução. Abra a lista Databases e, abaixo da opção These specific databases, marque o banco de dados AdventureWorks e clique em OK. As demais opções serão habilitadas. Você pode definir se o backup será feito em disco ou fita e em qual pasta (no caso de backup em disco), se os backups já existentes devem ser sobrescritos ou não e assim por diante. Defina as opções desejadas e clique em Next.

19. Nessa etapa você pode criar tarefas que farão o backup diferencial de um ou mais bancos de dados. Vamos incluir um backup diferencial do banco de dados AdventureWorks como parte do plano de manutenção. Na lista Database(s): selecione o banco de dados AdventureWorks e clique em OK. Aceite as demais opções e clique em Next, seguindo à fase seguinte.

20. Nessa etapa você poderá criar tarefas que farão o backup do log de transações de um ou mais bancos de dados. Vamos incluir um backup do log do banco de dados AdventureWorks, como parte do plano de manutenção. Na lista Database(s): selecione o banco de dados AdventureWorks e clique em OK. Aceite as demais opções e siga adiante, clicando em Next.

21. Agora você poderá definir uma conta conhecida como Proxy Account. Se for configurada uma conta como esta, as tarefas serão executadas no contexto desta conta, que, portanto, deverá ter todas as permissões necessárias para executar as tarefas do plano de manutenção. Não vamos configurar uma conta como Proxy Account em nosso exemplo. Então, clique novamente em Next.

22. Nessa etapa você definirá em qual pasta será gravado um relatório sobre o plano de manutenção. Por padrão, vem selecionada a pasta C:\. Aceite as configurações sugeridas e clique em Next.

23. Será exibida agora a tela final do Wizard. Se você precisar alterar alguma opção, recorra ao botão Back. Para finalizar o assistente e criar o plano de manutenção, clique em Finish. O SQL Server mostrará o progresso da criação do plano de manutenção, conforme indica a figura 46.

24. Uma vez concluída a criação do plano de manutenção, o sistema exibirá uma janela com o resultado da criação. Clique em Close para fechar a janela. Pronto, o plano de manutenção foi criado.

3.4.3. Uma linguagem versátil: SQL

Segundo Oliveira (2000), quando os bancos de dados relacionais estavam em desenvolvimento, foram criadas linguagens para manipulá-los. A SQL (sigla do inglês Structured Query Language, literalmente Linguagem de Consulta Estruturada) foi desenvolvida no início dos anos 1970, no departamento de pesquisas da IBM, como interface para o sistema de banco de dados relacional denominado SYSTEM R. Em 1986, o American National Standard Institute (ANSI) publicou um padrão de SQL e essa linguagem acabou se tornando padrão de banco de dados relacional.

A SQL apresenta uma série de comandos para definir dados, chamada de DDL (Data Definition Language ou linguagem de definição de dados) a qual é composta, entre outros, pelo Create, que se destina à tarefa de criar bancos de dados. Já os comandos da série DML (Data Manipulation Language ou linguagem de manipulação de dados), possibilitam consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Uma subclasse de DML, a DCL (Data Control Language ou linguagem de controle de dados) dispõe de comandos de controle, como o próprio nome diz.

Conforme Oliveira (2000), a grande virtude da linguagem SQL é sua capacidade de gerenciar índices sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro. Outra característica bastante importante em SQL é sua capacidade de construção de visões, ou seja, formas de visualizar os dados em listagens independentes das tabelas e organizações lógicas dos dados.

Também é interessante na linguagem **SQL** o fato de permitir o cancelamento de uma série de atualizações ou de gravá-las, depois que iniciarmos uma sequência de atualizações. Isto pode ser feito por meio dos comandos Commit e Rollback.

É importante notar que a linguagem SQL só pode nos fornecer tais soluções, porque está baseada em bancos de dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

3.4.3.1. Como utilizar os comandos SQL

Segundo Oliveira (2001) a linguagem SQL foi desenvolvida para acessar os bancos de dados relacionais. Seu objetivo é fornecer um padrão de acesso aos bancos de dados, seja qual for a linguagem usada em seu desenvolvimento. Mas, apesar da tentativa de torná-la um padrão (ANSI), cada fornecedor hoje possui uma série de extensões que deixam as várias versões incompatíveis entre si. Alguns bancos de dados suportam o padrão SQL ANSI-92, mais abrangente, o que representa um esforço para facilitar o processo de tornar transparente a base de dados utilizada pela aplicação. Entretanto, nem todos os fornecedores já oferecem suporte completo ao SQL ANSI-92 porque para isto teriam de alterar partes estruturais de seus sistemas gerenciadores.

3.4.3.2. Categorias da Linguagem SQL

De acordo com Oliveira (2001), as instruções SQL podem ser agrupadas em três grandes categorias:

- **DDL (Declarações de Definição de Dados):** parte da linguagem com comandos para criação de estruturas de dados como tabelas, colunas, etc. Exemplo: CREATE TABLE.

- **DML (Declarações de Manipulação de Dados):** parte da linguagem com comandos para acessar e alterar os dados armazenados no banco de dados. Os principais comandos dessa categoria são: SELECT, UPDATE, INSERT e DELETE.

- **DCL (Declarações de Controle de Dados):** parte da linguagem com comandos para definir usuários e controlar seus acessos aos dados. Exemplo: GRANT.

3.4.3.2.1. Instruções SQL

• Instrução CREATE DATABASE

De acordo com Oliveira (2000), para gerenciar os bancos de dados com comandos SQL é necessário que se esteja posicionado no banco de dados Master. Podemos criar um banco de dados com o comando SQL, CREATE DATABASE. A sintaxe completa é:

```
CREATE DATABASE nome_bancodedados
[ON {
[PRIMARY] (NAME = nome_logico_arquivo,
        FILENAME = 'caminho_e_nome_arquivo'
        [, SIZE = tamanho]
        [, MAXSIZE = tamanho_maximo]
        [, FILEGROWTH = taxa_crescimento]
        } [, ... n]
[LOG ON
        {
        (NAME = nome_logico_arquivo.
        FILENAME = 'caminho_e_nome_arquivo'
        [, SIZE = tamanho])
        } [, ..n]
]
```

Onde:

- **Nome_bancodedados:** é o nome do banco de dados que se deseja criar.

- **Nome_logico_arquivo:** é um nome usado para referenciar o arquivo em quaisquer comandos SQL executados depois que o banco de dados tiver sido criado.

- **PRIMARY:** especifica o grupo de arquivos primwário. Esse grupo deve conter todas as tabelas de sistema para o banco de dados. Um banco de dados só pode ter um grupo de arquivo PRIMARY. Se não for especificado algum, o primeiro listado será o primário.

- **FILENAME:** aqui se deve especificar o caminho e o nome do arquivo que estamos criando. O arquivo deve necessariamente estar na mesma máquina que o servidor SQL, mas pode estar em uma unidade de disco diferente.

- **SIZE:** especifica o tamanho em megabytes que queremos alocar para o banco de dados. O valor mínimo é de 1MB, mas o padrão é 3MB para arquivos de dados e 1MB para arquivos de log.

- **MAXSIZE:** permite especificar o tamanho máximo que o arquivo pode atingir. O padrão possibilita que o arquivo cresça até que o disco fique cheio.

- **FILEGROWTH:** especifica a taxa de crescimento do arquivo. Esse ajuste não pode exceder a configuração de MAXSIZE. Um valor zero indica que não é permitido aumento. O padrão é 10%, significando que cada vez que o arquivo cresce, será alocado um espaço adicional de 10% para ele. Um

banco de dados que estiver em mais de um arquivo só é expandido depois que o último arquivo se completar.

- **LOG ON:** se aplicam aqui as mesmas definições mostradas anteriormente, exceto pelo fato de que se criará o arquivo de log de transações, e não o arquivo de dados.

• Sintaxe simplificada:

```
CREATE DATABASE [IF NOT EXISTS] nome_banco_de_dados
```

Atenção: se não especificarmos IF NOT EXISTS, e o banco de dados já existir, ocorrerá um erro.

Exemplo utilizando o MYADMIN:

No exemplo criaremos um banco de dados com o nome banco_teste, como mostra a figura 47: CREATE DATABASE banco_teste

Digitamos a instrução e clicamos no botão executar, para obter o retorno do myadmin apresentado na figura 48.

Figura 47
Criando um banco de dados.

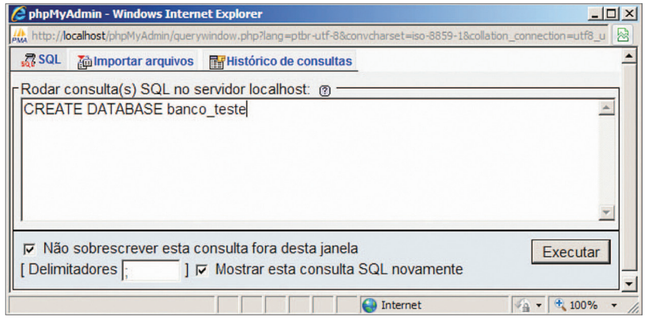
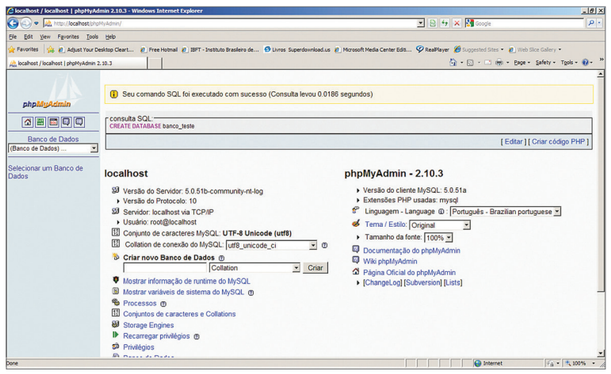


Figura 48
Retorno do myadmin.



• Instrução CREATE TABLE

Como sugere o nome, é usada para criar tabelas. Atenção para sintaxe:

```
CREATE TABLE Nome_Tabela

( Nom_Coluna_1 Tipo [CONSTRAINT PRIMARY| KEY| NOT NULL ]


Nom_Coluna_n Tipo [CONSTRAINT PRIMARY| KEY| NOT NULL ])
```

Exemplo:
CREATE TABLE Cliente (codigo int(7), nome varchar(40), endereco varchar(40)). Observe a figura 49.

Devemos clicar no botão executar, para ter o resultado da criação da tabela que aparece na figura 50.

Podemos verificar no lado esquerdo da tela que, agora, nosso banco de dados banco_teste possui uma tabela.

• Cláusula INSERT

O comando INSERT insere linhas em uma tabela e, sua forma mais simples, somente uma linha de dados. A sua sintaxe é:

```
INSERT [INTO] nome_tabela (colunas )

VALUES ( valores )
```

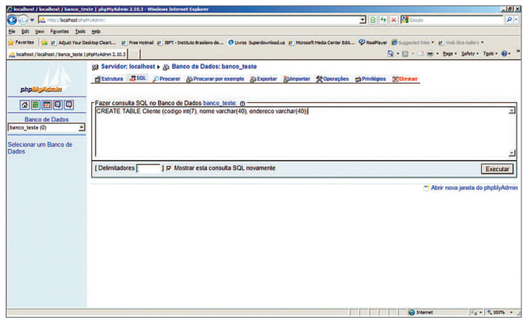
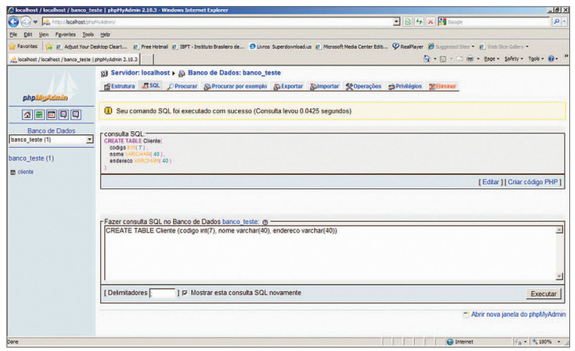


Figura 49
Criando uma tabela.

Figura 50
A tabela criada.



Onde:

Nome_tabela: é o nome da tabela em que se deseja incluir os dados.

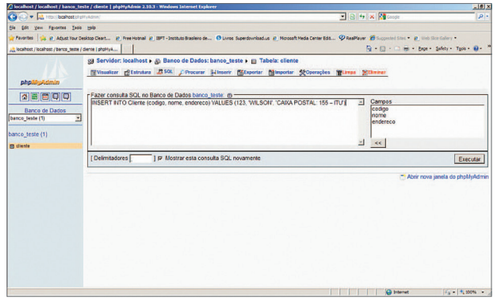
Colunas: parte da tabela onde se deseja acrescentar os dados.

Valores: é o conteúdo de cada coluna.

Exemplo no MYSQL:

INSERT INTO Cliente (codigo, nome, endereco) VALUES (123, 'WILSON', 'CAIXA POSTAL: 155 – ITU'). Como ilustra a figura 51.

Figura 51
Uso do INSERT.



Com esta instrução, iremos incluir um registro, como mostra a figura 52.

Para verificar a inclusão, utiliza-se a instrução SELECT, conforme mostra a figura 53.

Select * from cliente

E teremos o resultado mostrado na figura 54, na qual aparece listado somente o cliente Wilson.

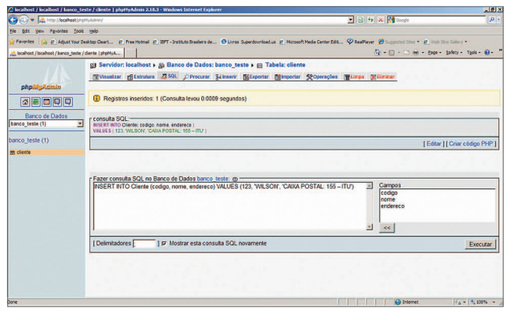


Figura 52
Inclusão de um registro.

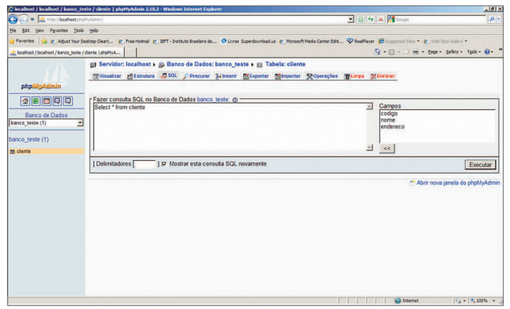


Figura 53
Verificando a inclusão.

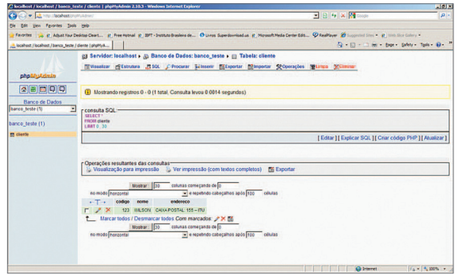


Figura 54
Registro do cliente Wilson.

• Cláusula UPDATE

Conforme Oliveira (2001), a cláusula UPDATE tem a finalidade de alterar campos de um conjunto de registros. Ou seja, para modificarmos uma ou mais linhas existentes, devemos utilizar a declaração UPDATE, cuja sintaxe é a seguinte:

```
UPDATE tabela
SET coluna=valor
Where condição
```

Onde:

Tabela: é o nome da tabela a ser atualizada.

Coluna: é o nome da coluna a ser atualizada.

Valor: é o novo valor para a coluna.

SET: determina os campos que receberão os valores.

Where: determina em quais registros a mudança ocorrerá. Na sua ausência, a mudança ocorrerá em todos os registros da tabela.

Exemplo em MYSQL:

```
UPDATE Cliente
Set nome = "Wilson Oliveira"
Where codigo= 123. (figura 55)
```

Para conferir, vamos fazer um select na tabela:

Select * from cliente. Acompanhe na figura 56.

Podemos observar na figura 57, que o nome do cliente agora aparece como Wilson Oliveira e não mais Wilson, apenas.

Figura 55
Cláusula update.

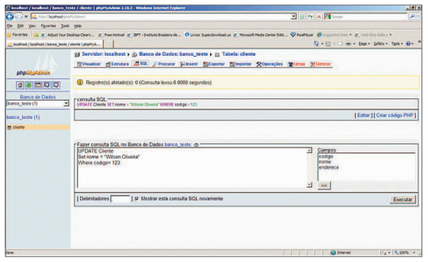
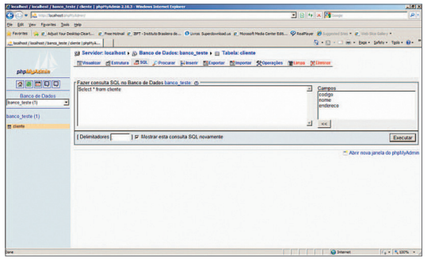


Figura 56
Select na tabela.



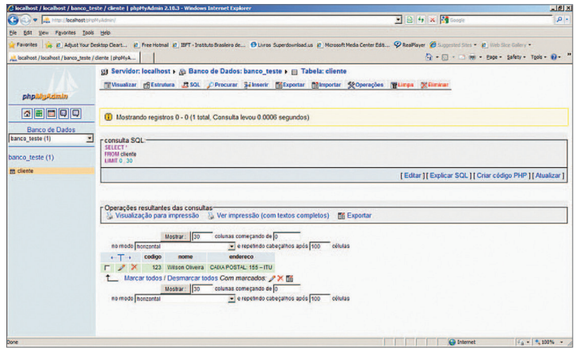


Figura 57
O nome do cliente completo na tabela.

• Cláusula DELETE

Tem a função de eliminar registros de uma tabela. O comando DELETE exclui permanentemente uma ou mais linhas, baseado em uma condição. Sintaxe é:

DELETE From nome_tabela Where condição

Onde:

Nome_tabela: é o nome da tabela em que se deseja excluir os dados.

Where: determina quais registros serão eliminados da tabela.

Condição: é a condição para selecionar os dados que se deseja excluir.

DELETE FROM Cliente Where codigo = 123. Veja a figura 58.

Quando damos o comando DELETE, o MYSQL solicita a confirmação, para que não cometamos um erro irreparável (ou quase, pois teríamos que retornar o backup ou redigitar os dados eliminados), como ilustra a figura 59.

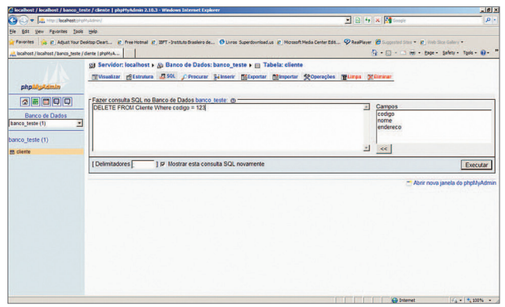
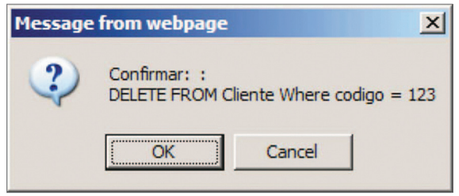


Figura 58
O comando DELETE.

Figura 59
Para confirmar o comando DELETE.



Se confirmarmos, teremos o retorno, conforme se apresenta na figura 60.

É preciso, então, fazer um select para verificar se o registro foi realmente eliminado (figura 61).

Podemos observar na figura 62 que, de acordo com o retorno, não há nenhum registro em nossa tabela.

Figura 60
O retorno depois do delete.

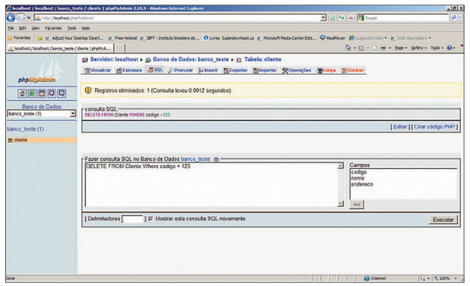


Figura 61
O SELECT para verificar o registro.

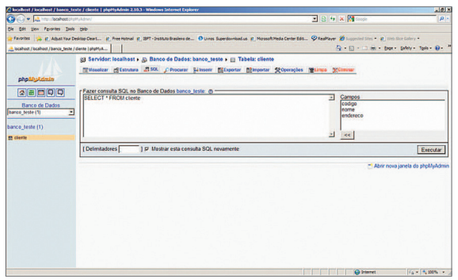
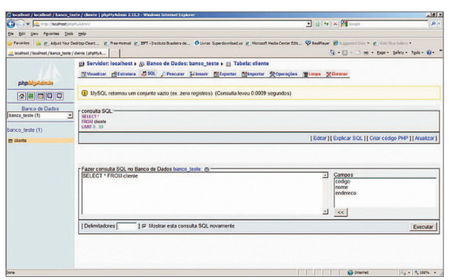


Figura 62
Sem registro na tabela.



• Comando SELECT

O Comando SELECT busca informações de um banco de dados. A sintaxe é:

```
SELECT [DISTINCT] {*, coluna [alias], ...}
FROM tabela;
```

Onde:
DISTINCT: elimina as colunas duplicadas da consulta.

***:** seleciona todas as colunas da tabela.

Coluna: especifica as colunas desejadas na pesquisa.

alias: fornece às colunas diferentes cabeçalhos.

FROM: especifica em qual tabela desejamos realizar a consulta.

Tabela: especifica que tabela será utilizada para pesquisa.

```
Exemplo:
SELECT codigo, nome
FROM departamento;
```

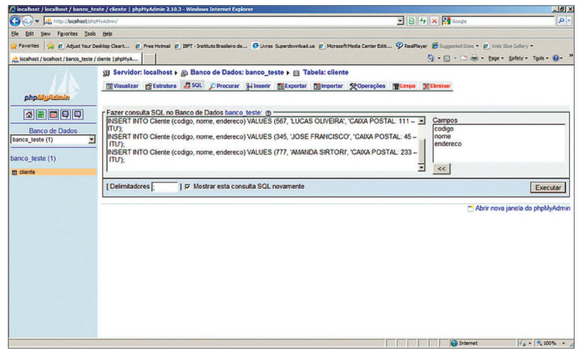
Exemplo com alias:

```
SELECT nome "Nome", salario*12 "Salário Anual"
FROM empresas;
```

Vamos fazer algumas inclusões para melhor verificação das instruções SELECT:

```
INSERT INTO Cliente (codigo, nome, endereco) VALUES (123,
        'WILSON OLIVEIRA', 'CAIXA POSTAL: 155 – ITU');

INSERT INTO Cliente (codigo, nome, endereco) VALUES (145,
'ANDREA SIRTORI OLIVEIRA', 'CAIXA POSTAL: 135 – ITU');

INSERT INTO Cliente (codigo, nome, endereco) VALUES (567, 'LU
        CAS OLIVEIRA', 'CAIXA POSTAL: 111 – ITU');

INSERT INTO Cliente (codigo, nome, endereco) VALUES (345,
        JOSE FRANCISCO', 'CAIXA POSTAL: 45 – ITU');

INSERT INTO Cliente (codigo, nome, endereco) VALUES (777,
        'AMANDA SIRTORI', 'CAIXA POSTAL: 233 – ITU');
```

Figura 63
O comando SELECT.



Observe a figura 63.

O resultado da inclusão aparece na figura 64.

Agora, façamos um select para verificar os registros incluídos: Select * from cliente (figura 65). Teremos o que aparece na figura 66.

3.4.3.4. Views (Visões)

Uma visão (VIEW) é uma forma alternativa de olhar os dados de uma ou mais tabelas. Para definir uma visão, usa-se um comando SELECT, que faz uma consulta sobre as tabelas. A visão aparece depois, como se fosse uma tabela.
Uma view é como uma janela que dá acesso aos dados da tabela, mas com restrições. Tem, no entanto, uma série de vantagens:

- Uma visão pode restringir quais colunas da tabela podem ser acessadas (para leitura ou modificação), o que é útil no caso de controle de acesso.
- Uma consulta SELECT, usada muito frequentemente, pode ser criada como visão. Com isso, a cada vez que é necessário fazer uma consulta, basta selecionar dados da visão.
- Visões podem conter valores calculados ou valores de resumo, o que simplifica a operação.

Figura 64
O resultado da inclusão.

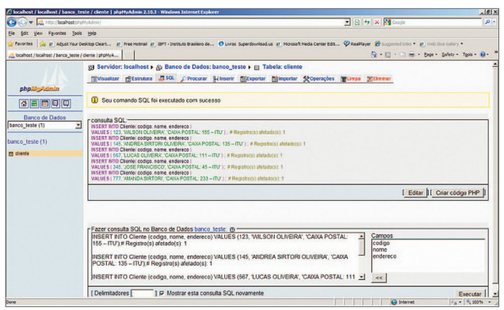


Figura 65
Verificação dos registros incluídos.

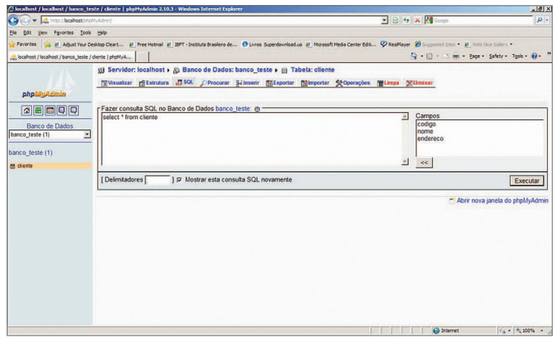
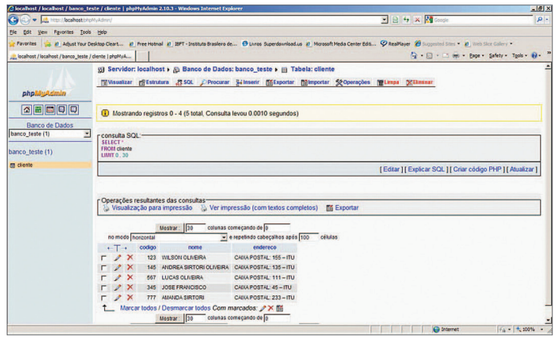


Figura 66
O resultado da verificação.



Criando uma visão com comandos SQL

Para criar uma visão através de SQL, usamos o comando CREATE VIEW. Este comando possui a seguinte sintaxe:

```
CREATE VIEW nome_visao [(coluna [,...n])]
[WITH ENCRYPTION]
AS
        Declaração_select
[WITH CHECK OPTION]
```

Onde:

Nome_visao: é o nome a ser dado à visão.

Coluna: é o nome a ser usado para uma coluna em uma visão. Nomear uma coluna em CREATE VIEW só é necessário quando uma coluna é obtida por uma

expressão aritmética, uma função, ou quando duas ou mais colunas poderiam ter o mesmo nome (frequentemente por causa de uma junção), ou ainda quando a coluna em uma visão recebe um nome diferente do nome da consulta da qual se originou. Os nomes de colunas também podem ser atribuídos no comando SELECT. Caso seja necessário nomear mais de uma coluna, entre com o nome de cada uma delas separado por vírgulas.

WITH ENCRYPTION: criptografa as entradas na tabela SYSCOMMENTS que contém o texto do comando CREATE VIEW.

WITH CHECK OPTION: força todas as modificações de dados executadas na visão a aderirem os critérios definidos na declaração SELECT. Quando uma coluna é modificada por meio de uma visão WITH CHECK OPTION garante que os dados permaneçam visíveis por meio da visão depois que as modificações forem efetivadas.

Exemplo:

A seguir, estamos criando uma VIEW que mostrará o CODIGO, NOME e SALARIO dos funcionários da tabela empregado, cujos salários são maiores que R$ 1.000,00. Esta VIEW terá o nome EMPM1000.

```
CREATE VIEW empm1000 AS
SELECT CODIGO, NOME, SALARIO
FROM EMPREGADO
WHERE SALARIO > 1000
```

Nossa VIEW de nome EMPM1000 foi criada, agora iremos dar um SELECT nela.

```
SELECT O FROM EMPM1000
```

E teremos o resultado dos funcionários com salário maior que R$ 1.000,00

Outro exemplo:

Criaremos uma VIEW chamada EMPMEDIA, baseados na tabela EMPREGADO, que mostrará o maior salário, o menor salário e a média salarial. Utilizaremos o recurso de Alias para mudar os nomes das colunas da VIEW para MAIOR, MENOR e MEDIA, conforme o exemplo seguinte:

```
CREATE VIEW EMPMEDA

(MAIOR, MENOR, MEDIA)

AS

SELECT MAX(SALARIO), MIN(SALARIO), AVG(SALARIO)

FROM EMPREGADO
```

Para vermos o resultado, devemos dar um SELECT em nossa VIEW.

```
SELECT * FROM EMPMEDIA
```

3.4.3.5. Stored Procedures (procedimento armazenado)

Um Stored Procedure (procedimento armazenado) é um conjunto de comandos SQL que são compilados e guardados no servidor. Pode ser acionado a partir de um comando SQL qualquer. Em alguns sistemas de banco de dados, armazenar procedimentos era uma maneira de pré-compilar parcialmente um plano de execução, o qual então ficava abrigado em uma tabela de sistema. A execução de um procedimento era mais eficiente que a execução de um comando SQL porque alguns gerenciadores de banco de dados não precisavam compilar um plano de execução completamente, apenas tinha que terminar a otimização do plano de armazenado para o procedimento. Além disso, o plano de execução completamente compilado para o procedimento armazenado era mantido na cache dos procedimentos, significando que execuções posteriores do procedimento armazenado poderiam usar o plano de execução pré-compilado.

A vantagem de usar procedimentos armazenados é que estes podem encapsular rotinas de uso frequente no próprio servidor, e estarão disponíveis para todas as aplicações. Parte da lógica do sistema pode ser armazenada no próprio banco de dados, e não precisa ser codificada várias vezes em cada aplicação.

Criando procedimentos armazenados

Para criar um procedimento, devemos utilizar o comando CREATE PROCEDURE. Por exemplo, o procedimento demonstrado a seguir recebe um parâmetro (@nome) e mostra todos os clientes cujos nomes contenham a informação:

```
CREATE PROCEDURE BUSCACLIENTE

@nomebusca varchar(50)
As

SELECT codcliente, Nome from CLIENTE

WHERE Nome LIKE '%'+@nomebusca+'%'
```

Devemos notar que os parâmetros são sempre declarados com @, logo após o nome do procedimento. Um procedimento pode ter zero ou mais parâmetros. Declara-se o nome do procedimento e a seguir o tipo de dados do parâmetro. Ao invés de CREATE PROCEDURE, pode-se utilizar CREATE PROC, com o mesmo efeito.

Dentro do procedimento pode haver vários comandos SELECT e o seu resultado será do procedimento. O corpo do procedimento começa com a palavra AS e vai até o seu final.

Não se pode usar os comandos SET SHOWPLAN_TEXT, e SET SHOWPLAN_ALL dentro de um procedimento armazenado, pois esses devem ser os únicos comandos de um lote (batch).

Executando procedimentos armazenados

Para executar um procedimento usa-se o comando EXEC (ou EXECUTE). A palavra EXEC pode ser omitida, se a chamada de procedimento for o primeiro comando em um Script ou vier logo após um marcador de fim de lote (a palavra GO).

Por exemplo, podemos executar o procedimento anterior da seguinte forma:

BuscaCliente 'an'

Os resultados serão as linhas da tabela cliente onde o valor de nome contém 'an' (se existirem tais linhas).

Ao executar um procedimento, podemos informar explicitamente o nome de cada parâmetro, por exemplo:

BuscaCliente @nomeBusca = 'an'

Isso permite passar os parâmetros (se houver mais de um) fora da ordem em que eles foram definidos no procedimento.

EXEC também pode executar um procedimento em outro servidor. Para isso, a sintaxe básica é:

```
EXEC
Nome_servidor.nome_banco_de_dados.nome_procedimentos
```

Triggers (Gatilhos)

Um Gatilho (Triggers) é um tipo de Stored Procedure, que é executado automaticamente quando ocorre algum tipo de alteração numa tabela. Gatilhos disparam quando ocorre uma operação INSERT, UPDATE ou DELETE numa tabela. Geralmente são usados para reforçar restrições de integridade que não podem ser tratadas pelos recursos mais simples, como regras, defaults, restrições, a opção NOT NULL, etc. Deve-se usar defaults e restrições quando estes fornecem toda a funcionalidade necessária. Um Gatilho também pode ser empregado para calcular e armazenar valores automaticamente em outra tabela.

Exemplo de Gatilhos:

Para utilizá-los, temos que criar antes algumas tabelas que serão usadas como exemplo. A tabela notafiscal conterá os cabeçalhos de notas fiscais. A tabela item notafiscal irá conter itens relacionados com as notas fiscais. Execute o Script a seguir para criar as tabelas:

```
CREATE TABLE NOTAFISCAL

        (NumeroNota numeric(10) primary key,
        ValorTotal numeric(10,2) default(0))

GO

CREATE TABLE ITEMNOTAFISCAL

(NumeroNota numeric(10) foreign key references NotaFiscal,
        CodProduto int foreign key references Produto,
        Quantidade int not null check (Quantidade > 0),
        Primary key (NumeroNota, CodProduto)

)
```

Vamos utilizar Gatilhos para duas finalidades. Primeiro, ao excluirmos uma nota fiscal, todos os seus itens serão excluídos automaticamente. Depois, quando incluirmos um item, a coluna Valor Total será atualizada na tabela NotaFiscal.

Criando os Gatilhos

Gatilhos são sempre criados vinculados a uma determinada tabela. Se a tabela for excluída, todos os seus Gatilhos serão excluídos como consequência. Ao criarmos um Gatilho, podemos especificar em qual ou quais operações ele será acionado: INSERT, UPDATE ou DELETE.

Gatilho para inserção

Realiza a inclusão de uma ou mais linhas na tabela virtual chamada inserted, que contém as linhas que serão incluídas (mas ainda não foram). Essa tabela tem a mesma estrutura da tabela principal. Podemos consultar dados nela com o SELECT, da mesma forma que uma tabela real.

Vamos criar um Gatilho, chamado InclusaoItemNota, que será ativado por uma operação INSERT na tabela ItemNotaFiscal. Primeiro, ele vai verificar se os valores que estiverem sendo inseridos possuem uma Nota Fiscal relacionada ou não. Devemos digitar as instruções a seguir:

```
CREATE TRIGGER InclusaoItemNota
On ItemNotaFiscal for insert
As
    If not exists (select * from
    Inserted, NotaFiscal
    Where inserted.NumeroNota = NotaFiscal.NumeroNota)
    Raiserror('Esse item não contém um número de nota válido')
    Update NotaFiscal
    Set ValorTotal = ValorTotal +
(select i.Quantidade * p.preco from produto p, inserted i
Where p.codProduto = i.CodProduto)
    Where NumeroNota = (select NumeroNota from inserted)
```

Primeiramente, o Gatilho usa as tabelas inserted e NotaFiscal para consultar se existe o valor de NumeroNota na tabela. Caso não exista, o comando RAISERROR gera um erro de execução, com uma mensagem que será retornada para a aplicação. Esse comando efetivamente cancela o comando INSERT que estiver sendo executado.

Depois verifica a quantidade que está sendo inserida (inserted.Quantidade) e multiplica pelo preço do produto (Produto.Preco). Este preço é encontrado na tabela do produto usando como valor de pesquisa o código do produto (inserted.CodProduto). Ele atualiza a nota fiscal relacionada com o item que está sendo inserido, para isso verifica Where NumeroNota=(select NumeroNota from inserted).

Gatilhos para exclusão

Na exclusão, as linhas da tabela são removidas e colocadas na tabela virtual default deleted, que tem a mesma estrutura da tabela principal. Um Gatilho para exclusão pode consultar DELETED para saber quais linhas foram excluídas.

Vamos criar um Gatilho na tabela NotaFiscal para que, quando a nota fiscal for excluída, todos os seus itens de nota relacionados na tabela ItemNotaFiscal, sejam excluídos em cascata. Para isso, devemos utilizar a seguinte sequência:


```
CREATE TRIGGER ExclusaoNota
On NotaFiscal for delete
As
Delete from ItemNotaFiscal
Where NumeroNota in (select NumeroNota from deleted)
```

Gatilhos para atualização

As tabelas INSERTED e DELETED, como já vimos, são tabelas virtuais que podem ser usadas dentro de um Gatilho. A primeira contém os dados que estão sendo inseridos na tabela real e a segunda, os dados antigos, que estão sendo incluídos.

Num Gatilho de atualização (FOR UPDATE), essas duas tabelas também estão disponíveis. No caso, DELETED permite acessar os dados como eram antes da modificação e INSERTED dá acesso aos dados depois da atualização.

Podemos então considerar uma atualização como uma exclusão seguida de uma inserção (excluem-se valores antigos e inserem-se valores novos).

Por exemplo, ao mudar a quantidade em um ItemNotaFiscal, o total da Nota deve ser recalculado. Para isso é levada em conta a diferença entre a quantidade antiga (deleted.Quantidade) e a nova (inserted.Quantidade). Vamos agora criar um Gatilho em ItemNotaFiscal que faz isso:

```
CREATE TRIGGER AlteracaoItemNota
On ItemNotaFiscal for update
    As
    If update(Quantidade) or update(CodProduto)
    Begin
      Update NotaFiscal
      Set ValorTotal = ValorTotal +
      (select p.preco * (i.Quantidade – d.Quantidade)
      From Produto p inner join inserted i
```

```
On p.CodProduto = i.CodProduto inner join deleted  d

On i.CodProduto = d.CodProduto and i.NumeroNota
        = d.NumeroNota)
  End
```

Note que o uso de "if update(nome_da_coluna)", dentro de um Gatilho de atualização, permite descobrir se a coluna está sendo alterada ou não. Isso para evitar trabalho desnecessário, caso não haja alguma modificação.

3.4.3.6. Um exemplo completo

Vamos construir um exemplo utilizando o phpMyAdmin com o banco de dados MySQL. Primeiro, vamos criar um banco de dados chamado ERP, onde armazenaremos as tabelas referentes ao nosso sistema de gestão empresarial.

Para criar esse banco de dados devemos utilizar a seguinte instrução:

```
CREATE DATABASE erp
```

Na figura 67, mostramos como será a instrução utilizando o phpMyAdmin.

O resultado é o sucesso da instrução, como se pode observar na figura 68.

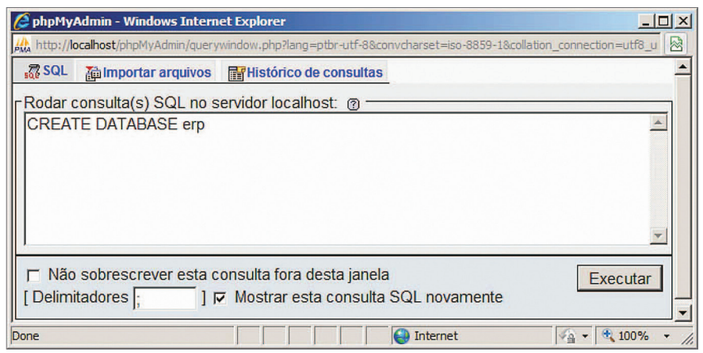
Agora, vamos criar uma tabela de clientes. Para isso, utilizaremos a seguinte instrução:

```
CREATE TABLE Cliente (codigo int(7), nome varchar(40), endereco
varchar(40))
```

Na figura 69 mostramos como será a instrução utilizando o phpMyAdmin.

Podemos observar o sucesso da instrução na figura 70.

Figura 67
Usando o phpMyAdmin.



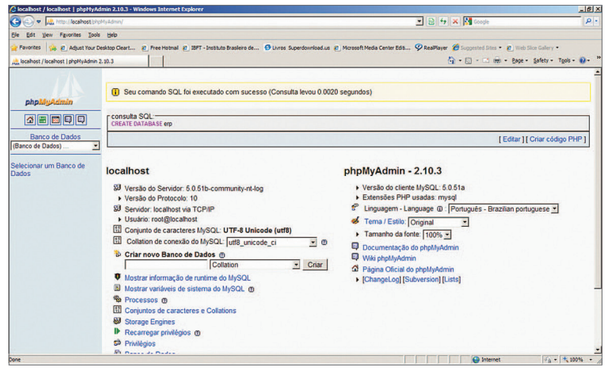


Figura 68
O resultado da instrução.

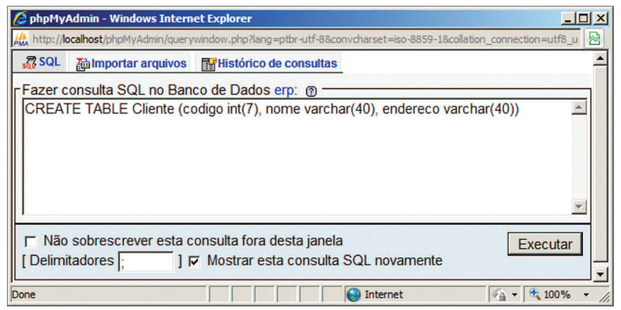


Figura 69
Instrução a partir do phpMyAdmin.

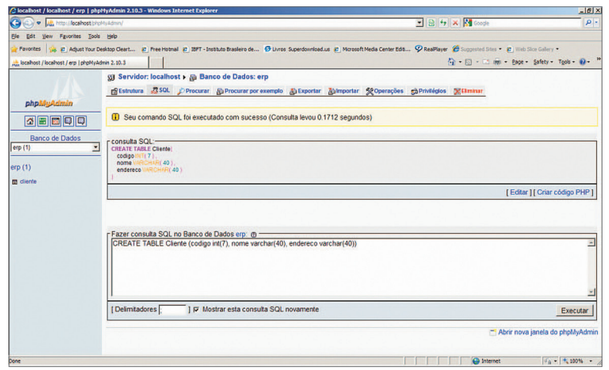


Figura 70
Instrução bem sucedida.

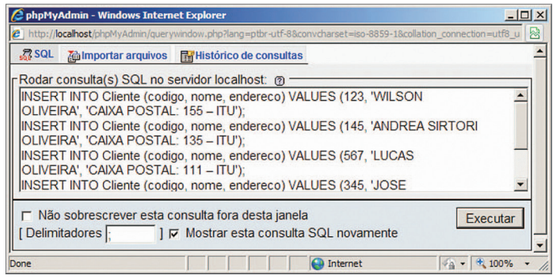
É hora de incluir clientes em nossa tabela para conseguirmos realizar algumas atividades. Utilizaremos as seguintes instruções INSERT para incluir cinco registros:

```
INSERT INTO Cliente (codigo, nome, endereco) VALUES (123,
'WILSON OLIVEIRA', 'CAIXA POSTAL: 155 – ITU');
```

```
INSERT INTO Cliente (codigo, nome, endereco) VALUES (145, 'ANDREA SIRTORI OLIVEIRA', 'CAIXA POSTAL: 135 – ITU');

INSERT INTO Cliente (codigo, nome, endereco) VALUES (567, 'LUCAS OLIVEIRA', 'CAIXA POSTAL: 111 – ITU');

INSERT INTO Cliente (codigo, nome, endereco) VALUES (345, 'JOSE FRANCISCO', 'CAIXA POSTAL: 45 – ITU');

INSERT INTO Cliente (codigo, nome, endereco) VALUES (777, 'AMANDA SIRTORI', 'CAIXA POSTAL: 233 – ITU');
```

Veja na figura 71, como será a instrução utilizando o phpMyAdmin.

Figura 71
Instrução usando phpMyAdmin.

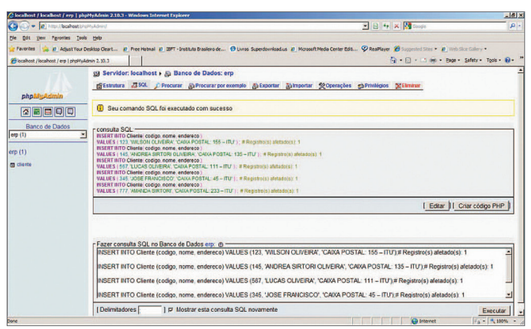


Podemos observar na figura 72 o sucesso da instrução.

Vamos agora fazer algumas manipulações de dados com os registros feitos na tabela de clientes. Listaremos todos esses registros, aplicando, para isto, a seguinte instrução:

```
Select * from Cliente.
```

Figura 72
Sucesso da instrução.



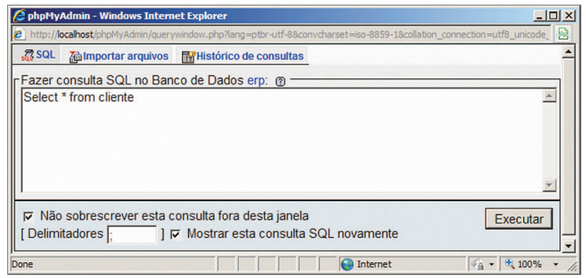


Figura 73
Instrução a partir do phpMyAdmin.

Observe na figura 73 como será a instrução a partir do phpMyAdmin.

O sucesso da instrução pode ser conferido na figura 74.

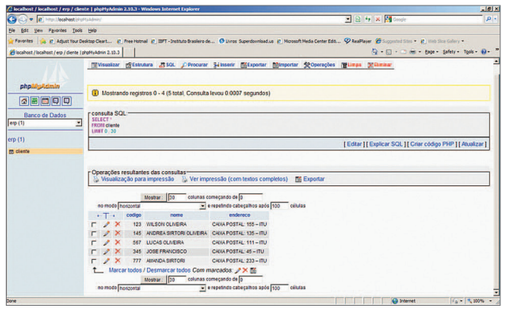


Figura 74
Resultado da instrução.

Vamos agora listar os clientes com código menor que 200. A instrução para isto é a demonstrada a seguir:

```
Select * from cliente where codigo < 200
```

Na figura 75, podemos observar como será a instrução empregando-se o phpMyAdmin.

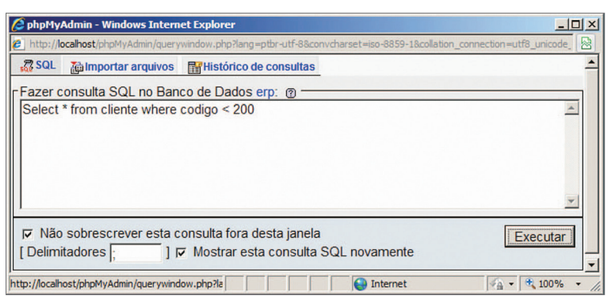
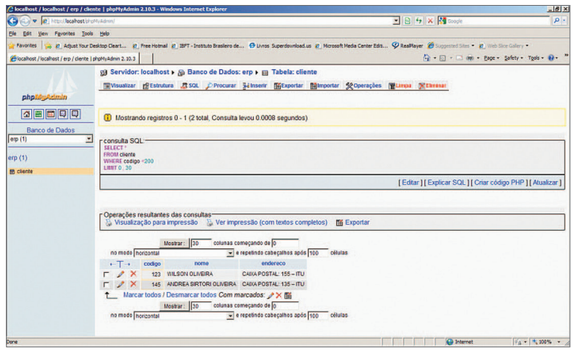


Figura 75
Listagem de clientes com código menor que 200.

Figura 76
Instrução bem sucedida.



A figura 76 demonstra o sucesso da instrução.

Vamos alterar o cliente de código 123, de nome Wilson Oliveira, para o nome completo dele, que é Wilson Jose de Oliveira. Para isso, utilizaremos a instrução a seguir:

```
UPDATE Cliente
Set nome = "Wilson Jose de Oliveira"
Where codigo= 123
```

Na figura 77, indicamos como será a instrução, novamente aplicando-se o phpMyAdmin.

Podemos constatar o sucesso da instrução na figura 78.

Vamos listar o cliente de código 123 para verificar se o UPDATE foi bem-sucedido. Para isso devemos utilizar esta instrução:

```
Select * from cliente where codigo = 123
```

Figura 77
Instrução para o nome completo.

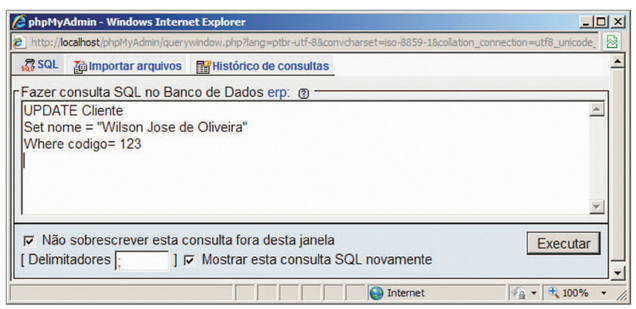
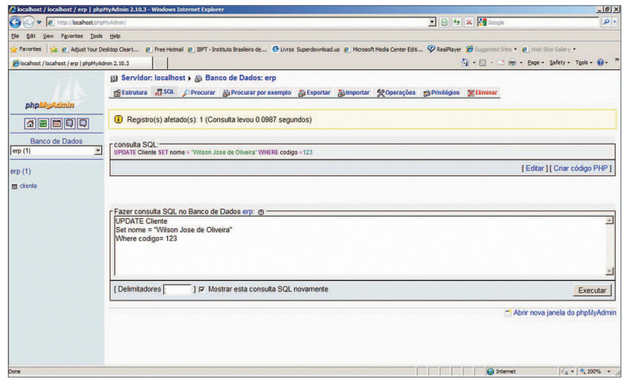
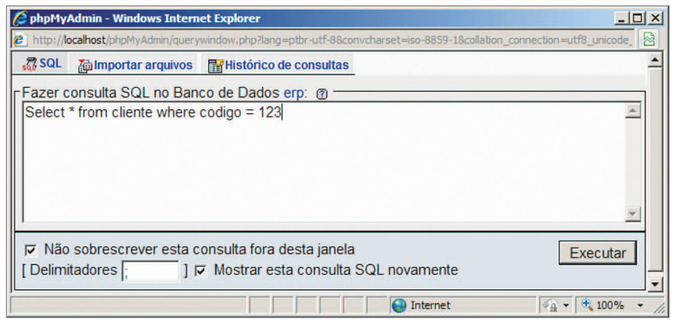


Figura 78
Outra instrução bem sucedida.



Na figura 79, mostramos como será a instrução, outra vez utilizando o phpMyAdmin.

Figura 79
A instrução para listar cliente de código 123.



Podemos observar na figura 80 que a alteração do cliente foi feita.

Figura 80
Realizada a alteração do cliente.

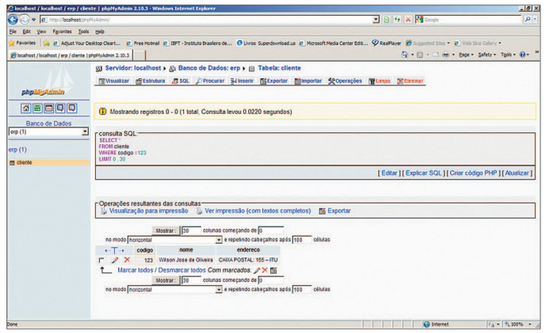
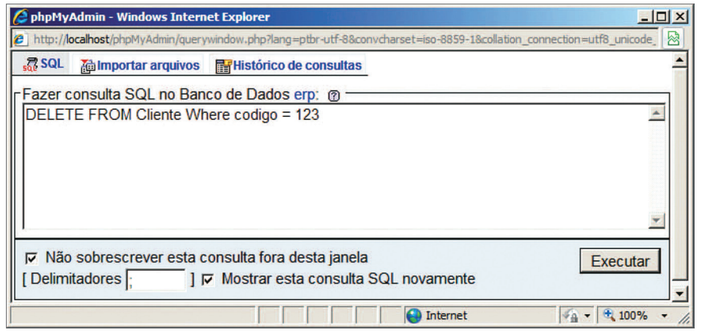


Figura 81
O comando DELETE para o cliente de código 123.



Agora vamos eliminar o cliente de código 123, utilizando a instrução DELETE, com a sintaxe apresentada a seguir:

```
DELETE FROM Cliente Where codigo = 123
```

Na figura 81, mostramos como será a instrução, usando o phpMyAdmin.

Podemos observar que a exclusão do cliente foi realizada na figura 82.

Vamos listar todos os clientes para podermos comprovar que a nossa exclusão do cliente foi bem-sucedida. Para isso, devemos utilizar a instrução SELECT, com a seguinte sintaxe:

```
Select * from cliente
```

Na figura 83, mostramos como será a instrução, aplicando o phpMyAdmin.

Podemos observar agora que o cliente de código 123 já não aparece na lista, como mostra a figura 84.

Figura 82
Efetuada a eliminação.

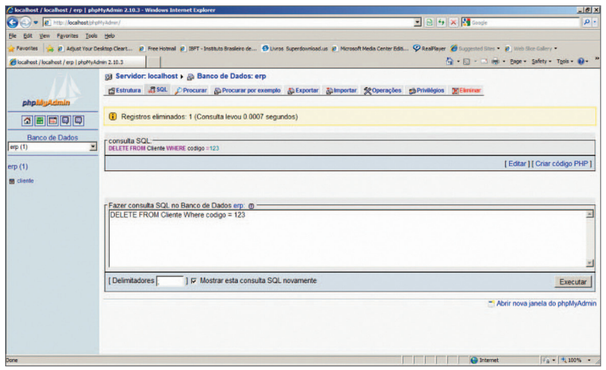


Figura 83
Listagem dos clientes para comprovar exclusão.

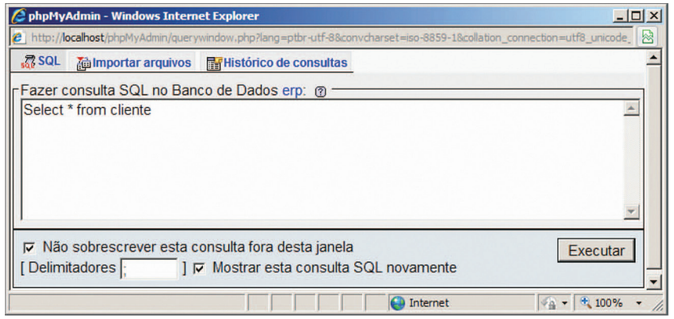
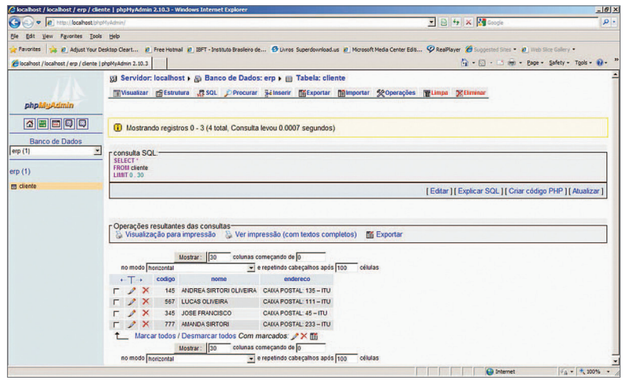


Figura 84
O cliente de código 123 foi eliminado.



Para aprender mais

A linguagem SQL é uma poderosa ferramenta de manipulação de dados. Até agora, aprendemos bastante sobre ela, mas há muito mais que você precisa saber. Para ter um conhecimento mais amplo, pesquise a linguagem mais detalhadamente e analise, por exemplo, instruções específicas para cada banco de dados disponível no mercado, como, por exemplo, SQL Server, Oracle, MySQL e Cache.