



MÓDULO:

BANCO DE DADOS

AUTOR:

CARLOS ALBERTO VALENTE ANDRADE

Módulo de: Banco de Dados;

Autor: Carlos Alberto Valente Andrade

Primeira edição: 2007

Todos os direitos desta edição reservados à
ESAB – ESCOLA SUPERIOR ABERTA DO BRASIL LTDA
<http://www.esab.edu.br>
Av. Santa Leopoldina, nº 840/07
Bairro Itaparica – Vila Velha, ES
CEP: 29102-040

A PRESENTAÇÃO

Este módulo irá apresentar conceitos fundamentais de Sistemas de Bancos de Dados dos principais fabricantes de software.

Nesses conceitos está incluído o Gerenciamento de um Banco de Dados, bem como aspectos de projeto, linguagens e implementação de um Sistema de Banco de Dados, além de explicar os conceitos básicos, fornecer uma visão geral dos recursos de um Banco de Dados e como se inter-relaciona com programas aplicativos, sistema operacional e usuários.

Discutiremos os vários modelos de Banco de Dados, dando ênfase ao modelo Entidade-Relacionamento, e focando na linguagem relacional orientada ao usuário, largamente utilizada, conhecida como SQL.

OBJETIVO

Capacitar o leitor a lidar com um Sistema de Gerenciamento de Banco de Dados (SGBD) aprendendo conceitos e técnicas para a administração eficiente destes dados.

EMENTA

Conceituação e análise de características próprias de Sistemas de Gerência de Banco de Dados multiusuário. Gerenciamento de transações, controle de concorrência, recuperação de falhas, segurança e integridade de dados. Comparação de abordagens não-convencionais para Bancos de Dados; integração de Bancos de Dados e Internet. Estudo dos principais comandos da Linguagem de Consulta Estruturada – SQL.

SOBRE O AUTOR

Carlos Alberto Valente Andrade:

- Professor e Consultor de Tecnologia de Informação
- Doutorando (ITA) e Mestre (IPT) em Engenharia de Computação, Pós-Graduado em Análise de Sistemas (Mackenzie), Administração (Luzwell-SP), e Reengenharia (FGV-SP). Graduado/Licenciado em Matemática.

- Professor e Pesquisador da Universidade Anhembi Morumbi, UNIBAN, e ESAB (Ensino a Distância). Autor de livros em Conectividade Empresarial. Prêmio em E-Learning no Ensino Superior (ABED/Blackboard).
- Consultor de T.I. em grandes empresas como Sebrae, Senac, Granero, Transvalor, etc. Viagens internacionais: EUA, França, Inglaterra, Itália, Portugal, Espanha, etc.

SUMÁRIO

UNIDADE 1	9
O que é um Banco de Dados? e o que vem a ser um SGBD?	9
UNIDADE 2	14
Principais Características de um SGBD	14
UNIDADE 3	19
Conceitos e Arquiteturas de um SGBD	19
UNIDADE 4	23
Características das Principais Arquiteturas de um SGBD	23
UNIDADE 5	28
Níveis de Abstração dos Dados na Arquitetura de um SGBD	28
UNIDADE 6	32
Quem são os usuários de um SGBD?	32
UNIDADE 7	36
Modelagem de Dados I: Modelos Lógicos baseado em Registros.....	36
UNIDADE 8	42
Modelagem de Dados II: Modelos Lógicos baseado em Objetos	42
UNIDADE 9	47
Definição de Entidade, Atributo e Tupla	47
UNIDADE 10	53
Definição de Chave Primária Simples, Composta, Única e Estrangeira	53
UNIDADE 11	56
Relacionamentos entre Bancos de Dados	56
UNIDADE 12	60
Relacionamentos Especiais com Múltiplas Entidades.....	60
UNIDADE 13	62
Integridade dos Dados	62
UNIDADE 14	65
Normalização – A Primeira, Segunda e a Terceira Forma Normal	65

UNIDADE 15	72
Fundamentos de Armazenamento e Manipulação de Dados.....	72
UNIDADE 16	78
A Linguagem SQL.....	78
UNIDADE 17	81
Comando para Criação de um Banco de Dados.....	81
UNIDADE 18	86
Comando SELECT para Consulta de Tabelas.....	86
UNIDADE 19	90
Cláusula e Operadores usados no Comando SELECT.....	90
UNIDADE 20	97
Funções Agregadas e a Cláusula HAVING.....	97
UNIDADE 21	101
Relacionamentos, sub-consulta e união.....	101
UNIDADE 22	106
Inserções, Alterações e Exclusões em Tabelas.....	106
UNIDADE 23	110
Relatórios – comando REPORT	110
UNIDADE 24	114
Privilégios de Acesso – GRANT e REVOKE	114
UNIDADE 25	119
Trabalhando com Índices	119
UNIDADE 26	123
Resumo dos Comandos SQL.....	123
UNIDADE 27	128
Otimizar Consultas SQL.....	128
UNIDADE 28	132
Comando CREATE TRIGGER para criação de um gatilho.....	132
UNIDADE 29	136
Protegendo seu Servidor de Banco de Dados	136
UNIDADE 30	142
Características dos vários SGBDs de uso Comercial	142

GLOSSÁRIO	148
BIBLIOGRAFIA.....	149

UNIDADE 1

Objetivo: Conceituar Banco de Dados, indicando os seus principais componentes básicos. Mostrar a importância e o significado de um SGBD dentro de um Sistema de Banco de Dados

O que é um Banco de Dados? E o que vem a ser um SGBD?

Os Bancos de Dados estão atualmente em situações corriqueiras do nosso cotidiano. Podem-se citar vários exemplos, desde quando anotamos os dados em nossos celulares do telefone e endereço de um amigo, até a inclusão de nossos dados cadastrais, através da Web, num Banco de Dados de uma loja de Comércio Eletrônico.

Podemos dizer que, assim que nascemos, já somos parte de um Banco de Dados!! Na maternidade nos cadastram em Banco de Dados, no cartório para tirar a Certidão de Nascimento, ou mais tarde quando passamos a ter a nossa identidade - o RG, ou mesmo junto à receita federal com o CPF e assim por diante.

Conceito de banco de dados

Pode-se definir de forma bem simples, que Banco de Dados é um conjunto de registros manipuláveis, de mesma natureza, inseridas em um mesmo local, obedecendo a um padrão de armazenamento.

Vimos anteriormente alguns exemplos de Banco de Dados, no entanto, para esse material nós vamos nos dedicar especialmente à técnica de informatizar estas informações de forma que seja possível dar manutenção nestes dados. Algumas dessas tarefas serão como incluir, alterar, consultar ou excluir, com o objetivo de manter estas informações organizadas e disponíveis. E outro ponto é que quando solicitados esses dados que sejam com rapidez e confiabilidade.

Portanto, em outras palavras, podemos afirmar que um Sistema de Banco de Dados é basicamente um sistema computadorizado de manutenção de registros, cuja finalidade é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando solicitadas.

Sistemas multiusuários e monousuários

Os sistemas de Bancos de Dados estão disponíveis em máquinas desde pequenos computadores de mão conhecidos como palmtops (genericamente como handhelds), em clusters de computadores (aglomerado de processadores), ou mesmo em mainframes (computadores de grande porte).

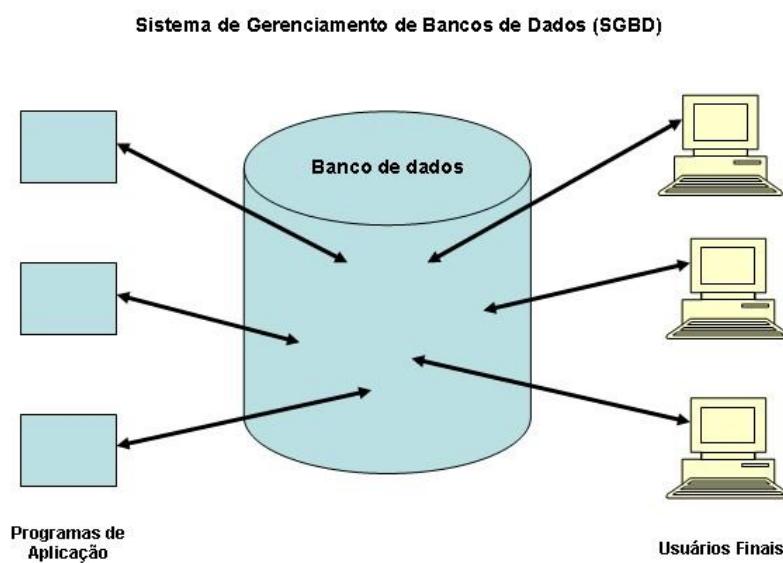


Figura 1.1 - Representação simplificada de um sistema de Banco de Dados

Quando vários usuários acessam o mesmo Banco de Dados, o que é muito comum hoje em dia, denominamos esses sistemas de multiusuários. Enquanto que em máquinas menores, ou em pequenos sistemas, tendem a ser monousuários. Interessante que mesmo em sistemas multiusuários o objetivo é que cada usuário se comporte como se tivesse

trabalhando como num sistema monousuário, de forma que os problemas internos ao sistema sejam invisíveis para o usuário final.

A figura 1.1 é uma imagem simplificada de um Sistema de Banco de Dados. De uma forma geral, um Sistema desses envolve quatro componentes principais: dados, hardware, software e usuários. Vejamos agora cada um destes componentes com mais riqueza de detalhes.

Dados

De modo geral, num Sistema Multusuário os dados são integrados e compartilhados:

Integrados: no sentido de ter uma unificação de vários arquivos para eliminar qualquer redundância (repetição da mesma informação ou dado) parcial ou total entre estes arquivos. Por exemplo, um arquivo de FUNCIONÁRIOS com vários campos sobre os funcionários como nome, endereço, departamento, salário e outros dados. E um arquivo de CURSOS com todas as informações sobre o treinamento que os funcionários tiveram, ou estão tendo, na empresa. Não seria bom senso incluir novamente todos os dados dos funcionários no arquivo CURSOS, pois ficaria redundante ao arquivo FUNCIONÁRIOS que já possui esses dados. Portanto, bastaria integrar estes dois arquivos e eliminar qualquer redundância entre eles.

Compartilhados: quando diferentes usuários acessam os mesmos dados, e provavelmente ao mesmo tempo (acesso concorrente). Por exemplo, no arquivo citado anteriormente, o de FUNCIONÁRIOS, ele poderia ser compartilhado tanto entre os usuários do Departamento Pessoal, como simultaneamente pelos usuários do Departamento de Treinamento.

Hardware

Os componentes de hardware usados num sistema de Banco de Dados consistem basicamente num volume de armazenamento secundário, normalmente em discos magnéticos. Esses discos rígidos são usados para manter por longo tempo os dados armazenados. Normalmente estão associados a um processador de alta performance e memória principal em grande quantidade. Eles darão suporte a execução do software de Sistema de Banco de Dados.

Software

Entre os dados fisicamente armazenados no hardware, e os usuários do sistema, existe o software conhecido como Gerenciador de Banco de Dados. Também denominado de Servidor de Banco de Dados, ou ainda mais frequentemente chamado de Sistema de Gerenciamento de Bancos de Dados (SGBD) - a sigla em inglês é DBMS.

Um SGBD é o conjunto de programas de computador (software) responsável pelo gerenciamento de uma base de dados. Todos os acessos ao Banco de Dados são tratados pelo SGBD. Porém, apesar de ser o componente de software mais importante de todo o sistema, não é o único a ser utilizado. Componentes como utilitários, ferramentas de desenvolvimento, geradores de relatórios, gerenciador de transações e outros, auxiliam também em um SGBD.

Podemos citar como principais exemplos de SGBD: Oracle, SQL-Server (Microsoft), e o DB2 (IBM). E alguns exemplos de SGBD que são software livre: MySQL, PostgreSQL e SQLite (veja detalhes dos principais SGBDs na Unidade 30).

Usuários

Existem várias classes de usuários em um Sistema de Banco de Dados. Temos o Administrador de Banco de Dados (DBA), e o Administrador de Dados (DA), que como o

próprio nome já diz, administram o Banco de Dados para que tenha a melhor performance possível, mantendo a integridade dos dados, além de outras atribuições.

Outros tipos de usuários são os programadores de aplicação, responsáveis pela escrita de programas de aplicações de Banco de Dados em alguma linguagem de programação, como COBOL, C++, JAVA entre outras linguagens de alto nível.

E temos os usuários finais que acessam o Banco de Dados usando uma linguagem própria para isso, o exemplo típico é o SQL (que veremos com maiores detalhes em outras unidades). Ou então, aqueles com conhecimento mais superficial fazem uso de interfaces acionado por comandos, ou seja, programas com formulários e menus que facilitam o trabalho do usuário final.



Dica

O nosso Glossário será super dinâmico!! Ou seja, toda vez que você tiver dúvida de alguma expressão técnica, ou algum termo desconhecido, deverá consultar o WIKIPÉDIA (<http://pt.wikipedia.org/>). Se tiver condições de ajudar nesse projeto, pois essa enciclopédia on-line é criada com as contribuições de todos os internautas, esteja à vontade e enriqueça essa Biblioteca Digital. Se mesmo assim, você ainda ficar com dúvidas, ou não conseguir achar o que você queria, então dirija para o nosso FÓRUM, intitulado GLOSSÁRIO, para que possamos ajudá-lo discutindo com o grupo.



Atividades

Pesquise no Wikipédia sobre Banco de Dados e SGBD, e responda, por escrito, as seguintes perguntas:

- Quais são os componentes básicos de um Sistema de Banco de Dados?
- O que é um SGBD?
- Mencione alguns exemplos de SGBDs



UNIDADE 2

Objetivo: Apontar e detalhar as principais vantagens de um Sistema de Gerenciamento de Banco de Dados.

Principais Características de um SGBD

A tecnologia aplicada aos métodos de armazenamento de informações vem crescendo e gerando um impacto cada vez maior no uso de Banco de Dados. Um Banco de Dados é, antes de tudo, uma coleção coerente de dados armazenados logicamente, cuja finalidade é organizar estas informações visando à otimização dos sistemas, facilitando a entrada, alterações, processamento e consulta de dados.

Para criação e manutenção de um Banco de Dados informatizado utiliza-se um **Sistema Gerenciador de Banco de Dados (SGBD)**. O conjunto formado por um Banco de Dados, o SGBD e mais as aplicações que o manipulam, é chamado de Sistema de Banco de Dados, conforme representado pela figura 2.1.

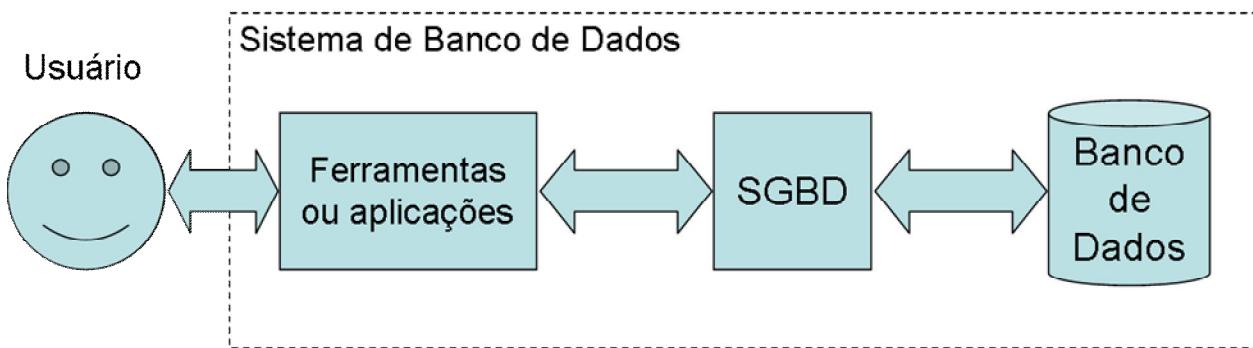


Figura 2.1 - Sistema de Banco de Dados

O acesso às informações em sistemas de processamento de dados que não utilizam SGBDs é feito pelo acesso sequencial a arquivos (um ou mais). Nesse caso, cabe ao desenvolvedor, criar mecanismos de recuperação da informação. No entanto, com a utilização de um SGBD o acesso é diferenciado. As informações são solicitadas ao Gerenciador do Banco de Dados

e devolvidas por ele mesmo. O gerenciador é que assume toda a responsabilidade de recuperar as informações desejadas.

As principais características ou vantagens de um SGBD são:

- Integridade;
- Consistência ou Compartilhamento de Dados;
- Segurança ou Restrição de Acesso;
- Restauração ou Tolerância a Falhas;
- Não Redundância ou Controle de Redundância;
- Padronização dos Dados.

Integridade

Consiste em assegurar que os dados no Banco de Dados estejam corretos, sempre permitindo que um código ou chave em uma tabela tenha correspondência adequada em outra tabela. Por exemplo, um código de uma determinada disciplina na tabela “Histórico Escolar” com a devida descrição na tabela “Disciplina”. Caso contrário, por exemplo, um empregado poderia ser mostrado como pertencendo a um departamento que não existe mais.

Consistência ou compartilhamento de dados

Por armazenar os dados em um único local, e compartilhando-os para vários sistemas, os usuários acabam utilizando a informação de forma muito mais confiável. Por outro lado, em sistemas inconsistentes ocorrem que um mesmo campo tem valores distintos pelos vários programas. Por exemplo, o estado civil de uma pessoa é “solteiro” em um sistema e “casado” em outro.

Isso ocorre com certa incidência em Sistemas sem SGBD, pois os usuários atualizam o campo em um sistema e não o fazem em outro. Quando o dado é armazenado em um único local e compartilhado pelos sistemas, esse problema é eliminado. Essas inconsistências também podem ser controladas por meio de regras estabelecidas no próprio Banco de Dados.

Segurança ou restrição de acesso

Define para cada usuário o nível de acesso à tabela e/ou ao campo (somente leitura, leitura e gravação ou sem acesso). Esse recurso impede que pessoas não autorizadas utilizem ou atualizem uma determinada informação. Por exemplo, num sistema bancário, o departamento pessoal necessita apenas do Banco de Dados as informações sobre os diversos empregados da empresa. Eles não necessitam, até por questão de segurança, ter acesso às informações sobre contas dos clientes do banco.

Restauração ou tolerância a falhas

Esta característica também mencionada como tolerância a falhas implica na característica que o SGBD deve apresentar como facilidade para recuperar falhas de hardware e software. A estratégia do SGBD para realizar isso é por meio da existência de arquivos de backup (cópia de segurança) ou de outros recursos automáticos.

Não redundância ou controle de redundância

A redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Em um Banco de Dados as informações se encontram armazenadas em um único local e não deve existir duplicação descontrolada dos dados. Os dados, que eventualmente são comuns a mais de um sistema, são compartilhados por eles, permitindo o acesso a uma única informação consultada pelos vários sistemas. Deve-se

observar apenas o processo de atualização concorrente, para não gerar erros de processamento (usuários atualizando simultaneamente o mesmo campo)

Às vezes, há motivos comerciais ou técnicos plausíveis para manter cópias distintas dos mesmos dados. Porém toda redundância deve ser cuidadosamente controlada; isto é, o SGBD deve estar ciente dela (caso exista) e deve garantir que qualquer mudança feita em uma das duas entradas também seja aplicada de forma automática a outra entrada. Esse processo é conhecido como propagação de atualizações.

Padronização dos dados

Permite que as informações da base de dados sejam padronizadas segundo um determinado formato de armazenamento (padronização de tabela, conteúdo de campos, etc.), e o nome de variáveis, segundo critérios preestabelecidos pelo analista. A padronização da representação dos dados é particularmente desejável quando ocorre a migração de dados entre sistemas.

Por exemplo: para o campo “sexo” somente será permitido como padrão o armazenamento dos conteúdos “M” ou “F”.

As desvantagens de um SGBD

Em algumas situações, cada vez mais raras, o uso de um SGBD pode representar uma carga desnecessária aos custos quando comparado ao processamento tradicional de arquivos como, por exemplo:

- Alto investimento inicial na compra de software e hardware adicionais;
- Generalidade que um SGBD fornece na definição e processamento de dados;
- Sobrecarga na provisão de controle de segurança, controle de concorrência, recuperação e integração de funções.

Também podemos considerar desvantagens de um SGBD quando os projetistas do Banco de Dados ou os administradores de Banco de Dados não elaborem os projetos corretamente ou se as aplicações não são implementadas de forma apropriada.



Dica

<http://www.plugmasters.com.br/sys/materias/108/1/SGBD---Sistema-Gerenciador-de-Banco-de-Dados>

Discussão sobre qual o melhor Banco de Dados?

<http://www.jack.eti.br/www/?p=38>



Estudo Complementar

Nesta unidade vimos as principais características de um SGBD, porém é interessante complementar seu estudo procurando saber quais são as vantagens e desvantagens de usar um SGBD free, ou seja, livres do pagamento da licença para uso do software, os mais conhecidos são: MySQL, PostgreSQL e SQLite. Em que casos é melhor um sistema proprietário do que pagar a licença?



UNIDADE 3

Objetivo: Indicar os principais aspectos das arquiteturas de um Sistema de Gerenciamento de Banco de Dados, abordando a arquitetura centralizada e a arquitetura cliente-servidor.

Conceitos e Arquiteturas de um SGBD

Atualmente, devem-se considerar alguns aspectos relevantes para atingir a eficiência e a eficácia dos sistemas informatizados desenvolvidos. Como se atende os usuários nos mais variados domínios de aplicação (automação de escritórios, sistemas de apoio a decisões, controle de reserva de recursos, controle e planejamento de produção, alocação e estoque de recursos, etc.), os aspectos a serem observados são:

Os projetos Lógico e Funcional do Banco de Dados devem ser capazes de prever o volume de informações armazenadas a curto, médio e longo prazo. Os projetos devem ter uma grande capacidade de adaptação;

Deve-se ter generalidade e alto grau de abstração de dados. Isso possibilitando ter confiabilidade e eficiência no armazenamento dos dados e permitindo a utilização de diferentes tipos de SGBDs através de linguagens de consultas padronizadas;

Projeto de uma interface ágil e com uma "rampa ascendente" para propiciar aprendizado suave ao usuário, no intuito de minimizar o esforço cognitivo;

Implementação desejável de um projeto de interface compatível com múltiplas plataformas (UNIX, Windows NT, Windows Workgroups, etc);

Independência de implementação da interface em relação aos SGBDs que darão condições às operações de armazenamento de informações (ORACLE, SYSBASE, INFORMIX, PADRÃO XBASE, etc).

Conversão e mapeamento da diferença semântica entre os paradigmas utilizados no desenvolvimento de interfaces (imperativo, Orientado a Objeto, Orientado a evento),

servidores de dados (relacional) e programação dos aplicativos (Imperativo, Orientado a Objetos).

Visão Geral das Arquiteturas

As primeiras arquiteturas usavam mainframes para executar o processamento principal e de todas as funções do sistema, incluindo os programas aplicativos, programas de interface com o usuário, bem como a funcionalidade dos SGBDs.

Esta é a razão pela qual a maioria dos usuários da época, que faziam acesso aos sistemas via terminais, não possuíam poder de processamento e sim a capacidade de visualização. Todos os processamentos eram feitos remotamente. Apenas as informações a serem visualizadas e controles eram enviadas do mainframe para os terminais de visualização, através de redes de comunicação. Com os preços do hardware decrescendo, os terminais foram substituídos por computadores pessoais (PC) e estações de trabalho.

No começo, os SGBDs usavam esses PCs da mesma maneira que usavam os terminais, o SGBD era centralizado e toda sua funcionalidade, execução de programas aplicativos e processamento da interface do usuário eram executados em apenas uma máquina. Gradualmente, os SGBDs começaram a explorar o poder de processamento do lado do usuário, o que levou à arquitetura cliente-servidor.

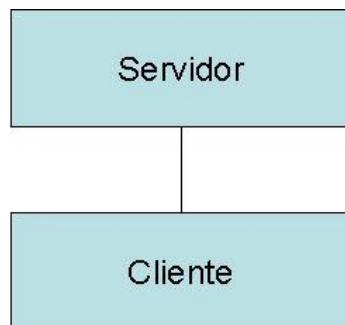


Figura 3.1 – Exemplo de Arquitetura Cliente-Servidor Simples

A arquitetura cliente-servidor foi desenvolvida para dividir ambientes de computação. A ideia é de definir servidores especializados, tais como servidor de arquivos, que mantém os arquivos de máquinas clientes, ou mesmo servidores de impressão que podem estar conectados a várias impressoras. Assim, quando se desejar imprimir algo, todas as requisições de impressão são enviadas a este servidor. As máquinas clientes disponibilizam para o usuário as interfaces apropriadas para utilizar esses servidores. Esta arquitetura é muito popular pelas seguintes razões:

A facilidade de implementação, com a clara separação das funcionalidades dos servidores.

Os servidores são mais inteligentemente utilizados porque as tarefas simples são delegadas às máquinas clientes mais baratas.

O usuário executa uma interface gráfica mais simples, ao invés de usar a interface do servidor que é mais complexa.

Diferentes técnicas foram propostas para se implementar essa arquitetura, sendo a mais adotada foi a inclusão da funcionalidade de um SGBD centralizado no servidor. As consultas e a funcionalidade transacional também ficam no servidor, sendo que este é chamado de servidor de consulta ou servidor de transação.

É assim que um servidor SQL é fornecido aos clientes. Cada cliente formula suas consultas SQL, provê a interface do usuário e as funções de interface usando a mesma linguagem de programação. Comumente o servidor SQL também é chamado de back-end e o cliente de front-end. Como SQL provê uma linguagem padrão para o SGBDs, esta criou o ponto de divisão lógica entre o cliente e o servidor.

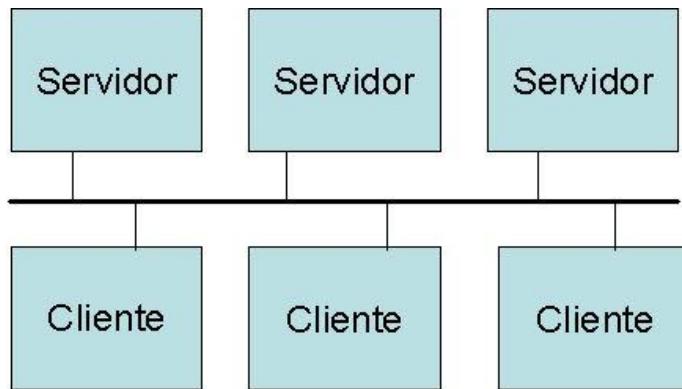


Figura 3.2 – Exemplo de Arquitetura Cliente-Servidor Bi-Nível



Estudo Complementar

http://www.sebraepb.com.br:8080/bte/download/informatica/190_1_arquivo_bdados.pdf

<http://pt.wikipedia.org/wiki/Servidor>



Atividades

Responda por escrito as seguintes perguntas:

O que ajudou muitas empresas a migrarem de uma arquitetura centralizada (Mainframe) para uma arquitetura baseada em computadores pessoais (PC)?

Cite algumas razões que fez a arquitetura cliente-servidor ganhar popularidade?

Numa arquitetura cliente-servidor quais são algumas das funcionalidades do servidor e do cliente?

Que tipos diferentes da arquitetura cliente-servidor existem?



UNIDADE 4

Objetivo: Mostrar as principais arquiteturas de um Sistema de Gerenciamento de Banco de Dados e o seu desenvolvimento.

Características das Principais Arquiteturas de um SGBD

Atualmente, existem várias tendências para arquitetura de Banco de Dados, nas mais diversas direções. Faremos aqui um resumo das principais arquiteturas do ponto de vista histórico.

Primeira Arquitetura: Plataformas Centralizadas (uso de Mainframes)

Na arquitetura centralizada, existe um computador com grande capacidade de processamento, o qual é o hospedeiro do SGBD e emuladores para os vários aplicativos. Esta arquitetura tem como principal vantagem a de permitir que muitos usuários manipulem grande volume de dados. Sua principal desvantagem está no seu alto custo, pois exige ambiente especial para mainframes e soluções centralizadas. Suas principais características são:

O processamento principal e de todas as funções do sistema (aplicativos, interface e SGBD) são executados no mainframe.

Os usuários interagiam com o sistema, via terminais, sem poder de processamento, conectados ao mainframe por redes de comunicação.

Com o barateamento do hardware, os terminais foram sendo trocados por estações de trabalho e naturalmente a tecnologia de Banco de Dados começou a aproveitar esse potencial de processamento no lado do usuário.

Segunda Arquitetura: Cliente-Servidor

Na arquitetura Cliente-Servidor, o cliente (*front-end*) executa as tarefas do aplicativo, ou seja, fornece a interface do usuário (tela, e processamento de entrada e saída). O servidor (*back-end*) executa as consultas no SGBD e retorna os resultados ao cliente. Apesar de ser uma arquitetura bastante popular, são necessárias soluções sofisticadas de software que possibilitem: o tratamento de transações, as confirmações de transações (*commits*), desfazer transações (*rollbacks*), linguagens de consultas (*stored procedures*) e gatilhos (*triggers*). A principal vantagem desta arquitetura é a divisão do processamento entre dois sistemas, o que reduz o tráfego de dados na rede. São características dessa arquitetura:

Divisão das tarefas de processamento criando servidores especializados como os servidores de arquivos.

As máquinas clientes disponibilizavam as interfaces para os usuários, de forma a capacitá-lo ao uso de servidores. Também tinham autonomia para executar aplicações locais.

Um SGBD centralizado é implantado no servidor, permitindo que as consultas (servidor SQL) e funcionalidades transacionais sejam executadas nesse servidor.

No lado do cliente é possível personalizar as consultas e desenvolver programas aplicativos específicos.

Terceira Arquitetura: Sistemas em Computadores Pessoais

Os computadores pessoais trabalham em sistema stand-alone, ou seja, fazem seus processamentos sozinhos. No começo, esse processamento era bastante limitado, porém, com a evolução do hardware, os PCs foram ganhando grande capacidade de processamento. Eles utilizam o padrão Xbase e quando se trata de SGBDs, funcionam como hospedeiros e terminais. Desta maneira, possuem um único aplicativo a ser executado na máquina. Suas características são:

Trabalham no sistema stand-alone, executando sozinhos todas as funções necessárias para o funcionamento do SGBD.

Principal vantagem desta arquitetura é a simplicidade.

Aplicações típicas são de baixa e média complexidade.

Quarta Arquitetura: Distribuída (N camadas)

Nesta arquitetura, a informação está distribuída em diversos servidores. Como exemplo, observe na figura 4.1 abaixo. Cada servidor atua como no sistema cliente-servidor, porém as consultas oriundas dos aplicativos são feitas para qualquer servidor indistintamente. Caso a informação solicitada seja mantida por outro servidor ou servidores, o sistema encarrega-se de obter a informação necessária, de maneira transparente para o aplicativo.

Exemplos típicos são as bases de dados corporativas, em que o volume de informação é muito grande e, por isso, deve ser distribuído em diversos servidores. A característica básica é a existência de diversos programas aplicativos consultando a rede para acessar os dados necessários, porém, sem o conhecimento explícito de quais servidores dispõem desses dados. Vejamos suas principais características:

- Os dados e o processamento são distribuídos por diversos servidores (ou hosts).
- Cada host pode atuar como um servidor de um sistema cliente-servidor, e como cliente.
- Muito usado em bases de dados corporativas, ou em aplicações sofisticadas, onde o volume de informações seja muito grande.
- Desvantagem: aumento da complexidade de gerenciamento.

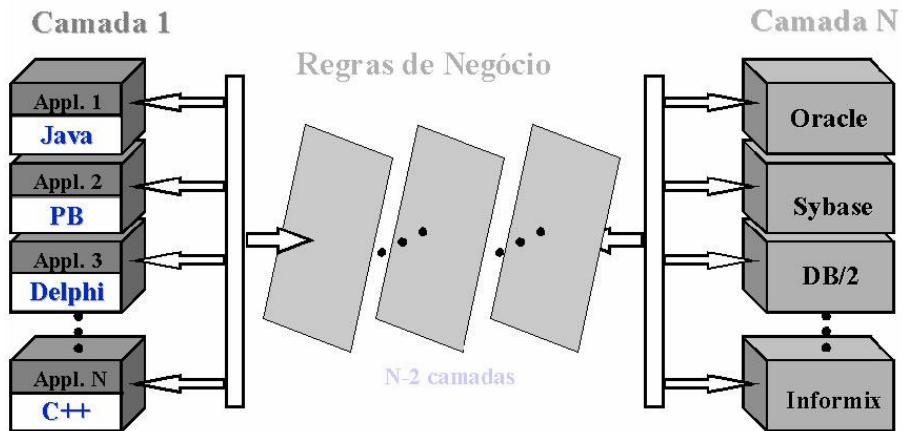


Figura 4.1 – Arquitetura Distribuída N Camadas

Quinta Arquitetura: Paralela

Combinam técnicas de gerência de dados e processamento paralelo para aumentar desempenho e confiabilidade. A arquitetura paralela vem tornando-se uma tendência em função da demanda sempre crescente por poder computacional. Os sistemas que oferecem essa capacidade de processamento, ainda têm um custo muito elevado ou são difíceis de programar. O estudo de arquiteturas paralelas contribui para o entendimento e para a busca de alternativas para as outras arquiteturas. São suas características:

- O processamento do sistema utiliza as técnicas de paralelismo.
- Computadores multiprocessados, ou vários computadores, são utilizados para o processamento paralelo de uma única transação.
- A paralelização do processamento interno de consultas resulta numa diminuição do tempo de resposta.
- A paralelização do processamento de transações resulta num aumento da capacidade do sistema (throughput).
- Desvantagem: custo e complexidade alta de gerenciamento.

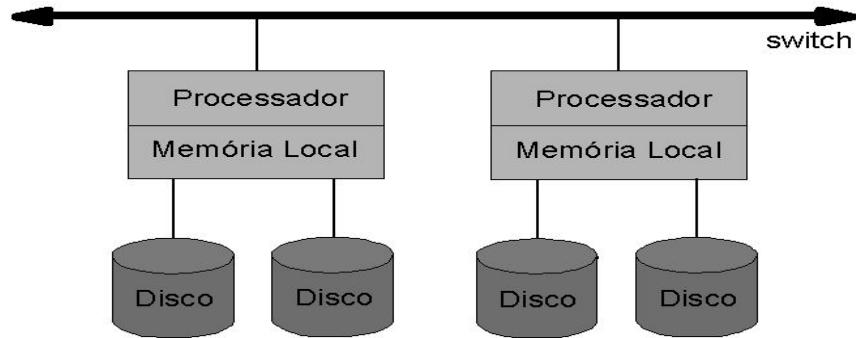


Figura 4.2 – Arquitetura de SGBDs Paralelos



Estudo Complementar

http://pt.wikipedia.org/wiki/Banco_de_dados_distribu%C3%A7%C3%A3o

http://pt.wikipedia.org/wiki/N_camadas

<http://www.juliobattisti.com.br/artigos/ti/ncamadas.asp>

<http://www.inf.puc-rio.br/~casanova/LivroCasanova/ncap1.pdf>

<http://www.inf.puc-rio.br/~casanova/LivroCasanova/ncap2.pdf>



Atividades

Visite os links referenciados anteriormente e responda as seguintes perguntas:

Quais são alguns exemplos de arquiteturas de SGDBs?

Qual tipo de arquitetura é a mais utilizada nas grandes empresas?



UNIDADE 5

Objetivo: Detalhar os níveis de abstração dos dados na Arquitetura de um Sistema de Gerenciamento de Banco de Dados.

Níveis de Abstração dos Dados na Arquitetura de um SGBD

Como já vimos o SGBD é composto de uma coleção de arquivos inter-relacionados e um conjunto de programas que permitem aos usuários fazerem o acesso e a modificação desses arquivos. O grande objetivo desse sistema é prover aos usuários uma visão abstrata dos dados. Isto é, o sistema omitir certos detalhes de como os dados são armazenados e mantidos, mantendo total transparência para os usuários.

Este conceito tem direcionado o projeto de estruturas de dados complexas para a representação desses dados. Existem diversos níveis de abstração que simplificam a interação do usuário com o sistema: Nível Físico, Nível Conceitual e o Nível de Visões. O inter-relacionamento entre estes três níveis de abstração é ilustrado na Figura 3.1:

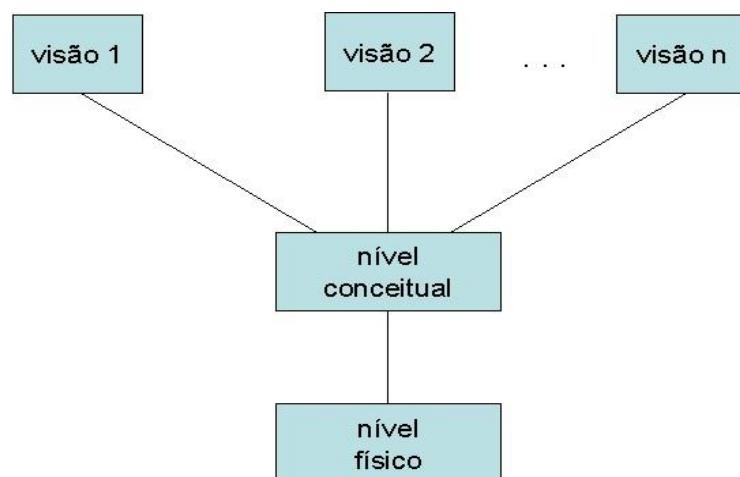


Figura 3.1 - Os três níveis de abstração de dados

Nível Físico, ou Nível Interno (também conhecido como nível de armazenamento)

O nível mais baixo de abstração descreve como os dados estão realmente armazenados. No nível físico, complexas estruturas de dados de baixo nível são descritas em detalhes. Este nível é o mais próximo do meio de armazenamento físico, ou seja, é aquele que se ocupa do modo como os dados são fisicamente armazenados no sistema, tais como: tamanho de campos, índices, como os campos armazenados estão representados, em que sequência física os registros serão armazenados, e assim por diante.

Nível Conceitual, ou Nível Lógico (também conhecido como nível lógico de comunidade)

Este nível de abstração descreve quais dados estão armazenados de fato no Banco de Dados e as relações que existem entre eles. Aqui, o Banco de Dados é descrito em termos de um pequeno número de estruturas relativamente simples. Embora a implementação de estruturas simples no nível conceitual possa envolver complexas estruturas de nível físico, o usuário do nível conceitual não precisa preocupar-se com isso.

O nível conceitual de abstração é usado pelos administradores de Banco de Dados, para decidir quais informações devem ser mantidas no Banco de Dados. Este nível está preocupado com a percepção da comunidade de usuários, ou seja, é uma visão do conteúdo total do Banco de Dados.

Nível de Visões, ou Nível Externo (também conhecido como nível lógico do usuário)

O mais alto nível de abstração descreve apenas uma visão limitada do Banco de Dados. Muitos usuários do SGBD não estão interessados em todos os dados, em vez disso, precisam apenas uma parte do Banco de Dados.

O nível de abstração das visões dos dados é definido para simplificar esta interação com o sistema, que pode fornecer muitas visões para o mesmo Banco de Dados. Apesar do uso de

estruturas mais simples do que no nível conceitual, alguma complexidade perdura nesse nível.

Conclusões Gerais dos Níveis de Abstração dos Dados

Observe que o Nível de Visões e o Conceitual são níveis de modelo, enquanto que o Nível Físico é um nível de implementação. Em outras palavras, o Nível de Visões e o Conceitual são definidos em termos de construções voltadas para o usuário, como registros e campos, enquanto o Nível Físico é definido em termos de construções voltadas para a máquina, como bits e bytes.

Enquanto o Nível de Visões se preocupa com as percepções dos usuários individuais, o Nível Conceitual está preocupado com a percepção da comunidade dos usuários. A maior parte dos usuários não está interessada no Banco de Dados inteiro, mas somente em alguma parte restrita dele.

Assim, haverá muitas “visões externas” distintas, cada qual consistindo em uma representação mais ou menos abstrata das partes de um Banco de Dados.



Estudo Complementar

<http://www.criarweb.com/artigos/arquitetura-bancos-de-dados.html>

<http://www.linhadecodigo.com.br/Artigo.aspx?id=332>





Atividades

Com base do link abaixo da UFRJ, que contém uma apresentação sobre Arquitetura de Banco de Dados, realize um pequeno resumo desse material:

<http://www.cos.ufrj.br/~marta/ArchitectureP.pdf>



UNIDADE 6

Objetivo: Descrever a formação e as características de cada usuário que trabalha com SGBD

Quem são os usuários de um SGBD?

Para um sistema de Banco de Dados aplicado numa média e complexa atividade, existe um grande número de pessoas envolvidas, desde o projeto até a manutenção propriamente dito. Entre estes, podemos destacar os seguintes profissionais:

Administrador de Dados

O grande objetivo do administrador de dados é permitir que vários usuários compartilhem os dados corporativos. Deste modo, os dados não pertencem a nenhum sistema ou usuário de forma específica, e sim, à organização como um todo. Assim, o administrador de dados se preocupa basicamente com a organização dos dados, e não com o seu armazenamento propriamente dito. Vejamos, suas características:

- Gerenciar o dado como um recurso da empresa.
- Planejar, desenvolver e divulgar as bases de dados da empresa.
- Permitir a descentralização dos processos, mas manter centralizado os dados.
- Permitir, fácil e rápido acesso às informações a partir dos dados armazenados.

Administrador de Banco de Dados (DBA)

Em qualquer organização que compartilha muitos recursos computacionais, existe a necessidade de um administrador para gerenciar esses recursos. Em um ambiente de Banco

de Dados, o recurso primário é o próprio Banco de Dados e o recurso secundário é o SGBD (e os recursos relacionados).

O Administrador de Banco de Dados é o responsável pela autorização de acesso ao Banco de Dados e pela coordenação e monitoração de seu uso. É a pessoa que, numa equipe de desenvolvimento, centraliza tanto o controle dos dados quanto os programas de acesso a eles. É conhecido com a sigla em inglês: DBA (DataBase Administrator).

O DBA é também responsável pelos problemas de quebra de segurança ou de baixo desempenho nos SGBDs. As principais funções do DBA são:

- Definição do esquema do Banco de Dados;
- Definição da estrutura de dados e métodos de acesso;
- Modificações no esquema ou na organização física;
- Controle das autorizações de acesso ao sistema;
- Especificação das regras de integridade.

Projetista de Banco de Dados (DB Designer)

O Projetista de Banco de Dados é responsável pela identificação dos dados que devem ser armazenados no Banco de Dados. Ele escolhe a estrutura mais adequada para representar e armazenar esses dados. É função do projetista também avaliar as necessidades de cada grupo de usuários. Muitas vezes, os projetistas de Banco de Dados atuam como “staff” do DBA, assumindo outras responsabilidades após a construção do Banco de Dados.

Usuários Finais

Existem basicamente quatro categorias de usuários de Banco de Dados, que fazem operações mais básicas nos SGBD, tais como consultas, atualizações e geração de documentos:

Usuários Casuais: acessam o Banco de Dados casualmente, mas que podem necessitar de diferentes informações a cada acesso. Utilizam normalmente sofisticadas linguagens de consulta para especificar suas necessidades;

Usuários Novatos ou Paramétricos: utilizam visões do Banco de Dados, utilizando consultas preestabelecidas que já foram exaustivamente testadas. São também chamados de usuários navegantes, ou seja, usuários comuns que interagem com o sistema através de interfaces pré-definidas;

Usuários Sofisticados: são usuários que estão familiarizados com o SGBD e realizam consultas mais complexas;

Usuários Especialistas: usuários sofisticados que chegam a escrever aplicações especializadas.

Analistas de Sistemas e Programadores de Aplicações

Os analistas de sistemas determinam os requisitos dos usuários finais e desenvolvem especificações para transações que atendam estes requisitos. Os programadores de aplicações implementam estas especificações com os programas, testando, depurando, documentando e dando manutenção aos mesmos. São profissionais em computação que interagem com o sistema por meio de DMLs, envolvidas em programas escritos em diferentes linguagens hospedeiras.

Profissionais de Apoio

Profissionais que auxiliam e apoiam a todos os outros. São projetistas e implementadores de SGBD, desenvolvedores de ferramentas, e também operadores de manutenção.



Estudo Complementar

Observe no link abaixo, no Capítulo II, as atribuições do Administrador de Banco de Dados:

http://pt.wikipedia.org/wiki/Administrador_de_banco_de_dados



Atividades

Procure saber qual a necessidade no mercado de trabalho de cada um dos profissionais mencionados, e qual a remuneração média deles.

Verifique nos fornecedores de Banco de Dados quais cursos e treinamentos irão aumentar e especializar seus conhecimentos nessa área.



UNIDADE 7

Objetivo: Apresentar os diversos modelos físicos e lógicos existentes em Banco de Dados.

Modelagem de Dados I: Modelos Lógicos baseado em Registros

Com a evolução dos processos, surgiram novas formas de modelar os Bancos de Dados, visando a suprir as necessidades das empresas e a maior eficácia no processo de armazenamento das informações. Na Modelagem de Dados podemos dividir em três grupos diferentes:

- Modelos Físicos de Dados
- Modelos Lógicos Baseados em Registros
- Modelos Lógicos Baseados em Objetos (veremos na Unidade 8)

Modelos Físicos de Dados

Os modelos físicos de dados são usados para descrever dados no nível mais baixo. Em comparação com os modelos lógicos de dados, existem poucos modelos físicos em uso. Dois dos mais conhecidos são: Modelo Unificador (unitying model) e Estrutura de Memória (frame memory). Para os nossos objetivos é mais importante conhecer com maior riqueza de detalhes os modelos lógicos.

Modelos Lógicos Baseados em Registros

Esses Modelos Lógicos são usados nas descrições de dados no Nível conceitual e no de Visões. Igualmente como os Modelos de Dados Baseado em Objetos, ambos são usados

para especificar a estrutura lógica geral do Banco de Dados e para fornecer uma descrição de alto nível da implementação.

Modelos baseados em registros são assim chamados porque o Banco de Dados é estruturado em registros de formato fixo. Cada tipo de registro define um número fixo de campos, ou atributos, e cada campo são usualmente de tamanho fixo. Com esse conceito surgiram os seguintes modelos de Banco de Dados: Hierárquico, Rede e Relacional.

Desses três, o Modelo Relacional é o que tem maior penetração no mercado em relação aos outros dois. O Modelo Hierárquico e o de Rede eram usados em um grande número de Banco de Dados antigos, e em algumas aplicações na Internet.

O Modelo Hierárquico

O modelo hierárquico surgiu na década de 60. Permite organizar dados em uma estrutura hierárquica (estrutura em árvore invertida), com acesso unidirecional, de pai para o filho, sempre começando pela raiz.

Em outras palavras, esse tipo de Banco de Dados é formado por uma coleção de registros conectada uns aos outros por links. Os SGBDs mais conhecidos desse tipo são o IMS (Information Management System da IBM), Adabas e o System2000.

Os dados organizados, segundo este modelo, podem ser acessados segundo uma sequência hierárquica com uma navegação do topo para as ramificações. Um registro pode estar associado a vários registros diferentes, desde que seja replicado. A replicação possui grandes desvantagens causando inconsistência nos dados e o desperdício de espaço.

Na Figura 7.1, vemos exemplo de um Banco de Dados no Modelo Hierárquico. Exibem-se dados como: Nome, Cidade, UF, Conta e Saldo. A conta corrente de dois clientes, José Silva e Maria Silva mantém uma conta em comum (conta: 20343). Por usar o Modelo Hierárquico, notamos que a conta em comum é replicada, apresentando as desvantagens apresentadas anteriormente.

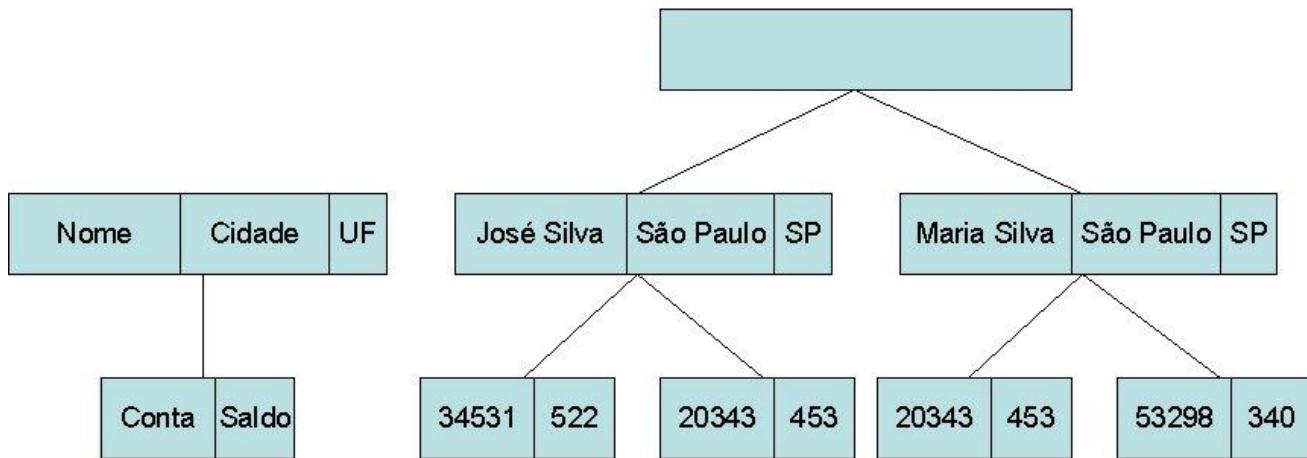


Figura 7.1 Exemplo de modelo hierárquico

O Modelo de Rede

Esse modelo foi utilizado principalmente no final da década de 60 e durante a década de 70. Ele permite organizar os dados em uma estrutura formada por várias listas, que definem uma intrincada rede de ligações (estrutura similar a um grafo direcionado). O IDMS e o Total são os SGBDs mais conhecidos com essa estrutura.

Esse Modelo de Rede é, essencialmente, um conjunto ilimitado de nós e de ramais de ligação. Na verdade, uma hierarquia é apenas um tipo particular de rede, pois uma rede não apresenta o conceito de nó raiz e os registros podem ter diversos tipos de registros-pai, assim como diversos tipos de registros-filho.

Similar ao Modelo Hierárquico, os dados no modelo de rede são organizados em tipos de registros e ligações entre registros. Não existe restrição hierárquica, ou seja, quaisquer dois tipos de registros podem se relacionar. Um esquema no Modelo de Rede é chamado de Diagrama de Estrutura de Dados.

Vejamos um exemplo na Figura 7.2, onde utilizamos os mesmo dados do exemplo anterior, porém adaptado ao Modelo de Rede. Observamos que não existe mais o conceito de raiz, e também não ocorre a replicação do registro. A principal desvantagem dessa abordagem é

que, se o Banco possuir muitos tipos de entidades, pode resultar em esquemas muito complexos de relacionamentos.

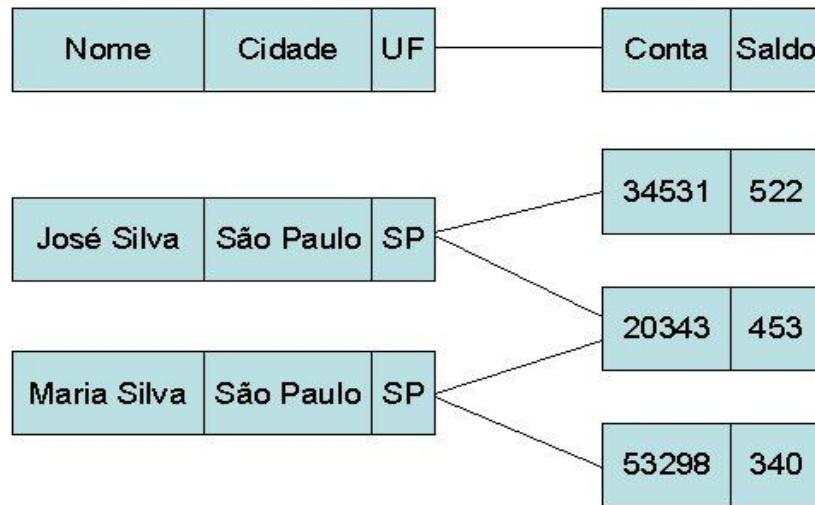


Figura 7.2 Exemplo de modelo de rede

O Modelo Relacional

O modelo relacional foi formalmente definido por E. Codd, do Laboratório da IBM em San Jose, Califórnia, em 1970. O projeto inicial foi denominado de Sistema R e definia a organização dos dados e linguagens formais para a sua manipulação.

Com base nessas linguagens, foi definida a primeira versão do SQL (Structured Query Language). Essa linguagem é o padrão em SGBD relacionais. Os Bancos de Dados Relacionais mais famosos são: Oracle, SQL Server, Informix, Sybase e Ingres.

O objetivo básico tratado pelo modelo relacional é entidade ou relação. Uma entidade equivale ao conceito matemático de conjunto, ou seja, um agrupamento de elementos.

Um Banco de Dados Relacional visa manter os dados de forma não redundante (repetição de vários campos em várias tabelas), executar processamento integrado, lidar com relações múltiplas (relacionamentos) e fornecer certo grau de independência dos dados.

O modelo relacional, de longe, é o mais utilizado, pois é o modelo que se obtém o maior desempenho e agilidade. Desde pequenas a grandes empresas e instituições utilizam, com segurança, gigantes bases relacionais para gerenciar e manter seus dados.

Novamente, usando o mesmo exemplo, vamos ilustrar na Figura 7.3 como que no modelo relacional são representados os dados. O relacionamento existente entre os dados foi realizado através de um conjunto de 3 tabelas.

A primeira com os dados cadastrais do cliente, a segunda relacionando as contas de cada cliente, e finalmente na última tabela o saldo de cada conta.

Cliente	Nome	Cidade	UF
1	José Silva	São Paulo	SP
2	Maria Silva	São Paulo	SP

Cliente	Conta	Conta	Saldo
1	34531	34531	522
1	20343	20343	453
2	20343	53298	340
2	53298		

Figura 7.3 - Exemplo de modelo relacional



Estudo Complementar

http://pt.wikipedia.org/wiki/Modelo_Hierárquico

http://pt.wikipedia.org/wiki/Modelo_em_rede

http://pt.wikipedia.org/wiki/Modelo_Relacional

<http://www.linhadecodigo.com.br/Artigo.aspx?id=396>

<http://www.plugmasters.com.br/sys/materias/586/1/Modelagem-de-Dados:A-Hierarquias---Parte-1>

<http://www.plugmasters.com.br/sys/materias/587/1/Modelagem-de-Dados:A-Hierarquias---Parte-2>



Atividades

Com base nessa Unidade, responda por escrito:

Que exemplos de modelo lógico, baseado em registros, podemos mencionar?

Qual é o modelo de Banco de Dados mais utilizado atualmente?

Quais são as vantagens do modelo relacional de Banco de Dados?



UNIDADE 8

Objetivo: Apresentar os vários modelos lógicos de Banco de Dados baseado em Objetos

Modelagem de Dados II: Modelos Lógicos baseado em Objetos

Na descrição de dados no Nível Conceitual e no de Visões, são usados Modelos Lógicos baseados em Objetos. Eles se caracterizam pela capacidade de formar estruturas flexíveis, com restrições de dados que podem ser explicitamente especificados. Existem muitos modelos diferentes, alguns dos mais conhecidos são:

- O Modelo Entidade-Relacionamento (MER);
- O Modelo Orientado a Objeto (OO);
- O Modelo Binário;
- O Modelo Semântico de Dados;
- O Modelo Infológico;
- O Modelo Funcional de Dados.

Nesta apostila iremos considerar apenas o Modelo Entidade-Relacionamento e o Modelo Orientado a Objeto, por serem os maiores representantes da classe de Modelos Lógicos baseados em Objetos.

O Modelo Entidade-Relacionamento (MER) tem ganhado bastante aceitação nos projetos de Banco de Dados, e é muito utilizado na prática. O Modelo Orientado a Objeto (OO) inclui muito dos conceitos do próprio MER, mas representa os códigos executáveis, assim como os dados, de forma diferente e está ganhando rápida aceitação.

Modelo Entidade-Relacionamento (MER)

O MER é baseado na percepção do mundo real numa coleção de objetos básicos chamados entidades, e nos relacionamentos entre esses objetos. Uma entidade é um objeto que é distingível de outro, por um conjunto específico de atributos. O relacionamento é uma associação entre várias entidades. Por exemplo, um relacionamento Conta-Cliente associa um cliente a cada conta que ele possui.

A estrutura lógica geral de um Banco de Dados pode ser expressa graficamente por um Diagrama Entidade-Relacionamento (DER), que consiste nos seguintes componentes:

- Retângulos que representam conjunto de entidades;
- Elipses que representam atributos;
- Losangos que representam relacionamentos entre conjuntos de entidades;
- Linhas que interligam atributos, conjuntos de entidades e relacionamentos.

A diferença entre MER e DER é que o Modelo Entidade-Relacionamento é conceitual, e o Diagrama Entidade-Relacionamento mostra graficamente a relação existente entre as entidades (tabelas). Exemplo: Em um MER você pode ter uma relação ‘n’ pra ‘n’ entre duas entidades, já em um DER nunca! Isto claramente geraria uma terceira tabela pelas regras de normalização.

No MER são representadas as entidades, dando uma ideia geral sobre como vai ser o Banco de Dados. Já no DER é representado graficamente o próprio Banco de Dados, com suas tabelas, relacionamentos, regras, restrições, tipos de dados, chaves primárias, etc...

Para ilustrar melhor, considere o sistema de Banco de Dados de uma instituição bancária tal como fizemos até o momento. O Diagrama Entidade-Relacionamento (DER) correspondente é mostrado na Figura 8.1 abaixo.

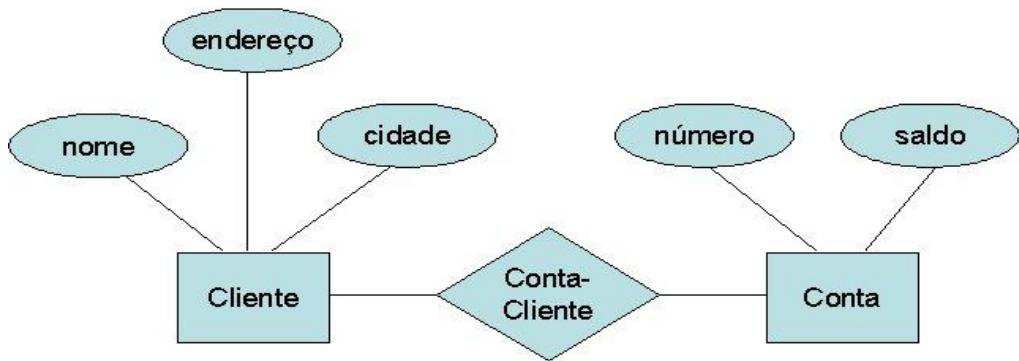


Figura 8.1 - Exemplo de um Diagrama Entidade-Relacionamento (DER)

Modelo Orientado a Objeto

Os Bancos de Dados Orientados a Objeto começaram a se tornar comercialmente viáveis em meados de 1980. A motivação para seu surgimento foi em função dos limites de armazenamento e representação semântica impostas no modelo relacional. Alguns exemplos são os Sistemas de Informações Geográficas (SIG), e os sistemas CAD e CAM, que são mais facilmente construídos usando tipos complexos de dados. A habilidade para criar esses tipos de dados é uma característica das linguagens de programação Orientadas a Objetos.

Assim como o MER, o Modelo Orientado a Objeto é baseado num conjunto de objetos. Um objeto contém valores armazenados em variáveis de instância dentro do próprio objeto. Contrariamente ao modelo a registros, estes valores são os objetos em si. Um objeto também possui trechos de código que operam nele mesmo. Estes trechos de códigos são chamados métodos. Os objetos que contêm os mesmos tipos de valores e os mesmos métodos são agrupados em classes. Uma classe pode ser vista como uma definição de tipo de objetos. O único modo pelo qual um objeto pode fazer o acesso ao dado de outro objeto é invocando o método desse outro objeto.

O termo Modelo Orientado a Objetos é usado para documentar o padrão que contém a descrição geral das facilidades de um conjunto de linguagens OO e a biblioteca de classes que formam a base para o SGBD. Quando os Bancos de Dados Orientados a Objetos foram introduzidos, algumas das falhas perceptíveis do Modelo Relacional pareceram ter sido

solucionadas, e acreditava-se que tais BD ganhariam grande parcela do mercado. Hoje, porém, acredita-se que os Bancos de Dados Orientados a Objetos somente devem ser utilizados em aplicações especiais, e os Sistemas Relacionais continuarão a sustentar os negócios tradicionais, onde as estruturas de dados baseadas em relações são suficientes.

O Diagrama de Classes da UML (Unified Modeling Language) serve geralmente de esquema para o Modelo de Dados Orientado a Objetos. Observe o exemplo da Figura 8.2, e compare as diferenças com o modelo anterior.

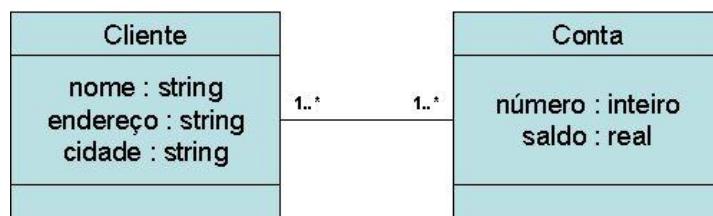


Figura 8.2 - Diagrama UML Cliente - Conta



Estudo Complementar

http://pt.wikipedia.org/wiki/Modelo_de_Entidades_e_Relacionamentos

http://pt.wikipedia.org/wiki/Diagrama_entidade_relacionamento

<http://www.unipan.br/emerson/Engenharia/DER.doc>

<http://www.plugmasters.com.br/sys/materias/453/1/Modelagem-de-Dados-4---Validação-do-modelo-ER>

Veja também o artigo “Uma Nova Era na Tecnologia dos Bancos de Dados” no site abaixo:

<http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=68>





Atividades

Faça um Diagrama Entidade-Relacionamento de um sistema de Banco de Dados de uma escola, usando as seguintes entidades Aluno, Professor e Disciplina. E preste atenção nos relacionamentos existentes entre cada entidade.



UNIDADE 9

Objetivo: Definir e exemplificar os conceitos de entidade, atributo e tupla

Definição de Entidade, Atributo e Tupla

Para construirmos uma base de dados, é necessário um alicerce, que será a estrutura do Banco de Dados. Essa estrutura é chamada de entidade e, na prática, conhecida como tabela. Para facilitar, vamos fazer uma analogia com uma agenda de dentista, que pode ser considerado um simples Banco de Dados.

Em uma agenda de dentista convencional, podemos encontrar pelo menos duas entidades, sendo uma o paciente (com o nome e telefone) e a outra os horários (com o dia e a hora da consulta). Note que o tipo de dado em cada entidade pode ser diferente, mas é de mesma natureza. A entidade não é o conjunto dos dados em si, mas sim os registros, que veremos a definição em seguida. A entidade, ou tabela, é a estrutura na qual foi elaborada a agenda (quais informações devem ser anotadas, a forma, o espaço reservado para cada telefone, etc).

A entidade é um objeto de um Banco de Dados (BD). A ela pertencem os atributos. No exemplo do BD Agenda, ela possui duas entidades, ou tabelas: Paciente (com os Atributos Nome e Telefone) e a Entidade Horário (com os atributos Data e Hora). Um atributo, ou campo, é um conjunto de características de um registro.

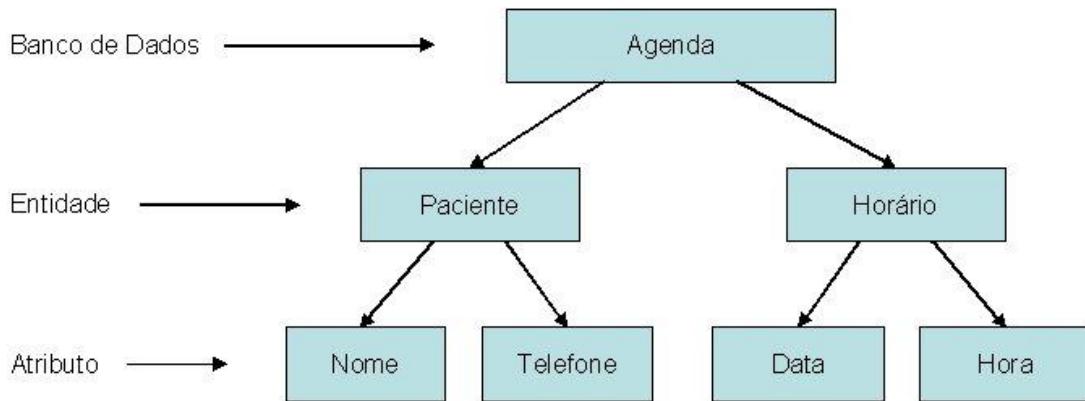


Figura 9.1 - Esquema de um Banco de Dados

O elemento principal de um Banco de Dados é a tabela (entidade). Análogo a uma planilha com colunas e linhas (ver Figura 9.2 a seguir), a tabela é um objeto do Banco de Dados, constituída de colunas (campos ou atributos) e linhas que são os dados em si (registros ou tuplas).

Na Figura 9.2 abaixo, podemos observar uma tabela (entidade) que possui os campos (atributos): Código, Nome, Endereço e Telefone. O campo deve definir os parâmetros para a inclusão de um registro. Podemos ainda observar a existência de três registros (ou tuplas), ou seja, três pessoas cadastradas, cada uma com os seus respectivos dados definidos pelo campo.

Código	Nome	Endereço	Telefone
1	José da Silva	Alameda Santos, 459	6456-6565
2	Paulo Santos	Rua Pedroso, 368	3177-6832
3	Ricardo Souza	Av. Paulista, 1200	9632-1005

Figura 9.2 – Exemplo de entidade e atributos

Nomenclatura de Campos

Você pode observar que nos exemplos anteriores, desobedecemos algumas regras de ortografia. Isso se deve ao fato de que os campos devem respeitar alguns padrões em termos de nomenclatura. Esses padrões não são sempre obrigatórios. Mas, recomenda-se observar as seguintes regras:

- Não utilizar caracteres especiais (@,#,\$,%...), exceto o underline (“_”);
- Evitar acentos e cedilhas;
- Nome do campo deve começar com uma letra maiúscula e não com número;
- Evitar o uso de nomes compostos de duas ou mais palavras com espaços em branco.

Exemplo de nomes de campos recomendados:

- Codigo (sem acento)
- Endereco (sem cedilha)
- NomeDoCliente ou Nome_Cliente (campos sem espaços em branco)
- Acessorio1
- Acessorio2



Estudo Complementar

Procure saber qual a necessidade no mercado de trabalho de cada um dos profissionais mencionados, e qual a remuneração média deles.

Verifique nos fornecedores de Banco de Dados quais cursos e treinamentos irão aumentar e especializar seus conhecimentos nessa área.





Dica

É interessante, para maior organização, definir-se um padrão de nomenclatura. Em um Banco de Dados pequeno pode parecer exagero, mas quando trabalharmos com uma base de dados maior, isso vai ser de grande ajuda. Você pode, por exemplo, definir quatro dígitos para o nome do campo e mais três dígitos para o nome da tabela.

Exemplos: Codi_cli (campo Codi da tabela cli), Nome_cli (campo Nome da tabela cli), Desc_pro, Fone_fun, e assim por diante.



Tipos de Dados

Você pode observar na figura 9.2, que o tipo de dado computado no campo “Nome” é bem diferente do tipo de dado computado no campo “Telefone”. Seria incorreto generalizar os campos. Cada campo deve conter um tipo de dado específico conforme as características que ele receberá.

Nesse caso, os campos “Nome” e “Endereço” podem ser do tipo alfanumérico. E o campo “Telefone” poderá ser do tipo numérico. Observe que o campo “Nome” contém menos caracteres (letras) que o campo “Endereço”. O tamanho dos campos também deve ser avaliado individualmente. Vamos supor um campo que contenha a sigla de um Estado do Brasil; ele somente precisará de dois dígitos.

Cada SGBD introduz tipos de valores de campo que não necessariamente são padrões. Entretanto, existe um conjunto de tipos de dados que são representados praticamente na totalidade destes bancos. Estes tipos de dados comuns são os seguintes:

Alfanuméricos	Contém letras e números. Tamanho limitado normalmente a 255 caracteres.
---------------	---

Numéricos	Normalmente com a possibilidade de inserir números inteiros (sem decimais) ou reais (com decimais).
Booleanos (ou Binários)	Permitem somente dois valores (Verdadeiro e Falso, Sim ou Não, ou ainda 0 e 1).
Datas	Armazena datas possibilitando ordenar os registros cronologicamente ou calcular os dias entre uma data e outra.
Memos	São campos alfanuméricos de tamanho ilimitado. Não podendo ser indexado (veremos mais adiante essa definição).
Autoincrementáveis	São numéricos inteiros que são incrementados automaticamente em uma unidade para cada registro incorporado. Ótimo para chaves.



Dica

Se tiver dificuldades para criar uma tabela no MS-Access, siga o tutorial abaixo que explicará passo a passo:

<http://www.apoioinformatica.inf.br/sgbd/sgbd1.htm>

<http://www.apoioinformatica.inf.br/sgbd/sgbd2.htm>

Por exemplo, veja os vários Tipos de Campos usados no MySQL no site abaixo:

<http://dev.mysql.com/doc/refman/4.1/pt/column-types.html>





Atividades

Procure no MS-Access criar uma tabela e preste atenção nos Tipos de Dados usados no Access quando for criar os campos de uma tabela.



UNIDADE 10

Objetivo: Conceituar o significado de chaves e analisar os vários tipos de chaves utilizados em um Banco de Dados.

Definição de Chave Primária Simples, Composta, Única e Estrangeira

É recomendado que uma tabela tenha um ou mais atributos que seja possível identificar um específico registro. Esse atributo é chamado de chave primária. Uma chave primária deverá ser sempre única, nunca poderá ser repetida. Por exemplo, normalmente todo cliente é identificado por um código. O código do cliente é a chave primária, e não poderá existir outro cliente com esse mesmo código. Se os códigos de dois clientes fossem iguais, como iríamos identificar cada um deles?!?

Ao escolher uma chave primária, você deve estar atento aos seguintes detalhes:

A chave primária NUNCA deve ser repetida, portanto não escolha campos que não satisfaçam esta condição. Por exemplo, datas de aniversário normalmente não são boas chaves primárias.

O tamanho da chave primária afeta a velocidade das operações do seu Banco de Dados. Para um bom desempenho, use o menor tamanho possível que acomode esses valores. Prefira campos numéricos.

O campo chave primária sempre vai estar automaticamente indexado, ou seja, ordenado.

Vejamos agora, com maiores detalhes, à análise dos três tipos de chaves convencionais:

Chave Primária Simples e Composta

Quando apenas um atributo compõe a chave primária, ela é chamada de chave primária simples. Quando mais de um atributo compõe a chave primária, ela é chamada de chave

primária composta. Por exemplo, o número da conta corrente normalmente é formado por dois atributos: o número da agência e o número da conta. Obviamente, o número da agência pode ser repetido individualmente, afinal uma agência bancária não tem apenas um número de conta.

Chave Única

Uma chave única é um meio que utilizamos quando um determinado campo não deve ser repetido e não deve ser chave primária. Com esse método, damos mais consistência ao Banco de Dados. Por exemplo, em um cadastro de clientes, normalmente a cada cliente é atribuído um número que é a chave primária. Para se ter maior segurança, adiciona-se o RG como chave única, para evitar que o cliente seja cadastrado duas vezes.

Chave Estrangeira

A chave estrangeira é utilizada quando queremos que o valor de um atributo seja validado a partir do valor de atributo de outra tabela. Nesse caso há certa relação de dependência entre as tabelas. Por exemplo, antes de efetuar o cadastro de um pedido de venda, devemos nos certificar de que o cliente em questão consta no cadastro de clientes. O campo código do cliente na tabela de clientes é uma chave primária. Esse mesmo valor na tabela de pedidos é uma chave estrangeira.



Estudo Complementar

http://www.imasters.com.br/artigo/5403/bancodedados/modelagem_de_dados_-_parte_05_transformacao_entre_modelos//imprimir/





Atividades

Antes de dar continuidade aos seus estudos é fundamental que você entre no site da ESAB e, em sua Sala de Aula, faça a Atividade 1, no link “Atividades”.



UNIDADE 11

Objetivo: Conceituar o significado de relacionamento e analisar os vários tipos utilizados em Banco de Dados.

Relacionamentos entre Bancos de Dados

Uma associação estabelecida entre campos comuns em duas tabelas leva o nome de relacionamento. Dessa forma permitimos o estabelecimento de correspondência entre registros de diferentes tabelas.

Um relacionamento funciona pela coincidência de dados em campos-chaves (geralmente um campo com o mesmo nome em ambas as tabelas). Na maioria dos casos, esses campos coincidentes são a chave primária de uma tabela, e uma chave estrangeira em outra tabela. Um Banco de Dados Relacional requer dados duplicados entre tabelas (os dados que efetivarão as ligações), mas não permite dados duplicados dentro das próprias tabelas.

Por exemplo, podemos associar um cliente específico a uma tabela de pedidos através do campo código do cliente. Com isso, na tabela de pedidos não é necessário ter os dados cadastrais do cliente (nome, endereço, cidade, etc.), pois devido à associação, obtêm-se esses dados a qualquer momento.

Com o relacionamento entre duas entidades, o número de ocorrências de uma entidade com a outra, determina o Grau de Relacionamento, ou a Cardinalidade. No mundo real apresentam-se três possibilidades de relacionarmos os dados, ou seja, três Graus de Relacionamento, que são:

- Relacionamento um-para-um
- Relacionamento um-para-muitos
- Relacionamento muitos-para-muitos

Relacionamento um-para-um

Em um relacionamento um-para-um, cada registro na tabela X pode ter somente um registro coincidente na tabela Y. E por sua vez, cada registro na tabela Y pode ter somente um registro coincidente na tabela X. Para que esse relacionamento aconteça, os campos relacionados nas duas tabelas devem ser chaves primárias. Esse tipo de relacionamento não é muito comum, pois a maioria dos dados relacionados poderia simplesmente estar em uma única tabela.

A utilização de um relacionamento um-para-um é recomendada quando você deseja dividir uma tabela com muitos campos, isolar parte de uma tabela por segurança ou armazenar dados que se apliquem somente a um subconjunto da tabela principal. Outra possibilidade seria aumentar a performance do Banco de Dados quando uma tabela, muito acessada, apresenta-se com muitos campos dividindo-a em duas tabelas.

Por exemplo, o relacionamento na tabela de Funcionário com a de Armário (veja a Figura 11.1). Observe que para que haja este relacionamento, os campos relacionados nas duas tabelas devem ser chaves primárias (codigo e funcionario). Nesta aplicação específica, um funcionário somente poderá ter um armário e vice-versa.

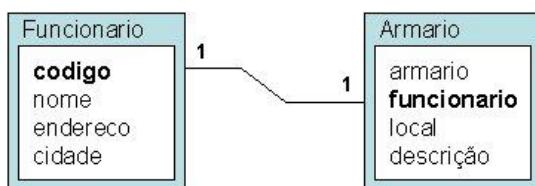


Figura 11.1 - Relacionamento um-para-um

Relacionamento um-para-muitos

Um relacionamento um-para-muitos estabelece que um registro em uma tabela X pode ter vários registros associados em uma tabela Y. Para isso o campo relacionado na tabela X deve ser chave primária, e o campo na tabela Y não deve ser chave primária. Ao efetuarmos

o relacionamento, o campo relacionado na tabela Y será chamado de chave estrangeira. Este é o relacionamento mais comum e o mais utilizado em Banco de Dados.

Por exemplo, um cliente pode efetuar vários pedidos, mas um pedido só pode ser feito por um cliente. Note que para isso é preciso que o campo relacionado na tabela Cliente seja chave primária (código), e o campo relacionado em Pedido não seja chave primária (cliente).

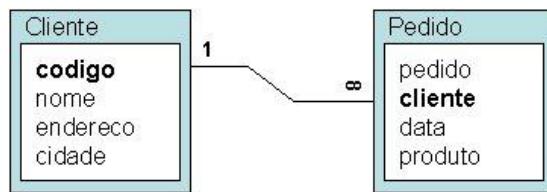


Figura 11.2 – Relacionamento um-para-muitos

Em um relacionamento um-para-muitos, a tabela com a chave primária é dita “tabela pai”. E, consequentemente, a outra será chamada de “tabela filho”.

Relacionamento muitos-para-muitos

Note que a tabela de Pedido no exemplo anterior possui uma falha: cada pedido somente poderá ter um único produto. Na vida real, os clientes ao solicitarem vários produtos teriam que fazer um pedido para cada produto, inviável !!

Uma solução seria colocar mais campos na tabela de Pedido para que pudessem colocar mais produtos (produto1, produto2, etc). Mas, mesmo assim, estaríamos limitados ao número de campos criados, e o processamento de informações nessa tabela seria lento. Uma melhor solução seria estabelecermos um relacionamento muitos-para-muitos, em que vários produtos podem estar em vários pedidos.

Em um relacionamento muitos-para-muitos, um registro na tabela X pode ter vários registros coincidentes na tabela Y, e um registro da tabela Y pode ter vários registros coincidentes na tabela X. No entanto, não é possível efetuar este relacionamento de forma direta. Esse tipo de relacionamento só é possível definindo uma terceira tabela (denominada tabela de associação ou tabela de detalhes). Ela deve possuir chave primária composta de dois campos, e as chaves estrangeiras provenientes tanto da tabela X como da Y. Observe que

na verdade, um relacionamento muitos-para-muitos são dois relacionamentos um-para-muitos, com uma tabela intermediária.

Por exemplo, para efetuar uma venda de um produto apenas, são necessários no mínimo um registro na tabela de Pedido e um registro na tabela de ItensPedido. Ao efetuar uma venda com dois produtos, são necessários um registro em Pedido e dois registros em ItensPedido. Dessa forma, cada produto vendido será necessário um registro na tabela de ItensPedido, e a cada pedido, um registro na tabela de Pedido.

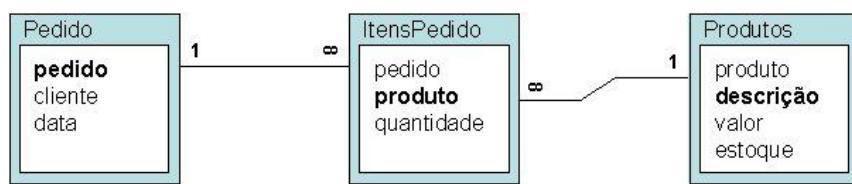


Figura 11.3 – Relacionamento muitos-para-muitos

Note que a tabela de associação possui uma chave primária composta pelos campos pedido e produto. Esses campos são chaves estrangeiras nos relacionamentos um-para-muitos.



Estudo Complementar

<http://access-exemplos.blogspot.com/2007/08/relacionamentos-numa-base-de-dados.html>



Atividades

Tente através do MS-Access criar as tabelas dos exemplos citados, e fazer os relacionamentos aprendidos.



UNIDADE 12

Objetivo: Conceituar o significado de Relacionamentos Especiais com Múltiplas Entidades em um Banco de Dados.

Relacionamentos Especiais com Múltiplas Entidades

Até o momento estudamos e analisamos situações em que as entidades se relacionam aos pares. Este é o princípio de descoberta dos relacionamentos na construção de um modelo de dados: analisar as entidades aos pares.

Entretanto um relacionamento pode existir envolvendo mais de duas entidades, que podem ser três, quatro, ou uma quantidade indeterminada de entidades que os relacionamentos estiverem envolvidos.

Os relacionamentos entre múltiplas entidades expressam um fato em que todas as entidades ocorrem simultaneamente, ou seja, todas as ocorrências do relacionamento possuem, sempre, ligações com todas as entidades envolvidas no relacionamento. Não pode existir um relacionamento triplo, que em um determinado momento, se transforme em duplo.

O Relacionamento Múltiplo com mais de quatro entidades relacionadas é extremamente difícil de se encontrar na realidade do dia a dia. Quando você encontrar com algum fato que dê origem a um relacionamento múltiplo, analise com cuidado, pois o mesmo pode ser desmembrado em mais de um relacionamento. A implementação de relacionamento múltiplo em Bancos de Dados torna o trabalho de manipulação bastante complexo.



Atividades

Baixe a apresentação do Prof. Yasuo Kono sobre Relacionamentos Especiais (link referenciado anteriormente no Estudo complementar) e realize um breve resumo.



UNIDADE 13

Objetivo: Definir o que vem a ser integridade dentro do contexto de Banco de Dados.

Integridade dos Dados

Os dados de um SGBD devem ser íntegros e consistentes. Para isso são necessários que se estabeleçam alguns critérios que garantam a consistência em um Banco de Dados e sua integridade. Veja as seguintes situações típicas de falha na integridade dos dados:

Pense o leitor como seria complicado conseguir o endereço de um cliente, se na hora do seu cadastro esqueceu-se esse detalhe. Ainda por cima se existem outros clientes cadastrados com esse mesmo problema. Uma mala direta seria inviável.

Um funcionário, ao cadastrar o cliente, cometeu um erro de digitação e digitou um estado inexistente ou um estado em que a empresa não atua?

Imagine um pedido de vendas de um valor bem alto e importante para a empresa, e que devido a um lamentável erro perdeu-se o código do cliente.

São problemas que devemos considerar na criação de um Banco de Dados. As regras de integridade podem se apresentar de várias formas, vamos aqui considerar as três formas mais comuns:

- Integridade de Domínio
- Integridade de Entidade
- Integridade Referencial

Integridade de Domínio

Este tipo de integridade zela pelos valores ideais e necessários para um atributo. Para isso, definimos algumas regras de validação por meio de expressões compostas de valores constantes. Por exemplo:

- Não permitir um estoque negativo.
- Impedir uma data de nascimento incompatível com a idade mínima do público-alvo.
- Não permitir que o valor de um produto seja negativo.

Integridade de Entidade

A integridade de entidade tem o objetivo de validar os valores permitidos a partir de valores já inseridos anteriormente. Após uma “autoconsulta” a entidade vai permitir ou não a gravação de um novo registro. Por exemplo:

- Não permitir duas pessoas com o mesmo RG.
- Em uma locadora, impedir que seja locada uma fita que já está locada.

Integridade Referencial

A partir do momento que estabelecemos alguns relacionamentos, as regras de integridade referencial já estão automaticamente criadas. Elas zelam pela consistência dos registros de uma entidade a partir de valores provenientes de outras entidades. Ou seja, determinado registro vai depender diretamente de um registro em outra tabela.

Um registro em uma determinada tabela pai pode ter um ou mais registros coincidentes na tabela filho.

Um registro em uma tabela filho sempre tem um único registro coincidente em uma tabela pai.

Para a inclusão de um registro em uma determinada tabela filho, é necessário que exista um registro pai coincidente.

Um registro pai só poderá ser excluído se não possuir nenhum registro filho.



Atividades

Realize uma análise crítica, e comparativa com o que a gente já viu até agora, com o excelente artigo de João Roberto da Cunha, da Serpro, no link abaixo:

<http://www1.serpro.gov.br/publicacoes/tematec/pubtem03.htm>



UNIDADE 14

Objetivo: Fundamentar o conceito de normalização e as suas técnicas.

Normalização – A Primeira, Segunda e a Terceira Forma Normal

O conceito de normalização foi introduzido por E.F. Codd em 1970 (primeira forma normal). Esta técnica é um processo matemático formal, que tem seus fundamentos na teoria dos conjuntos. Através deste processo pode-se, gradativamente, substituir um conjunto de entidades e relacionamentos, “purificando” os mesmos evitando certos problemas, tais como: grupos repetitivos de dados, redundância de dados desnecessários e perdas acidentais de informações.

Um cadastro de funcionário certamente conteria todos os dados necessários sobre o funcionário, porém terá um grande volume de redundâncias com relação a outros funcionários. Normalização é uma técnica de análise e organização de dados, que consiste na decomposição de um depósito de dados (estrutura não-normalizada), em formas mais simples, denominadas formas normais.

Os objetivos da normalização são:

- Independência dos dados.
- Minimizar redundâncias, que por sua vez minimiza os riscos de inconsistências.
- Facilitar a manipulação do Banco de Dados.
- Facilitar a manutenção dos sistemas de informação, sem grandes impactos.

Existem duas formas de utilização da normalização:

TOP-DOWN: após a definição do modelo de dados (diagrama entidade-relacionamento, por exemplo), aplica-se a normalização para se obter uma síntese dos dados, bem como uma

decomposição das entidades e relacionamentos em elementos mais estáveis, tendo em vista sua implementação física em um BD;

BOTTOM-UP: aplica-se a normalização como ferramenta de projeto do modelo de dados, usando os relatórios, formulários e documentos utilizados pela realidade em estudo, constituindo-se em uma ferramenta de levantamento.

Para ilustrar os problemas da falta de normalização e iniciar um processo BOTTOM-UP, considere a entidade PEDIDO baseada numa nota fiscal de uma loja, como mostrado abaixo:

Num_Ped	Nome_Cli	Endereco	Cidade	UF	Cod_Prod	Qtde	Descr	Valor_Unit	Total_Prod	Total_Ped	Cod_Vend	Nome_vend
---------	----------	----------	--------	----	----------	------	-------	------------	------------	-----------	----------	-----------

Pedido (Num_Ped, Nome_Cli, Endereco, Cidade, UF, Cod_Prod, Qtde, Descr, Valor_Unit, Total_Prod, Total_Ped, Cod_Vend, Nome_Vend)

As anomalias de atualização desse projeto são:

- Anomalia de Inclusão: ao ser incluído um novo cliente, o mesmo tem que estar relacionado a uma venda;
- Anomalia de Exclusão: ao ser excluído um cliente, os dados referentes às suas compras serão perdidos;
- Anomalia de Alteração: caso algum fabricante de produto altere o preço de um produto, será preciso percorrer toda a relação para se realizar múltiplas alterações.

Primeira Forma Normal (1FN)

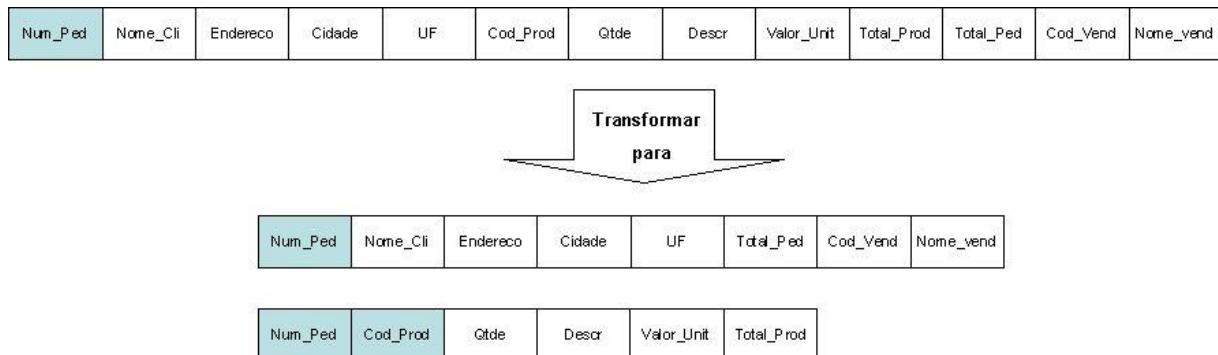
A 1FN trata de informações que se repetem (atributos multivalorados) e define que cada ocorrência da chave primária deve corresponder a uma e somente uma informação de cada atributo, ou seja, a entidade não deve conter grupos repetitivos.

Para um esquema com relações não normalizadas (como a relação PEDIDO acima), a solução é decompor cada relação não normalizada em tantas entidades quanto for o número de grupos repetitivos.

Nas novas relações, a chave primária é a concatenação da chave primária da relação original mais os atributos do grupo repetitivo visualizados como chave primária.

Exemplo:

Aplicando 1FN na relação PEDIDO, observa-se que existe um grupo repetitivo dentro da relação, que trata dos vários produtos que podem constar num pedido. Isto se deve ao fato de que este grupo forma uma coluna não atômica. Este grupo é composto dos atributos Cod_Prod, Qtde, Descr, Valor_Unit, Total_Prod. Portanto, cria-se uma nova relação, a ser chamada de ITEM_PEDIDO, ficando o esquema da seguinte forma:



PEDIDO (Num_Ped, Nome_Cli, Endereco, Cidade, UF, Total_Ped, Cod_Vend, Nome_vend)

ITEM_PEDIDO (Num_Ped, Cod_Prod, Qtde, Descr, Valor_Unit, Total_Prod)

Dependência

Para mostrar a 2FN e 3FN, é necessário à introdução dos Conceitos de Dependência entre atributos.

Dependência Funcional Total - Um atributo ou conjunto de atributos depende de forma total da chave primária concatenada se a cada valor da chave está associado um valor para este atributo.

Dependência Funcional Parcial - O atributo só depende de parte da chave primária. Exemplo:

ITEM_PEDIDO (Num_Ped, Cod_Prod, Qtde, Descr, Valor_Unit, Total_Prod)

O atributo Qtde depende de forma total da chave primária (Num_Ped + Cod_Prod)

O atributo Descr (descrição do produto) depende de forma parcial da chave primária, pois só depende do Cod_Prod e não do Num_Ped + Cod_Prod.

Dependência Funcional Transitiva - Quando um atributo ou conjunto de atributos 'A' depende de outro atributo 'B' que não pertence à chave primária, mas é dependente funcional desta, dizemos que 'A' é dependente transitivo de 'B'. Exemplo:

PEDIDO (Num_Ped, Nome_Cli, Endereco, Cidade, UF, Total_Ped, Cod_Vend, Nome_vend)

Os atributos Endereco, Cidade, UF são dependentes transitivos do atributo Nome_Cli.

Nome_vend é dependente transitivo de Cod_Vend

Segunda Forma Normal (2FN)

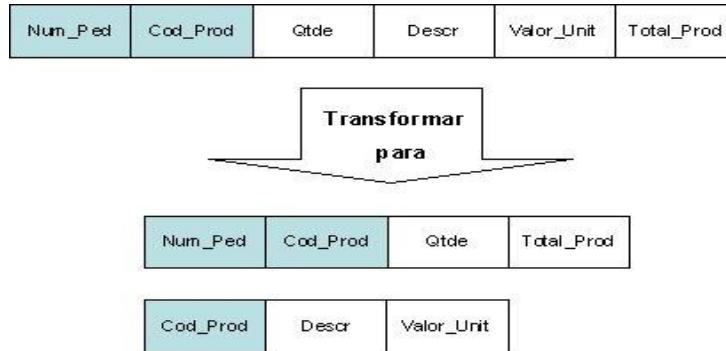
Uma relação se encontra em 2FN quando estiver em 1FN e todos os atributos que não participam da chave primária são dependentes desta. Assim devemos verificar se todos os atributos são dependentes da chave primária, e retirar da relação todos os atributos de um grupo não dependente que dará origem a uma nova relação, que conterá esse atributo como não chave. Desta maneira, em 2FN evita inconsistências devido a duplicidades.

Exemplo: (a única relação no nosso esquema com chave concatenada)

ITEM_PEDIDO (Num_Ped, Cod_Prod, Unid, Qtde, Descr, Valor_Unit, Total_Prod)

Os atributos Descr, Valor_Unit dependem de Cod_Prod o qual faz parte da chave primária.

Cria-se a relação PRODUTO (Cod_Prod, Descr, Valor_Unit), com a chave primária Cod_Prod e em ITEM_PEDIDO, a chave primária permanece Num_Ped + Cod_Prod.



ITEM_PEDIDO (Num_Ped, Cod_Prod, Qtde, Total_Prod)

PRODUTO (Cod_Prod, Descr, Valor_Unit)

Terceira Forma Normal (3FN)

Uma relação está na 3FN se está em 2FN e se nenhum de seus atributos possui dependência transitiva em relação a outro atributo que não esteja na chave primária. Ao retirarmos a dependência transitiva, criam-se novas relações com os atributos que dependem transitivamente do outro, e a sua chave primária é o atributo que causou essa dependência.

Uma relação na 3FN, além de não ter dependência transitiva, não pode ter atributos que sejam resultado de algum cálculo sobre outro atributo, o que, de certa forma, pode ser encarado como dependência funcional. Exemplo:

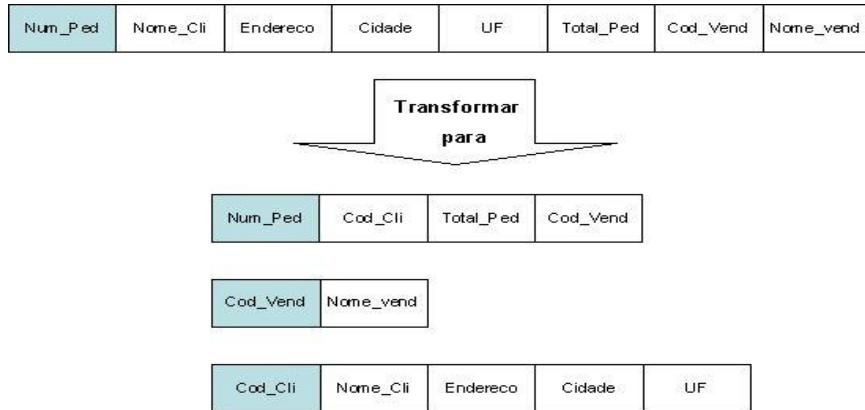
PEDIDO (Num_Ped, Nome_Cli, Endereco, Cidade, UF, Total_Ped, Cod_Vend, Nome_vend)

O atributo Nome_vend depende transitivamente de Cod_Vend, o qual não pertence à chave primária, mas depende desta;

Cria-se a relação VENDEDOR (Cod_Vend, Nome_vend);

Os atributos Endereco, Cidade, UF dependem transitivamente de Nome_Cli que não está na chave primária (mas depende desta);

Cria-se a relação CLIENTE (Cod_Cli, Nome_Cli, Endereco, Cidade, UF).



No final ficaríamos com um esquema, mais ou menos, assim:

PEDIDO (Num_Ped, Cod_Cli, Total_Ped, Cod_Vend)

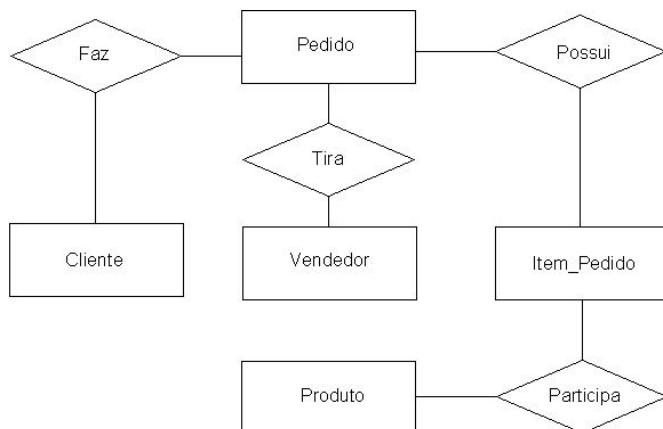
VENDEDOR (Cod_Vend, Nome_vend)

CLIENTE (Cod_Cli, Nome_Cli, Endereco, Cidade, UF)

ITEM_PEDIDO (Num_Ped, Cod_Prod, Qtde, Total_Prod)

PRODUTO (Cod_Prod, Descr, Valor_Unit)

Observação: note que foi criado um código para o cliente, sendo este código a chave primária da relação, pois o nome do cliente não seria uma chave apropriada. Um modelo ER do problema acima seria:



O processo de normalização deve ser aplicado em uma relação por vez, pois durante o processo de normalização vamos obtendo quebras, e, por conseguinte, novas relações. No

momento em que o sistema estiver satisfatório, do ponto de vista do analista, este processo iterativo é interrompido. Embora existam 4FN e 5FN, não vamos explorá-las devido a sua complexidade.



Estudo Complementar

http://pt.wikipedia.org/wiki/Normaliza%C3%A7%C3%A3o_de_dados



Atividades

A normalização é uma parte complexa, mas importante dentro dos conceitos de Banco de Dados. Quem domina adequadamente esse conceito, consegue criar Bancos de Dados inteligentes e racionais. Portanto, faça um comparativo entre o que foi explicado nesta unidade com o material referenciado do wikipédia no Estudo Complementar.



UNIDADE 15

Objetivo: Entender os princípios básicos do armazenamento e manipulação de dados.

Fundamentos de Armazenamento e Manipulação de Dados

Linguagem de Definição de Dados

Um esquema de Banco de Dados é especificado por um conjunto de definições expressas por uma linguagem especial chamada linguagem de definição de dados (Data Definition Language, DDL). O resultado da compilação de comandos de uma DDL é um conjunto tabelas que são armazenadas em um arquivo especial chamado dicionário (ou diretório) de dados.

Um diretório de dados é um arquivo que contém metadados, ou seja, "dados sobre dados". Este arquivo é consultado antes que os dados sejam lidos ou modificados no sistema de Banco de Dados.

A estrutura de armazenagem e os métodos de acesso usados em um sistema de Banco de Dados são especificados por um conjunto de definições em um tipo especial de DDL chamado linguagem de armazenagem e definição de dados. O resultado da compilação destas definições é um conjunto de instruções para especificar a implementação de detalhes do esquema de Banco de Dados que estão normalmente escondidos dos usuários.

Linguagem de Manipulação de Dados

Os níveis de abstração discutidos anteriormente (níveis físico, conceitual e de visão) não se aplicam somente à definição ou estrutura de dados, mas também à sua manipulação. A manipulação de dados significa:

- A busca da informação armazenada no BD;

- A inserção de novas informações nos BD;
- A eliminação de informações no BD;
- A modificação de dados armazenados no BD.

No nível físico precisamos definir algoritmos que permitam um acesso eficiente aos dados. Nos níveis mais altos de abstração é dada ênfase à facilidade de uso. O objetivo é fornecer uma interação humana eficiente com o sistema.

A linguagem de manipulação de dados (Data Manipulation Language, DML) é a linguagem que permite aos usuários fazer o acesso a os dados ou manipulá-los, conforme modelo apropriado de dados. Existem basicamente dois tipos:

- DMLs procedurais requerem do usuário a especificação de qual dado é necessário e de como obtê-lo;
- DMLs não-procedurais requerem do usuário a especificação de qual dado é necessário sem especificar como obtê-lo.

DMLs não-procedurais são usualmente mais fáceis de aprender e usar do que DMLs procedurais. Entretanto, se um usuário não necessita especificar como obter os dados, estas linguagens podem gerar código não tão eficiente como o produzido por linguagens procedurais. Esta dificuldade pode ser remediada por meio de várias técnicas de otimização.

Uma consulta (query) é um comando requisitando a busca de uma informação. A porção de uma DML que envolve busca de informações é chamada linguagem de consulta. Embora tecnicamente incorreto, é comum utilizar os termos linguagem de consulta e linguagem de manipulação de dados como sinônimos.

Estrutura geral do sistema

Um sistema de Banco de Dados é dividido em módulos que tratam de cada uma das responsabilidades do sistema geral. Na maioria dos casos, o sistema operacional do

computador fornece apenas os serviços mais básicos e o sistema de Banco de Dados precisa ser construído sobre essa base. Portanto, o projeto do sistema de Banco de Dados precisa incluir considerações sobre a interface entre o sistema de Banco de Dados e o sistema operacional.

Os componentes funcionais de um sistema de Banco de Dados incluem:

- Gerenciador de arquivos, que gerencia a alocação do espaço na armazenagem do disco e as estruturas de dados usadas para representar a informação armazenada no disco.
- Gerenciador do Banco de Dados, que fornece a interface entre os dados de baixo nível armazenados no disco e os programas aplicativos e de consulta submetidos ao sistema.
- Processador de consultas, que traduz os comandos numa linguagem de consulta para instruções de baixo nível que o gerenciador do Banco de Dados pode interpretar. Além disso, o processador de consultas tenta transformar uma requisição do usuário em uma forma compatível e mais eficiente com respeito ao Banco de Dados, encontrando uma boa estratégia para a executar a consulta.
- Pré-compilador da DML, que converte comandos da DML embutidos em um aplicativo para chamadas de procedimento normal na linguagem hospedeira. O pré-compilador precisa interagir com o processador de consultas pra gerar o código apropriado.
- Compilador da DDL, que converte comandos da DDL em um conjunto de tabelas contendo metadados ou "dados sobre dados".

Adicionalmente, diversas estruturas de dados são requeridas como parte da implementação do sistema físico, incluindo:

Arquivos de dados, que armazenam o Banco de Dados propriamente dito.

Dicionário de dados, que armazena metadados sobre a estrutura do Banco de Dados. O dicionário de dados é usado com frequência. Assim, deve-se dar grande ênfase no desenvolvimento de um bom projeto e implementação eficiente do dicionário.

Índices, que fornecem acesso rápido aos itens de dados guardando determinados valores.

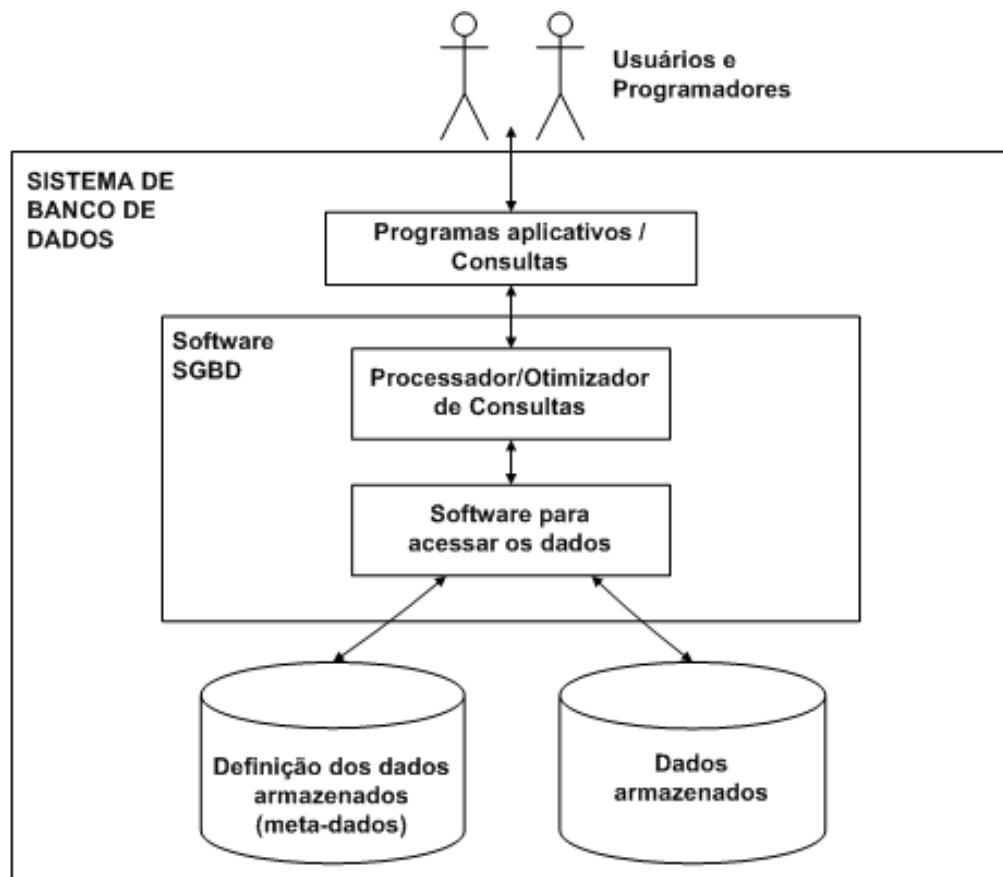


Figura 15.1 - Diagrama simplificado da arquitetura do sistema de Banco de Dados

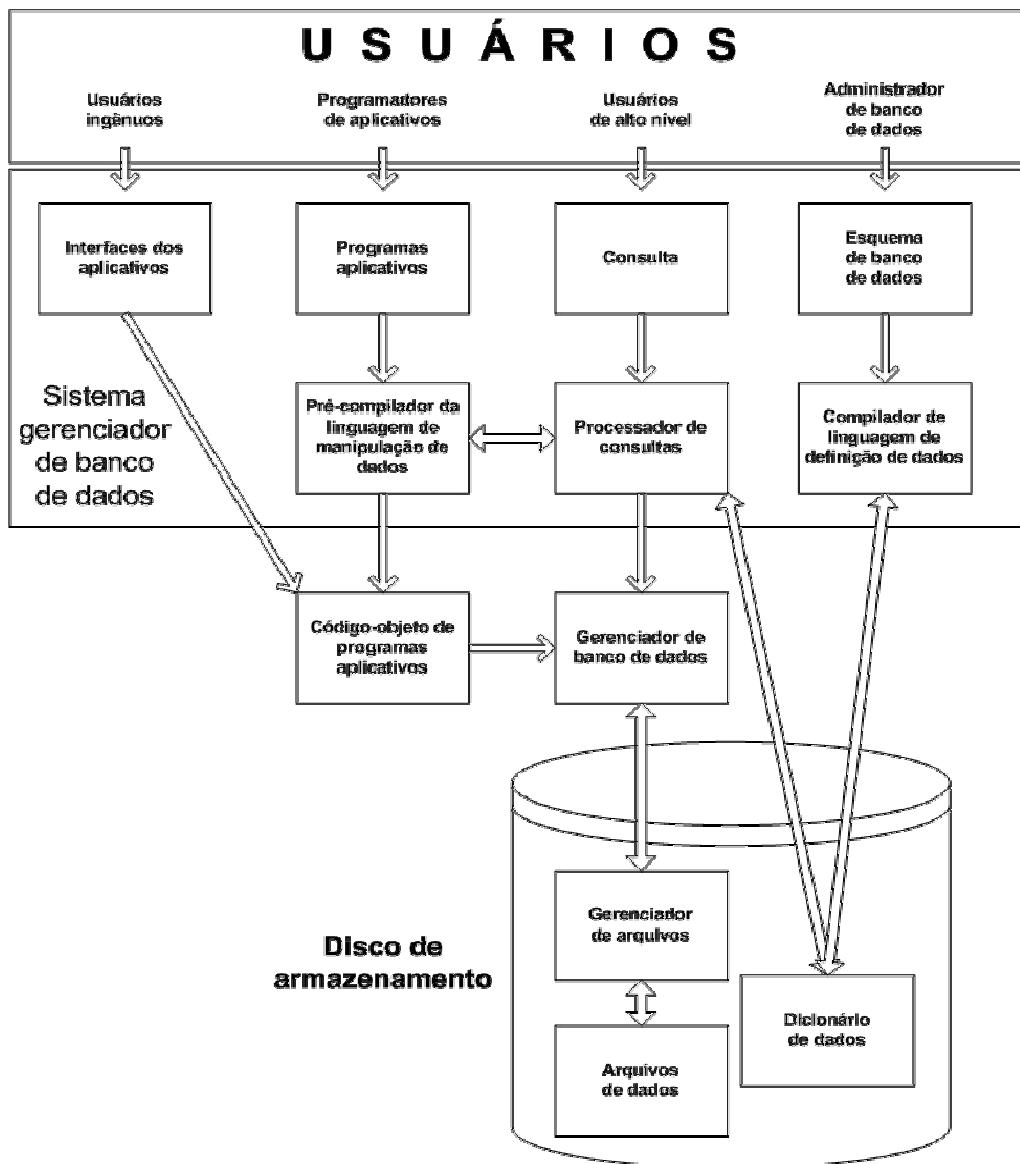


Figura 15.2 - Diagrama expandido da arquitetura do sistema de Banco de Dados



Estudo Complementar

<http://pt.wikipedia.org/wiki/Servidor>



Atividades

Com base nas figuras 15.1 e 15.2, tente descrever através de palavras o que os diagramas tentam repassar.



UNIDADE 16

Objetivo: Definir e exemplificar a linguagem mais significativa em Banco de Dados

A Linguagem SQL

O nome SQL significa “Structured Query Language” (Linguagem Estruturada de Pesquisa). Essa linguagem, de grande utilização, teve seus fundamentos no modelo relacional de Edgar Frank Codd (1970). Sua primeira versão recebeu o nome de SEQUEL (“Structured English Query Language”), em 1974, nos laboratórios da IBM. Entre 1976 e 1977, o SEQUEL foi revisado e ampliado, mas teve que ser alterado o seu nome para SQL devido já ter sido registrado o cognome anterior.

Graças ao sucesso dessa nova forma de consulta e manipulação de dados, dentro de um ambiente de Banco de Dados, a utilização da SQL foi se tornando cada vez maior. Com isso uma grande quantidade de SGBD's tem como linguagem básica o SQL, como por exemplo, DB2 da IBM, ORACLE da Oracle Corporation, RDB da Digital, SYBASE da Sybase INC e SQL Server da Microsoft, entre outros.

O SQL se tornou um padrão de fato, no mundo dos ambientes de Banco de Dados relacionais, porém somente em 1982. Infelizmente, existem hoje vários “dialetos” SQL, cada SGBD inclui implementações específicas. Neste curso seguiremos o padrão ANSI do SQL, mostrando os conceitos básicos e a importância desta linguagem.

Atualmente, a linguagem SQL, assume um papel muito importante nos SGBD, podendo ter muitos enfoques, como os listados abaixo:

Linguagem interativa de consulta: Por meio de comandos SQL, os usuários podem montar consultas poderosas sem a necessidade de criação de um programa, podendo utilizar Forms ou ferramentas de montagem de relatório;

Linguagem de programação para acesso a Banco de Dados: Comandos SQL embutidos em programas de aplicação que acessam os dados armazenados;

Linguagem de administração de Banco de Dados: O responsável pela administração do Banco de Dados (DBA) pode utilizar comandos SQL para realizar suas tarefas;

Linguagem cliente/servidor: Os programas (cliente) dos computadores pessoais usam comandos SQL para se comunicarem por meio de uma rede local, compartilhando os dados armazenados em um único local (servidor). A arquitetura cliente/servidor minimiza o tráfego de dados pela rede;

Linguagem para Banco de Dados Distribuído: O SQL auxilia na distribuição dos dados por meio de vários nós conectados ao sistema de computação. Auxilia também na comunicação de dados com outros sistemas;

Caminho de acesso a outros Bancos de Dados em diferentes máquinas: O SQL auxilia na conversão entre diferentes produtos de Banco de Dados colocados em diferentes máquinas (de micro até mainframe).

O SQL apresenta uma série de comandos que permitem a definição dos dados, chamada de DDL (Data Definition Language). Como exemplo de comandos da classe DDL temos os comandos Create, Alter e Drop.

Os comandos da série DML (Data Manipulation Language), são destinados a consultas, inserções, exclusões e alterações em um ou mais registros. Como exemplo de comandos da classe DML temos os comandos Select, Insert, Update e Delete.

Existe ainda uma subclasse de comandos DML, a DCL (Data Control Language). Ela dispõe de comandos de controle como Grant e Revoke.

Outra característica muito importante disponível em SQL é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados. Os comandos Commit e Rollback na linguagem SQL

têm a capacidade de cancelar uma série de atualizações ou de as gravarmos, depois de iniciarmos uma sequência de atualizações.

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices. Vamos destacar alguns comandos:

COMMIT; Confirma a transação

ROLLBACK; Desfaz a transação

SHOW <tabela>; Mostra os nomes das tabelas existentes em determinado Banco de Dados

SHOW FIELDS FOR <tabela>; Mostra os campos de determinada tabela

SHOW INDEXES FOR <tabela>; Lista de índices da tabela

SHOW RELATIONSHIPS FOR <tabela>; Lista os relacionamentos da tabela

LIST <tabela>; Lista conteúdo da tabela



Estudo Complementar

<http://pt.wikipedia.org/wiki/SQL>



Atividades

Devido à importância do SQL em Banco de Dados, recomendamos fortemente a leitura complementar do link acima dentro do Wikipédia, e a visita aos links recomendados.



UNIDADE 17

Objetivo: Detalhar a sintaxe do comando SQL mais básico de um SGBD.

Comando para Criação de um Banco de Dados

Comando Create

Este comando permite a criação de tabelas no Banco de Dados.

Sintaxe:

```
CREATE DATABASE < nome_db >;
```

onde:

nome_db - indica o nome do Banco de Dados a ser criado.

Sintaxe:

```
CREATE TABLE < nome_tabela >
```

```
( nome_atributo1 < tipo > [ NOT NULL ],
```

```
  nome_atributo2 < tipo > [ NOT NULL ],
```

.....

```
  nome_atributoN < tipo > [ NOT NULL ] );
```

onde:

nome_table indica o nome da tabela a ser criada.

nome_atributo indica o nome do campo a ser criado na tabela.

tipo indica a definição do tipo de atributo (integer(n), char(n),

real(n,m), date...).

n número de dígitos ou de caracteres.

m número de casas decimais.

Agora vamos criar nossa primeira tabela:

CREATE DATABASE TRABALHO;

O comando acima criou um Banco de Dados, porém este na verdade não passa de uma abertura no diretório, pois não conta com nenhuma tabela. Agora criaremos as tabelas que estarão contidas no Banco de Dados TRABALHO.

A primeira Tabela será a de Departamentos (Dept). Essa tabela conterá, além de campos, também sua chave primária, suas chaves estrangeiras e também seus índices. A segunda tabela será a de Empregados (EMP).

Não devemos nos esquecer de primeiramente abrirmos o Banco de Dados. Diferentemente do que ocorre em alguns aplicativos, em SQL quando criamos um Banco de Dados, não significa que o banco recém criado já está preparado para utilização. A instrução a seguir, providencia a abertura do Banco de Dados:

OPEN DATABASE TRABALHO;

Agora estamos prontos para criarmos as tabelas necessárias. A seguir, entramos com o código necessário para a criação da tabela Departamento (Dept) e seu índice:

create table Dept

(DepNume integer(4) not null,

DepNome char(20) not null,

DepLoca char(20) not null,

DepOrca integer(12,2),

primary key (DepNume)

);

create unique index DepNum on Dept (DepNume asc);

Note que a chave primária já está definida juntamente com o registro da tabela. A criação do índice, que por razões óbvias deve ser criado após a tabela, naturalmente é um comando totalmente independente do primeiro create, que serviu para criar a tabela e suas características básicas.

Vamos analisar o código necessário para a criação da tabela de empregados, apresentado a seguir:

create table Emp

(EmpNume integer(5) not null,

EmpNome char(30) not null,

EmpGere integer(5) ,

EmpServ char(20) ,

DepNume integer(4) not null,

EmpAdmi date not null,

EmpSala integer(10,2),

EmpComi integer(10,2),

primary key (EmpNume),

foreign key has (DepNume)

references Dept

on delete restrict

on update cascade

);

```
create unique index EmpNum on Emp (EmpNume asc);
```

```
create index EmpDep on Emp (DepNume asc);
```

A Tabela de Empregados não poderia ter sido criada *antes* da Tabela de Departamento, pois contém uma referência direta àquela tabela. Quando declaramos que *DepNume* é chave estrangeira, promovemos de fato a ligação do cadastro de empregados com o cadastro de departamentos. Ao restringirmos as exclusões, permitimos a existência de funcionários não alocados a nenhum departamento. Apesar de esta prática ser contrária à tese de que devemos possuir apenas registros perfeitamente relacionáveis em nossas tabelas, podemos deixar esta pequena abertura, pois um usuário que excluisse inadvertidamente determinado departamento, acabaria por excluir também uma grande quantidade de funcionários, que estivessem ligados a este departamento.

Já a atualização em cascata dos códigos de departamento é uma boa providência. Uma vez alterado algum código de departamento, a atualização será imediata em todos os funcionários pertencentes ao departamento cujo código foi modificado.

Observações importantes:

Observar que os índices são partes intrínsecas das tabelas.

A integridade relacional é garantida pelo Banco de Dados e não pelo aplicativo.

Exclusões ou Alterações em Chaves Primárias podem acarretar exclusões, anulações ou até mesmo perda de integridade nas tabelas onde esta chave primária existir como chave estrangeira.

Portanto, é imprescindível muito cuidado quando da elaboração de um Banco de Dados. Uma tentação muito comum dos novatos é começar criando as tabelas do Banco de Dados sem utilizar a “Normalização”. Este talvez seja o melhor caminho para perder tempo em vão, pois quando você terminar de projetar suas telas de entrada de dados, notará “que nada funciona!”.

Comando *DROP*

Este comando elimina a definição da tabela, seus dados e referências.

Sin taxe:

```
DROP TABLE < nome_tabela > ;
```

Exemplo: `DROP TABLE EMP;`



Estudo Complementar

<http://pgdocptbr.sourceforge.net/pg80/sql-createtableas.html>





Atividades

Nos links acima estamos verificando o mesmo comando em dois conhecidos Bancos de Dados. Existem diferenças significativas?



UNIDADE 18

Objetivo.: Verificar e exemplificar o comando SQL mais importante em Banco de Dados.

Comando SELECT para Consulta de Tabelas

Devemos ressaltar que a linguagem SQL é utilizada tanto por profissionais responsáveis pelos dados, onde é ressaltada principalmente pela figura do Administrador do Banco de Dados, como também pelos desenvolvedores de aplicações. Enquanto aqueles estão preocupados com o desempenho, integridade do Banco de Dados e utilizam toda gama de recursos disponíveis no SQL, estes últimos estão preocupados em "transformar dados em informações". Portanto, para os desenvolvedores costuma-se dizer que conhecer o SELECT já basta, devido à importância e o potencial desse comando. Devido à importância deste comando, vamos ensiná-lo didaticamente através de vários exercícios que começam de forma bem simples, até os mais complexos.

Exemplo 1 - Selecione todos os campos (ou colunas) da tabela de Departamentos.

RESPOSTA:

```
SELECT * FROM DEPT;
```

O exemplo acima utiliza o coringa "*" para selecionar as colunas na ordem em que foram criadas. A instrução *Select*, como podemos observar, seleciona um grupo de registros de uma ou mais tabela(s). No caso a instrução *From* nos indica a necessidade de pesquisarmos tais dados apenas na tabela Dept.

Where como base das Restrições de registros

A cláusula where corresponde ao operador “restrição” da álgebra relacional. Contém a condição que os registros devem obedecer a fim de serem listadas. Ela pode comparar valores em colunas, literais, expressões aritméticas ou funções.

A seguir apresentamos operadores lógicos e complementares a serem utilizados nas expressões apresentadas em where.

Operadores lógicos

operador	Significado
=	igual a
>	maior que
>=	maior que ou igual a
<	menor que
<=	menor que ou igual a

Exemplos:

```
SELECT EMPNOME, EMPSERV
```

```
FROM EMP
```

```
WHERE DEPNUME > 10;
```

```
SELECT EMPNOME, EMPSERV
```

```
FROM EMP
```

```
WHERE EMPSERV = 'GERENTE';
```

O conjunto de caracteres ou datas deve estar entre apóstrofes (') na cláusula WHERE.

Exemplo 2 - Selecione todos os departamentos cujo orçamento mensal seja maior que 100000. Apresente o Nome de tal departamento e seu orçamento anual, que será obtido multiplicando-se o orçamento mensal por 12.

RESPOSTA: Neste problema precisamos de uma expressão que é a combinação de um ou mais valores, operadores ou funções que resultarão em um valor. Esta expressão poderá conter nomes de colunas, valores numéricos, constantes e operadores aritméticos.

```
SELECT DEPNOME, DEPORCA * 12  
FROM DEPT  
WHERE DEPORCA > 100000;
```

Exemplo 3 - Apresente a instrução anterior, porém ao invés dos nomes "feios" DepNome e DepOrca, os Títulos Departamento e Orçamento.

RESPOSTA: Neste exemplo deveremos denominar colunas por apelidos. Os nomes das colunas mostradas por uma consulta são geralmente os nomes existentes no Dicionário de Dados. Porém, geralmente estão armazenados na forma do mais puro "informatiquês", onde "todo mundo" sabe que CliCodi significa Código do Cliente. É possível (e provável) que o usuário desconheça estes símbolos, portanto devemos os apresentar dando apelidos às colunas "contaminadas" pelo informatiquês, que apesar de fundamental para os analistas, somente são vistos como enigmas para os usuários.

```
SELECT DEPNOME "DEPARTAMENTO", DEPORCA * 12 "ORCAMENTO ANUAL"  
FROM DEPT  
WHERE DEPORCA > 100000;
```

Exemplo 4 - Apresente todos os salários existentes na empresa, porém omita eventuais duplicidades.

RESPOSTA: A cláusula DISTINCT elimina duplicidades, significando que somente relações distintas serão apresentadas como resultado de uma pesquisa.

```
SELECT DISTINCT EMPSERV
```

```
FROM EMP;
```

Exemplo 5 - Apresente todos os dados dos empregados, considerando sua existência física diferente de sua existência lógica (ou seja, devidamente inicializado).

RESPOSTA: Desejamos um tratamento diferenciado para valores nulos. Qualquer coluna de um registro que não contenha informações é denominada de nula, portanto informação não existente. Isto não é o mesmo que "zero", pois zero é um número como outro qualquer, enquanto que um valor nulo utiliza um "byte" de armazenagem interna e são tratados de forma diferenciada pelo SQL.

```
SELECT EMPNOME, EMPSALA + EMPCOMI
```

```
FROM EMP;
```

```
SELECT EMPNOME, NVL(EMPSALA,0) + NVL(EMPCOMI,0)
```

```
FROM EMP;
```

Obs: a função "NVL" é utilizada para converter valores nulos em zeros.



Estudo Complementar

<http://www.htmlstaff.org/postgresqlmanual/sql-select.html>

http://www.hospedaria.com.br/artigos/10/mysql/1/mysql_basico_-_o_comando_select_-_realizando_consultas.html



Atividades

Resolva os exercícios apresentados nessa Unidade, sem ver as respostas, e veja quanto você acerta!!



UNIDADE 19

Objetivo: Explorar com maior profundidade o comando SELECT.

Cláusula e Operadores usados no Comando SELECT

Vimos na unidade anterior que na linguagem SQL temos o importante comando SELECT para consulta de registros de uma tabela num Banco de Dados. Porém, se queremos consultas mais complexas, teremos que usar algumas cláusulas e/ou operadores junto com o comando SELECT. Por exemplo, a cláusula ORDER BY modificará a ordem de apresentação do resultado da pesquisa de forma ascendente ou descendente.

Exemplo 6 - Liste os nomes e cargos da cada funcionário contidos na tabela empresa (EMP), porém classificados alfabeticamente (A.. Z) e depois alfabeticamente invertido (Z..A).

RESPOSTA:

```
SELECT FUN_NOME, FUN_CARG  
FROM EMP  
ORDER BY FUN_NOME;  
  
SELECT EMPNOME, EMPSERV  
FROM EMP  
ORDER BY EMPPNOME DESC;
```

NOTA: Também é possível fazer com que o resultado da pesquisa venha classificado por várias colunas. Sem a cláusula ORDER BY as linhas serão exibidas na sequência que o SGBD determinar.

Exemplo 7 - Selecione os Nomes dos Departamentos que estejam na fábrica.

RESPOSTA:

```
SELECT DEPNOME
FROM DEPT
WHERE DEPLOCA = "SAO PAULO";
```

O exemplo exigiu uma restrição (São Paulo) que nos obrigou a utilizar na instrução WHERE.

Demais Operadores

Operador	Significado
BETWEEN ... AND ...	entre dois valores (inclusive)
IN (....)	lista de valores
LIKE	com um padrão de caracteres
IS NULL	é um valor nulo

Exemplos:

```
SELECT EMPNOME, EMPSALA
FROM EMP
WHERE EMPSALA BETWEEN 500 AND 1000;
```

```
SELECT EMPNOME, DEPNUME
FROM EMP
WHERE DEPNUME IN (10,30);
```

```
SELECT EMPNOME, EMPSERV
```

```

FROM EMP

WHERE EMPNOME LIKE 'F%';

SELECT EMPNOME, EMPSERV

FROM EMP

WHERE EMPCOMI IS NULL;

```

O símbolo "%" pode ser usado para construir a pesquisa ("% = qualquer sequência de nenhum até vários caracteres).

Operadores Negativos

Operador	Descrição
<>	diferente
NOT NOME_COLUNA =	diferente da coluna
NOT NOME_COLUNA >	não maior que
NOT BETWEEN	não entre dois valores informados
NOT IN	não existente numa dada lista de valores
NOT LIKE	diferente do padrão de caracteres informado
IS NOT NULL	não é um valor nulo

Exemplo 8 - Selecione os Empregados cujos salários sejam menores que 1000 ou maiores que 3500.

RESPOSTA: Necessitaremos aqui a utilização de expressão negativa. A seguir apresentamos operadores negativos.

```
SELECT EMPNOME, EMPSALA  
FROM EMP  
WHERE EMPSALA NOT BETWEEN 1000 AND 3500;
```

Exemplo 9 - Apresente todos os funcionários com salários entre 200 e 700 e que sejam Vendedores.

RESPOSTA: Necessitaremos de consultas com condições múltiplas. Operadores AND (E) e OR (OU).

```
SELECT EMPNOME, EMPSALA, EMPSERV  
FROM EMP  
WHERE EMPSALA BETWEEN 700 AND 2000  
AND EMPSERV = 'VENDEDOR';
```

Exemplo 10 - Apresente todos os funcionários com salários entre 200 e 700 ou que sejam Vendedores.

RESPOSTA:

```
SELECT EMPNOME, EMPSALA, EMPSERV  
FROM EMP  
WHERE EMPSALA BETWEEN 700 AND 2000  
OR EMPSERV = 'VENDEDOR';
```

Exemplo 11 - Apresente todos os funcionários com salários entre 200 e 700 e que sejam Vendedores ou Balconistas.

RESPOSTA:

```

SELECT EMPNOME, EMPSALA, EMPSERV
FROM EMP
WHERE EMPSALA BETWEEN 700 AND 2000
AND ( EMPSERV = 'BALCONISTA' OR EMPSERV = 'VENDEDOR' );
    
```

Funções de Caracteres

Lower	Força caracteres maiúsculos aparecerem em minúsculos
Upper	Força caracteres minúsculos aparecerem em maiúsculos
Concat(x,y)	Concatena a string "x" com a string "y"
Substring(x,y,str)	Extrai um substring da string "str", começando em "x", e termina em "y"
To_Char(num)	Converte um valor numérico para uma string de caracteres
To_Date(char,fmt)	Converte uma string caracter em uma data
^Q	Converte data para o formato apresentado

Exemplo 12 - Apresente o nome de todos os empregados em letras minúsculas.

RESPOSTA:

```
SELECT LOWER( EMPNOME )
```

FROM EMP;

Exemplo 13 - Apresente o nome de todos os empregados (somente as 10 primeiras letras).

RESPOSTA:

```
SELECT SUBSTRING (1,10,EMPNAME)
```

FROM EMP;

Exemplo 14 - Apresente o nome de todos os empregados admitidos em 01/01/80.

RESPOSTA:

```
SELECT *
```

FROM EMP

```
WHERE EMPADMI = ^Q"DD-AAA-YYYY"("01-JAN-1980");
```

ou

```
SELECT *
```

FROM EMP

```
WHERE EMPADMI = ^Q("01-JAN-1980");
```



Estudo Complementar

<http://pgdocptbr.sourceforge.net/pg80/sql-select.html>





Atividades

Resolva os exercícios apresentados nessa Unidade, sem ver as respostas, e veja quanto você acerta!!



UNIDADE 20

Objetivo: Explorar mais recursos do complexo comando SELECT.

Funções Agregadas e a Cláusula HAVING

Funções Agregadas (ou de Agrupamento)

Função	Retorno
avg(n)	Média do valor n, ignorando nulos
count(expr)	Vezes que o número da expr avalia para algo não nulo
max(expr)	Maior valor da expr
min(expr)	Menor valor da expr
sum(n)	Soma dos valores de n, ignorando nulos

Exemplo 15 - Apresente a Média, o Maior, o Menor e também a Somatória dos Salários pagos aos empregados.

RESPOSTA:

```
SELECT AVG(EMPSALA) FROM EMP;
```

```
SELECT MIN(EMPSALA) FROM EMP;
```

```
SELECT MAX(EMPSALA) FROM EMP;
```

```
SELECT SUM(EMPSALA) FROM EMP;
```

Agrupamentos

As funções de grupo operam sobre grupos de registros (linhas). Retornam resultados baseados em grupos em vez de resultados de funções por registro individual. A cláusula GROUP BY do comando SELECT é utilizada para dividir registros em grupos menores.

A cláusula GROUP BY pode ser usada para dividir os registros de uma tabela em grupos menores. As funções de grupo devolvem uma informação sumarizada para cada grupo.

Exemplo 16 - Apresente a média de salário paga por departamento.

RESPOSTA:

```
SELECT DUPNUME, AVG(EMPSALA)
```

```
FROM EMP
```

```
GROUP BY DEPNUME;
```

Obs.: Qualquer coluna ou expressão na lista de seleção, que não for uma função agregada, deverá constar da cláusula GROUP BY. Portanto é errado tentar impor uma "restrição" do tipo agregada na cláusula WHERE.

Cláusula HAVING

A cláusula HAVING pode ser utilizada para especificar quais grupos deverão ser exibidos, portanto restringindo-os.

Exemplo 17 - Retome o problema anterior, porém apresente resposta apenas para departamentos com mais de 10 empregados.

RESPOSTA:

```
SELECT DEPNUME, AVG(EMPSALA)
```

```
FROM EMP
```

GROUP BY DEPNUME

HAVING COUNT(*) > 10;

Obs.: A cláusula GROUP BY deve ser colocada antes da HAVING, pois os grupos são formados e as funções de grupos são calculadas antes de se resolver a cláusula HAVING. E a cláusula WHERE não pode ser utilizada por restringir grupos que deverão ser exibidos.

Exemplificando ERRO típico - Restringindo Média Maior que 1000:

SELECT DEPNUME, AVG(EMPSALA)

FROM EMP

WHERE AVG(SALARIO) > 1000

GROUP BY DEPNUME;

(Esta seleção está ERRADA!)

SELECT DEPNUME, AVG(EMPSALA)

FROM EMP

GROUP BY DEPNUME

HAVING AVG(EMPSALA) > 1000;

(Seleção Adequada)

Sequência no comando SELECT:

SELECT coluna(s)

FROM tabela(s)

WHERE	Condição(ões) do(s) registro(s)
GROUP BY	Condição(ões) do(s) grupo(s) de registro(s)
HAVING	Condição(ões) do(s) grupo(s) de registro(s)
ORDER BY	Coluna(s);

O SQL fará a seguinte avaliação:

- WHERE, para estabelecer tuplas individuais candidatas (não pode conter funções de grupo)
- GROUP BY, para fixar grupos.
- HAVING, para selecionar grupos para exibição.



Estudo Complementar

<http://www.imasters.com.br/artigo/241>



Atividades

Antes de dar continuidade aos seus estudos é fundamental que você entre no site da ESAB e, em sua Sala de Aula, faça a Atividade 2, no link “Atividades”.



UNIDADE 21

Objetivo: Explorar os comandos SQL que trabalham com relacionamentos em SGBD.

Relacionamentos, sub-consulta e união

Equi-Junção (Junção por igualdade)

O relacionamento existente entre tabelas é chamado de equi-junção, pois os valores das colunas das duas tabelas são iguais. A Equi-junção é possível apenas quando tivermos definido de forma adequada a chave estrangeira de uma tabela e sua referência a chave primária da tabela precedente.

Apesar de admitir-se em alguns casos, a equi-junção de tabelas, sem a correspondente Chave Primária-Chave Estrangeira, recomendamos fortemente a não utilização desse tipo de construção.

Exemplo 18 - Listar Nomes de Empregados, Cargos e Nome do Departamento onde o empregado trabalha.

RESPOSTA:

Observemos que dois dos três dados solicitados estão na Tabela Emp, enquanto o outro dado está na Tabela Dept. Deveremos então acessar os dados restringindo convenientemente as relações existentes entre as tabelas.

De fato, sabemos que DEPNUME é chave primária da tabela de Departamentos e também é chave estrangeira da Tabela de Empregados. Portanto, este campo será o responsável pela equi-junção.

```
SELECT A.EMPNAME, A.EMPSERV, B.DEPNAME
```

```
FROM EMP A, DEPT B
```

```
WHERE A.DEPNUME = B.DEPNUME;
```

OBSERVAÇÃO: Note que as tabelas quando contêm colunas com o mesmo nome, usa-se um apelido (ALIAS) para substituir o nome da tabela associado à coluna. Imagine que alguém tivesse definido NOME para ser o Nome do Empregado na Tabela de Empregados e também NOME para ser o Nome do Departamento na Tabela de Departamentos. Tudo funcionaria de forma adequada, pois o ALIAS se encarregaria de evitar que uma ambiguidade fosse verificada.

Embora SQL resolva de forma muito elegante o problema da nomenclatura idêntica para campos de tabelas, recomendamos fortemente evitar tal forma de nomear os campos. O SQL nunca confundirá um A.NOME com um B.NOME, porém podemos afirmar o mesmo de nós mesmos?

Exemplo 19 - Liste os Códigos do Cada Funcionário, seus Nomes, seus Cargos e o nome do Gerente ao qual este se relaciona.

RESPOSTA:

Precisamos criar um auto-relacionamento, ou seja, juntar uma tabela a ela própria. É possível juntarmos uma tabela a ela mesma com a utilização de apelidos (permitindo juntar registros da tabela a outros registros da mesma tabela).

```
SELECT A.EMPNUME, A.EMPNAME, A.EMPSERV, B.EMPNAME
```

```
FROM EMP A, EMP B
```

```
WHERE A.EMPGERE = B.EMPNUME;
```

SubConsultas

Uma subconsulta é um comando SELECT que é aninhado dentro de outro SELECT e que devolve resultados intermediários.

Exemplo 20 - Relacione todos os nomes de funcionários e seus respectivos cargos, desde que o orçamento do departamento seja igual a 300.000

RESPOSTA:

```
SELECT EMPNOME, EMPSERV  
FROM EMP A  
WHERE 300000 IN ( SELECT DEPORCA  
FROM DEPT  
WHERE DEPT.DEPNUME = A.DEPNUME );
```

NOTA: Observe que a cláusula IN torna-se verdadeira quando o atributo indicado está presente no conjunto obtido através da subconsulta.

Exemplo 21 - Relacione todos os departamentos que possuem empregados com remuneração maior que 3.500

RESPOSTA:

```
SELECT DEPNOME  
FROM DEPT A  
WHERE EXISTS (SELECT *  
FROM EMP  
WHERE EMPSALA > 3500 AND EMP.DEPNUME = A.DEPNUME');
```

NOTA: Observe que a cláusula EXISTS indica se o resultado de uma pesquisa contém ou não registros. Observe também que poderemos verificar a não existência (NOT EXISTS), caso esta alternativa seja mais conveniente.

Unões

Podemos eventualmente unir duas linhas de consultas simplesmente utilizando a palavra reservada UNION.

Exemplo 22 - Liste todos os empregados que tenham códigos > 10 ou Funcionários que trabalhem em departamentos com código maior que 10.

RESPOSTA:

Poderíamos resolver esta pesquisa com um único SELECT, porém devido ao fato de estarmos trabalhando, em nosso exemplo, com apenas duas tabelas, não conseguimos criar um exemplo mais adequado para utilização deste recurso.

```
(SELECT *
  FROM EMP
  WHERE EMPNUME > 10)
UNION
(SELECT *
  FROM EMP
  WHERE DEPNUME > 10);
```



Estudo Complementar

www.jsoares.net/CEFET/BD/apostilaSQL.pdf



Atividades

Resolva os exercícios apresentados nessa Unidade, sem ver as respostas, e veja quanto você acerta!!



UNIDADE 22

Objetivo: Estudar os comandos SQL responsáveis por manipulações em tabelas.

Inserções, Alterações e Exclusões em Tabelas

Uma linguagem direcionada a extração de informações de um conjunto de dados não deveria incorporar comandos de manipulação de tabelas. No entanto, a mera existência de uma linguagem padronizada para acesso a dados "convida" os desenvolvedores a aderirem a uma linguagem "padrão" de manipulação de tabelas. E naturalmente cada fabricante coloca "um algo mais" em seu SQL (SQL PLUS, SQL *, ISQL, e toda sorte de nomenclaturas).

Isso por um lado, desvirtua os objetivos da linguagem (padronização absoluta), mas em contrapartida, otimiza os acessos ao Banco de Dados. E por maior que sejam essas mudanças, não chegam a impedir que um programador versado em SQL tenha grandes dificuldades em se adaptar ao padrão de determinada implementação.

De fato as diferenças entre o SQL da Sybase, Oracle, Microsoft, são muito menores dos que as existentes entre as linguagens "irmãs": C, BASIC e o Pascal (todas com origem conceitual do FORTRAN). Podemos observar que todas as três linguagens mencionadas possuem estruturas de controle, blocos de instrução, regras semelhantes para declaração de variáveis e usam comandos de tomada decisão, porém apesar de tantas semelhanças, é praticamente impossível que um programador excelente em uma linguagem consiga em curto espaço de tempo ser excelente em outra linguagem.

O mesmo não ocorrerá com um especialista em SQL, ao migrar de um SGBD para outro. Naturalmente existirá a necessidade de algum aprendizado, mas o programador poderá ir adaptando-se aos poucos sem precisar ser re-treinado, o que é um aspecto extremamente vantajoso para as empresas e para os próprios profissionais.

Inserir (INSERT)

INSERT INTO <tabela> [<campos>] [VALUES <valores>]

Exemplos:

INSERT INTO DEPT;

Possibilita a inserção de registros de forma interativa.

INSERT INTO DEPT (DEPNUME,DEPNOME,DEPLOCA) VALUES (70,"PRODUCAO","RIO DE JANEIRO");

Possibilita a inserção de registros em tabelas sem digitação dos dados.

Atualizar (UPDATE)

UPDATE <tabela> SET <campo> = <expressão> [WHERE <condição>];

Exemplo:

UPDATE EMP SET EMPSALA = EMPSALA * 1.2 WHERE EMPSALA < 1000;

Excluir (DELETE)

DELETE FROM <tabela> [WHERE <condição>];

Exemplo:

DELETE FROM emp WHERE EMPSALA > 5000;

Transações

Muitas vezes gostaríamos que determinado processo, caso fosse abortado por qualquer motivo, pudesse ser inteiramente cancelado. Imaginemos por exemplo um usuário digitando um pedido. Imaginemos ainda que o sistema possa reservar cada item solicitado de maneira

"on-line", ou seja, ao mesmo tempo em que se está digitando a quantidade, o sistema já "empenhe" uma quantidade equivalente no estoque. Imaginemos ainda que o sistema deva cancelar todas as operações se apenas um dos itens não puder ser atendido. Seria um grande problema, caso não pudéssemos anular todos os processos a partir de determinada condição.

Vamos simular tal ocorrência com nosso Banco de Dados EMP. Imaginemos que ao invés de digitarmos `DELETE FROM emp WHERE salário > 5000;` tivéssemos digitado `DELETE FROM emp WHERE salário > 500;` Ao invés de eliminarmos 2 registros, praticamente teríamos eliminado o Banco de Dados todo. Para evitarmos que um erro de digitação, ou um processo iniciado, porém sem condição de ser completado integralmente, comprometa todos nossos dados, podemos criar uma transação que nos assegurará que nossos testes sejam bem sucedidos ou cancelados sem comprometer os dados.

```
begin transaction;  
  
delete from emp where salario > 500;  
  
if SQL_RECORDCOUNT > 20 THEN;  
  
ROLLBACK TRASACTION;  
  
else  
  
COMMIT;  
  
endif;  
  
end transaction;
```

Visões (VIEW)

Uma visão consiste basicamente de uma tabela derivada de outras tabelas. Considerando o exemplo TRABALHO, poderíamos criar uma visão baseada na tabela de Empregados (EMP) e na tabela de Departamentos (DEPT) onde tivéssemos somente os Nomes dos Funcionários e os Departamentos nos quais estes trabalhassem. Teríamos algo assim:

```
CREATE VIEW EMP_DEP
AS SELECT E.EMPNAME, D.DEPNAME
FROM EMP E, DEPT D
WHERE E.DEPNUME = D.DEPNUME;
```

Devemos observar que:

- 1.Uma visão definida sobre uma única tabela somente será atualizável se os atributos da tal visão contiverem a chave primária.
- 2.Visões sobre várias tabelas não são passíveis de atualizações.
- 3.Visões que se utilizam de funções de agrupamentos, também não poderão ser atualizadas.



Estudo Complementar

<http://pt.wikipedia.org/wiki/CRUD>

<http://pgdocptbr.sourceforge.net/pg80/rules-update.html>



Atividades

Com o link do Wikipédia, referenciado no Estudo Complementar, verifique se todos os comandos abordados nesta unidade estão no acrônimo CRUD.



UNIDADE 23

Objetivo: Entender a sintaxe do comando SQL responsável em gerar relatórios.

Relatórios – comando REPORT

Sintaxe:

REPORT DISTINCT / UNIQUE

[atributo(s)]

REPORTTOP

PAGETOP

TOP

DETAIL

NONE

BOTTOM

PAGEBOTTOM

REPORTBOTTOM

FROM [tabela(s)]

[WHERE clausula-where]

[GROUP BY clausula-grupo]

[ORDER BY clausula-order by];

Como exemplo, converteremos um simples Select em um Report:

SELECT EMPNOME	REPORT
----------------	--------

<pre>FROM EMP WHERE DEPNUME = 1000;</pre>	<pre>DETAIL EMPNOME WHERE DEPNUME = 1000;</pre>
---	---

Podemos direcionar a saída de um relatório tanto para um arquivo como para uma impressora.

Para um arquivo: <pre>REPORT ON "RELAT.DAT" ...</pre>	Para uma impressora: <pre>REPORT ON LP:" ...</pre>
---	--

Agora vamos incrementar mais o REPORT (veja na próxima folha):

REPORT

REPORTTOP COL 10, “*** RELATORIO DE FUNCIONARIOS *** ”,

TODAY %Q”DD/MM/YY”, SKIP,

COL 10, “=====”, SKIP 2

DETAIL COL 10, NOME %C22, SALARIO %FS, ADMISSAO %Q”DD/MM/YY”

REPORTBOTTOM COL 10,

“=====”, SKIP,

COL 20, “TOTAL:”, TOTAL(SALARIO)

FROM EMP

ORDER BY NOME;

Onde:

REPORTTOP	O que será impresso no topo do relatório
PAGETOP	Impresso em cada topo de página
TOP	Impresso em cada topo do Sort (Grupo do Relatório)
DETAIL	O que será impresso em cada linha
NONE	Se o select não tiver resultado, não será impresso o relatório
BOTTOM	Impresso em cada rodapé do Sort (Grupo do relatório)
PAGEBOTTOM	O que será impresso no rodapé de cada página
REPORTBOTTOM	O que será impresso no rodapé do relatório

Formatos	Tipos
%C	caracter
%D	data
Y	ano
M	mês numérico
A	mês alfanumérico
D	dia
J	dia e ano Juliano

Exemplo: %D"dd/mm/yy"

%I	inteiro
%F	ponto flutuante

%FSZ	onde: S - separador de 3 dígitos e decimal point e Z - zeros serão suprimidos
%Q	Data
%J	Hora
H	Hora
M	Minutos
S	Segundos
%T	Hora

E temos as funções: TOTAL, AVERAGE, MAXIMUM, MINIMUM.



Estudo Complementar

<http://www.activedelphi.com.br/modules.php?op=modload&name=News&file=article&sid=53&mode=thread&order=0&thold=0>



Atividades

Com base no link do Estudo Complementar, aproveite para fazer um bom resumo de tudo o que já vimos até agora de comandos SQL.



UNIDADE 24

Objetivo: Estudar como definir privilégios específicos conforme as necessidades dos usuários

Privilégios de Acesso – GRANT e REVOKE

Como a grande maioria dos Bancos de Dados Relacionais pode ser acessada por diversos usuários, cada um possui necessidade específica em relação aos dados armazenados. De acordo com o projeto do Banco de Dados, alguns usuários só podem consultar alguns dados, outros podem atualizar, outros podem inserir, etc. Para que o dado fique protegido do uso indevido de algum usuário, a linguagem SQL permite a definição de privilégios que cada um pode ter em relação às tabelas. Os privilégios garantem a segurança e a integridade dos dados, bem como a responsabilidade de cada usuário sobre seus dados específicos.

1. Comando GRANT (garantir)

Quando uma tabela/view é criada, o nome do usuário que a criou é anexado, internamente, ao nome da tabela. Por exemplo: se a tabela produto foi criada pelo usuário Felipe, então internamente ela será conhecida como Felipe - produto.

O criador da tabela/view tem total privilégio sobre a tabela criada, podendo disponibilizar qualquer privilégio para outros usuários pelo comando GRANT.

Sintaxe:

GRANT {ALL | Lista de privilégios}

ON {nome da tabela/view (lista de colunas)}

TO {PUBLIC | Lista de usuários}

[WITH GRANT OPTION]

A palavra ALL indica que estão sendo dados TODOS privilégios, ou então especificamos qual o privilégio que está sendo dado (SELECT, UPDATE, etc.).

A cláusula ON especifica a tabela ou view e suas colunas para as quais está sendo dado privilégio.

Dica: Somente pode ser utilizada uma tabela ou view por comando.

Os privilégios podem ser especificados para algumas colunas, porém devem ser todas da mesma tabela. Se não for especificada nenhuma coluna, os privilégios valem para todas.

A Cláusula opcional WITH GRANT OPTION permite que quando se dá o privilégio a um usuário, ele passe esse privilégio para os outros usuários.

Lista de opções de privilégios:

Select	Pode executar uma consulta sobre a tabela
Insert	Pode executar uma inserção sobre a tabela
Delete	Pode apagar registros da tabela
Update	Pode modificar registros da tabela
All privileges / all	Pode executar qualquer operação sobre a tabela
<usuário>	Nome do usuário que vai receber os privilégios
PUBLIC	Concede privilégios a todos os usuários do ambiente

Exemplos:

Permite somente consultas ao usuário Jorge sobre a tabela Produto.

GRANT Select

ON Produto

TO Jorge;

Concede ao usuário Gilmar, os privilégios de seleção, inserção e alteração sobre a tabela Pedido.

GRANT Select, Insert, Update

ON Pedido

TO Gilmar;

Permite todos os privilégios a todos os usuários sobre a tabela Cliente.

GRANT All privileges

ON Cliente

TO PUBLIC;

Concede aos usuários Felipe e Carlos, o privilégio de seleção sobre a tabela Cliente.

GRANT Select

ON Cliente

TO Felipe, Carlos;

Disponibilizar para seleção, somente os campos código de vendedor e nome de vendedor da tabela Vendedor a todos os usuários.

GRANT Select (Ven_Codi, Ven_nome)

ON Vendedor

TO PUBLIC;

Conceder ao usuário Jorge o poder de permitir a concessão de todos os privilégios a outros usuários sobre a tabela PEDIDO.

GRANT ALL

ON Pedido

TO Jorge

WITH GRANT OPTION;

2.Comando REVOKE (revogação)

Da mesma forma que o criador da tabela pode garantir os privilégios de acesso aos outros usuários (GRANT), ele pode revogar esses privilégios por meio do comando REVOKE.

Sintaxe:

REVOKE [Lista de privilégios]

ON [nome da tabela / view]

FROM [Lista de usuários]

Retirar o privilégio de seleção sobre a tabela Produto do usuário Carlos.

REVOKE select

ON Produto

From Carlos;

Revogar todos os privilégios concedidos a todos os usuários sobre a tabela Cliente.

REVOKE select

ON Cliente

FROM PUBLIC;

Retirar os privilégios de atualização e inserção concedidos ao usuário Felipe sobre a tabela Pedido.

REVOKE Insert, Update

ON Pedido

FROM Felipe;



Estudo Complementar

<http://dev.mysql.com/doc/refman/4.1/pt/grant.html>

<http://www.htmlstaff.org/postgresqlmanual/sql-grant.html>

<http://www.htmlstaff.org/postgresqlmanual/sql-revoke.html>



Atividades

Com base nos Manuais de Referência do MySQL, e do PostgreSQL, vistos no Estudo Complementar, veja a diferença que possa existir.



UNIDADE 25

Objetivo: Definir índices e estratégias para a sua melhor utilização.

Trabalhando com Índices

Índice é uma estrutura que permite rápido acesso às linhas de uma tabela, com base nos valores de uma ou mais colunas. O índice é simplesmente outra tabela no Banco de Dados, na qual estão armazenados valores e ponteiros (arrumados de forma ascendente ou descendente).

O SGBD utiliza os índices para pesquisar rapidamente um determinado valor dentro do Banco de Dados. Por intermédio dos ponteiros se localiza em que linha o valor desejado está armazenado. As tabelas de índices são utilizadas internamente pelo SGBD, ficando totalmente transparente ao usuário.

Quando vamos criar índices e colunas, devemos considerar:

Criar índice sobre valores que vamos pesquisar com frequência.

Indexe suas chaves estrangeiras quando precisar fazer consultas mais eficientes e rápidas.

As colunas que são regularmente utilizadas em consultas devem ser indexadas porque o sistema pode otimizar estas consultas, deixando-as mais rápidas.

Crie índice sempre em colunas que são pesquisadas por um intervalo de valores.

E por fim, crie índice sempre em colunas que são utilizadas em cláusulas WHERE.

Criando índices

Cada índice é aplicado a uma tabela, especificando uma ou mais colunas dessa tabela. Índices são criados por meio do comando CREATE INDEX, que cria um índice sobre colunas de uma tabela.

Sintaxe:

```
CREATE [UNIQUE] INDEX <nome do índice>
```

```
ON <nome da tabela> (<coluna(s)>);
```

A cláusula UNIQUE é opcional e define para aquela coluna não existirão duplicados, ou seja, todos os dados armazenados na coluna serão ÚNICOS.

A junção do índice unique e da especificação NOT NULL para uma coluna define a chave primária da tabela quanto ao aspecto lógico, pois uma chave primária, como vimos, não pode ser NULA.

A criação dos índices depende muito do projeto do Banco de Dados e das necessidades de pesquisa formuladas pelos usuários do Banco de Dados. Os índices estão muito ligados às necessidades de velocidade na recuperação da informação, e na execução rápida de uma operação de consulta.

Para cada SGBD existem cláusulas específicas operacionais que devem ser usadas, mas neste caso vamos apresentar a sintaxe padrão geral do SQL ANSI.

Exemplos:

Cria a tabela de índices chamada nome_pro baseada no campo nome_produto da tabela Produto.

```
CREATE INDEX nome_pro
```

```
ON Produto (nome_produto);
```

Cria a tabela de índices ped_pro baseada na concatenação dos campos num_pedido e cod_produto da tabela item_pedido

```
CREATE INDEX ped_pro
```

```
ON item_pedido (num_pedido, cod_produto);
```

É importante considerar que praticamente todas as sintaxes em se tratando de SGBDs relacionais exigem que se identifique o database proprietário da tabela, principalmente no Microsoft SQL Server.

Cria o índice único para a tabela Cliente baseada no código do cliente, não podendo haver duplicidade de informação armazenada.

```
CREATE UNIQUE INDEX cliente_ind
```

```
ON [nome do database] cliente (cod_cliente);
```

Eliminando Índices

Da mesma forma que um índice é criado, ele pode ser eliminado, dependendo das necessidades do projeto do Banco de Dados.

Sintaxe:

```
DROP INDEX <nome do índice>;
```

Exemplos:

Elimina o índice que foi criado para o nome do produto

```
DROP INDEX nome_pro;
```

Elimina o índice criado para o código do cliente

```
DROP INDEX cod_cliente;
```



Estudo Complementar

<http://www.htmlstaff.org/postgresqlmanual/sql-createindex.html>

<http://www.htmlstaff.org/postgresqlmanual/sql-dropindex.html>



Atividades

Ao consultar o Manual de Referência do PostgreSQL citado no Estudo Complementar, compare com a sintaxe apresentada nesta unidade.



UNIDADE 26

Objetivo: Rever comandos SQL e os específicos de outros SGBDs.

Resumo dos Comandos SQL

Esta Unidade contém informações de referência para os comandos SQL e também aqueles suportados pelo PostgreSQL (ver mais detalhes na Unidade 30).

Table of Contents

ABORT -- aborta a transação corrente

ALTER GROUP -- inclui ou exclui usuários em um grupo

ALTER TABLE -- altera a definição da tabela

ALTER USER -- altera a conta de um usuário do Banco de Dados

ANALYZE -- coleta estatísticas sobre um Banco de Dados

BEGIN -- inicia um bloco de transação

CHECKPOINT -- força um ponto de controle no log de transação

CLOSE -- fecha o cursor

CLUSTER -- agrupa uma tabela de acordo com um índice

COMMENT -- cria ou altera o comentário de um objeto

COMMIT -- efetiva a transação corrente

COPY -- copia dados entre arquivos e tabelas

CREATE AGGREGATE -- define uma nova função de agregação

CREATE CONSTRAINT TRIGGER -- define um novo gatilho de restrição

CREATE DATABASE -- cria um Banco de Dados novo

CREATE FUNCTION -- define uma nova função

CREATE GROUP -- define um novo grupo de usuários

CREATE INDEX -- define um índice novo

CREATE LANGUAGE -- define uma nova linguagem procedural

CREATE OPERATOR -- define um novo operador

CREATE RULE -- define uma nova regra

CREATE SEQUENCE -- define um novo gerador de sequência

CREATE TABLE -- define uma nova tabela

CREATE TABLE AS -- cria uma nova tabela a partir do resultado de uma consulta

CREATE TRIGGER -- define um novo gatilho

CREATE TYPE -- define um novo tipo de dado

CREATE USER -- define uma nova conta de usuário do Banco de Dados

CREATE VIEW -- define uma nova visão

DECLARE -- define um cursor

DELETE -- exclui linhas de uma tabela

DROP AGGREGATE -- remove uma função de agregação definida pelo usuário

DROP DATABASE -- remove um Banco de Dados

DROP FUNCTION -- remove uma função definida pelo usuário

DROP GROUP -- remove um grupo de usuários

DROP INDEX -- remove um índice

DROP LANGUAGE -- remove uma linguagem procedural definida pelo usuário

DROP OPERATOR -- remove um operador definido pelo usuário

DROP RULE -- remove uma regra

DROP SEQUENCE -- remove uma sequência

DROP TABLE -- remove uma tabela

DROP TRIGGER -- remove um gatilho

DROP TYPE -- remove um tipo de dado definido pelo usuário

DROP USER -- remove uma conta de usuário do Banco de Dados

DROP VIEW -- remove uma visão

END -- efetiva a transação corrente

EXPLAIN -- mostra o plano de execução de uma instrução

FETCH -- busca linhas de uma tabela usando um cursor

GRANT -- define privilégios de acesso

INSERT -- cria novas linhas na tabela

LISTEN -- escuta uma notificação

LOAD -- carrega ou recarrega um arquivo de biblioteca compartilhada

LOCK -- bloqueia explicitamente uma tabela

MOVE -- posiciona o cursor em uma determinada linha da tabela

NOTIFY -- gera uma notificação

REINDEX -- reconstrói índices corrompidos

RESET -- atribui a um parâmetro de tempo de execução o seu valor padrão

REVOKE -- revoga privilégios de acesso

ROLLBACK -- aborta a transação corrente

SELECT -- retorna linhas de uma tabela ou de uma visão

SELECT INTO -- cria uma nova tabela a partir do resultado de uma consulta

SET -- muda um parâmetro de tempo de execução

SET CONSTRAINTS -- especifica o modo de restrição da transação corrente

SET SESSION AUTHORIZATION -- define o identificador do usuário da sessão e o identificador do usuário corrente, da sessão corrente.

SET TRANSACTION -- define as características da transação corrente

SHOW -- mostra o valor de um parâmetro de tempo de execução

TRUNCATE -- esvazia a tabela

UNLISTEN -- pára de escutar uma notificação

UPDATE -- atualiza linhas de uma tabela

VACUUM -- limpa e opcionalmente analisa o Banco de Dados



Estudo Complementar

<http://www.htmlstaff.org/postgresqlmanual/index.html>





Atividades

Navegue pelo site da PostgreSQL, e revise os principais comandos SQL.



UNIDADE 27

Objetivo: Aprender as técnicas básicas de otimização das consultas SQL.

Otimizar Consultas SQL

Existem diferentes formas de otimizar as consultas realizadas em SQL. A linguagem SQL é não procedural, ou seja, nas sentenças se indica o que queremos conseguir e não como. Isso seria na teoria, pois na prática todos os gerenciadores de SQL têm que especificar seus próprios truques para otimizar o rendimento.

Portanto, se quisermos que o tempo de resposta seja o mínimo, não basta especificar uma sentença SQL correta, e sim indicar como tem que fazer. Nesta seção, veremos como melhorar o tempo de resposta de nosso intérprete ante as determinadas situações:

Design de tabelas

Normalize as tabelas, pelo menos até a terceira forma normal, para garantir que não haja duplicidade de dados. Se tiver que desnormalizar alguma tabela pense na ocupação e no rendimento antes de proceder.

Os primeiros campos de cada tabela devem ser aqueles campos requeridos, e primeiro se definem os de longitude fixa e depois os de longitude variável. Ajuste ao máximo o tamanho dos campos para não desperdiçar espaço. É normal deixar um campo de texto para observações nas tabelas. Se este campo for utilizado com pouca frequência ou se for definido com grande tamanho, por via das dúvidas, é melhor criar uma nova tabela que contenha a chave primária da primeira e o campo para observações.

Gerenciamento e escolha dos índices

Os índices são campos escolhidos pelo construtor do Banco de Dados e que permitem a busca de um campo a uma velocidade notavelmente superior. Entretanto, ocupam muito memória (o dobro mais ou menos), e requer para sua inserção e atualização, um tempo de processo superior.

Evidentemente, não podemos indexar todos os campos de uma tabela extensa já que dobramos o tamanho do Banco de Dados. Igualmente, tampouco serve indexar todos os campos numa tabela pequena já que as seleções podem se efetuar rapidamente de qualquer forma.

Um caso em que os índices podem ser muito úteis é quando realizamos petições simultâneas sobre várias tabelas. Neste caso, o processo de seleção pode se acelerar sensivelmente se indexamos os campos que servem de nexo entre as duas tabelas.

Campos a Selecionar

Na medida do possível devem-se evitar sentenças SQL que estejam embebidas dentro do código da aplicação. É muito mais eficaz usar vistas ou procedimentos armazenados, porque o gerenciador os salva compilados. Tratando-se de uma sentença embebida, o gerenciador deve compilá-la antes de executá-la.

Selecionar exclusivamente aqueles que se necessitem

Não utilizar o SELECT * porque o gerenciador deve ler primeiro a estrutura da tabela antes de executar a sentença. Se utilizar várias tabelas na consulta, especifique sempre a que tabela pertence cada campo, isso economizará tempo ao gerenciador de localizá-los. Ao invés de:

SELECT Nome, Fatura FROM Clientes, Faturamento WHERE IdCliente = IdClienteFaturado;

utilize:

SELECT Clientes.Nome, Faturamento.Fatura WHERE Clientes.IdCliente = Faturamento.IdClienteFaturado;

Interrogar sempre por campos que sejam chave

Se desejarmos interrogar por campos pertencentes a índices compostos é melhor utilizar todos os campos de todos os índices. Suponhamos que temos um índice formado pelo campo NOME e o campo SOBRENOME e outro índice formado pelo campo IDADE. A sentença WHERE NOME='Jose' AND SOBRENOME Like '%' AND IDADE = 20 seria melhor que WHERE NOME = 'Jose' AND IDADE = 20 porque o gerenciador, neste segundo caso, não pode usar o primeiro índice e ambas as sentenças são equivalentes porque a condição SOBRENOME Like '%' devolveria todos os registros.

Ordem das Tabelas

Quando se utilizam várias tabelas dentro da consulta tomar cuidado com a ordem empregada na cláusula FROM. Se desejarmos saber quantos alunos se matricularam no ano 1996 e escrevermos: FROM Alunos, Matriculas WHERE Aluno.IdAluno = Matriculas.IdAluno AND Matriculas.Ano = 1996 o gerenciador percorrerá todos os alunos para buscar suas matrículas e devolver as correspondentes. Se escrevermos FROM Matriculas, Alunos WHERE Matriculas.Ano = 1996 AND Matriculas.IdAluno = Alunos.IdAlunos, o gerenciador filtra as matrículas e depois seleciona os alunos, desta forma percorre menos registros.



Estudo Complementar

<http://sqlcomoumtodo.wordpress.com/2007/09/29/como-criar-consultas-sql-mais-rapidas/>





Atividades

Realize a Leitura Complementar ao excelente artigo de Flavio Augusto Weber e Elaini Simoni Angelotti, intitulado “Otimizando Consultas SQL no ambiente SQLServer”, para conhecer melhor esse ambiente:

<http://www.pr.gov.br/batebyte/edicoes/2003/bb132/otimizando.shtml>



UNIDADE 28

Objetivo: Mostrar os parametros necessários para disparar um gatilho para um determinado evento em um Banco de Dados.

Comando CREATE TRIGGER para criação de um gatilho

O gatilho define um conjunto de ações a serem executadas quando ocorre um evento de Banco de Dados em uma determinada tabela. O evento de Banco de Dados pode ser uma operação de exclusão, inserção ou de atualização. Por exemplo, se for definido um gatilho para exclusão em uma determinada tabela, a ação do gatilho ocorre sempre que se remove uma ou mais linhas da tabela.

Junto com as restrições, os gatilhos podem ajudar a impor regras de integridade com ações como exclusões ou atualizações em cascata. Os gatilhos também podem realizar várias funções como emitir alertas, atualizar outras tabelas, enviar e-mail, e outras ações úteis.

Pode ser definido qualquer número de gatilhos para uma única tabela, inclusive vários gatilhos para a mesma tabela para o mesmo evento. Pode ser criado gatilho em qualquer esquema, exceto os começados por SYS. O gatilho não precisa residir no mesmo esquema da tabela para a qual é definido. Se for especificado um nome de gatilho qualificado, o nome do esquema não poderá começar por SYS.

Sintaxe

```
CREATE TRIGGER nome-do-gatilho
```

```
{ BEFORE | AFTER } { INSERT | DELETE | UPDATE [ OR ... ] }
```

```
ON nome-da-tabela [FOR EACH { ROW | STATEMENT }]
```

```
EXECUTE PROCEDURE nome-da-função
```

Disparar Gatilhos

Os gatilhos são definidos como: BEFORE (antes) ou AFTER (depois).

Os gatilhos BEFORE disparam antes das modificações da instrução serem aplicadas, e antes de qualquer restrição ser aplicada.

Os gatilhos AFTER disparam após todas as restrições terem sido satisfeitas, e após todas as alterações terem sido aplicadas à tabela de destino.

Tanto o gatilho BEFORE, como o AFTER podem ser tanto de linha (ROW) quanto de instrução (STATEMENT).

Inserção, exclusão e atualização: o que faz o gatilho disparar

O gatilho é disparado por um dos seguintes eventos do Banco de Dados, dependendo de como foi definido:

INSERT

DELETE

UPDATE

O gatilho pode ser especificado para disparar antes de tentar realizar a operação na linha ou após a operação estar completa. Se o gatilho for disparado antes do evento, o gatilho pode evitar a operação para a linha corrente, ou modificar a linha sendo inserida (para as operações de INSERT e UPDATE somente). Se o gatilho for disparado após o evento, todas as mudanças, incluindo a última inserção, atualização ou exclusão, são "visíveis" para o gatilho.

Se o gatilho estiver marcado como FOR EACH ROW então ele é chamado uma vez para cada linha modificada pela operação. Por exemplo, um comando DELETE afetando 10 linhas faz

com que todos os gatilhos ON DELETE da relação de destino sejam chamados 10 vezes, uma vez para cada linha excluída. Por outro lado, um gatilho marcado como FOR EACH STATEMENT somente executa uma vez para uma determinada operação, independente de quantas linhas sejam modificadas. Em particular, uma operação que não modifica nenhuma linha, ainda assim resulta na execução de todos os gatilhos FOR EACH STATEMENT aplicáveis.

Se vários gatilhos do mesmo tipo estão definidos para o mesmo evento, estes são disparados na ordem alfabética de seus nomes. O SELECT não modifica nenhuma linha e, portanto, não é possível criar gatilhos para SELECT. Regras e visões são mais apropriadas neste caso.

Parâmetros Utilizados

`CREATE TRIGGER nome-do-gatilho`

Nome dado ao novo gatilho, devendo ser distinto do nome de qualquer outro gatilho para a mesma tabela.

`{ BEFORE | AFTER }`

Determina se a função é chamada antes ou depois do evento.

`{ INSERT | DELETE | UPDATE [OR ...] }`

Especifica o evento que dispara o gatilho. Vários eventos podem ser especificados utilizando OR.

`ON nome-da-tabela`

O nome (opcionalmente qualificado pelo esquema) da tabela que o gatilho se destina.

`[FOR EACH { ROW | STATEMENT }]`

Especifica se o procedimento do gatilho deve ser disparado uma vez para cada linha afetada pelo evento do gatilho, ou apenas uma vez para a declaração SQL. Se nenhum dos dois for especificado, FOR EACH STATEMENT é usado por padrão.

EXECUTE PROCEDURE nome-da-função

Uma função fornecida pelo usuário, declarada como não recebendo nenhum argumento e retornando o tipo trigger, que é executada quando o gatilho dispara.

Exemplo:

```
CREATE TRIGGER tafter
```

```
AFTER INSERT OR UPDATE OR DELETE
```

```
ON ttest FOR EACH ROW
```

```
EXECUTE PROCEDURE trigf();
```

Observações:

Para poder criar um gatilho em uma tabela, o usuário deve possuir o privilégio TRIGGER na tabela. Deve ser utilizado o comando DROP TRIGGER para remover um gatilho.



Estudo Complementar

<http://htmlstaff.org/postgresqlmanual/sql-createtrigger.html>
<http://www.javalinux.com.br/javalinux/pg74/sql-createtrigger.html>



Atividades

Para você se ambientar ao PL/SQL da ORACLE, veja no final desse artigo como eles conseguem TRIGGER:
http://www.sqlmagazine.com.br/artigos/oracle/04_Intro_PLSQL.asp



UNIDADE 29

Objetivo: Apresentar aspectos de segurança e proteção num SGBD.

Protegendo seu Servidor de Banco de Dados

Geralmente Bancos de Dados contêm dados muito confidenciais (por exemplo: detalhes pessoais de recursos humanos, detalhes sobre clientes, ordens de compra ou detalhes sobre cartão de crédito). Esses dados devem ser armazenados com segurança e protegidos contra divulgação não autorizada, violação ou uso mal intencionado. O servidor de Banco de Dados, mesmo que não esteja diretamente conectado à Internet, precisa ser protegido contra ataques que exploram os pontos fracos da configuração, estouros de buffer existentes ou práticas de desenvolvimento ineficazes. Esses ataques podem ser executados, por exemplo:

- Um aplicativo da Web desprotegido, usado para explorar o Banco de Dados.
- Um administrador mal intencionado com acesso à rede.
- Um usuário do Banco de Dados que, sem saber, executa códigos mal intencionados.
- Um invasor pode visar e comprometer um servidor de Banco de Dados de diversas formas ao explorar várias configurações e vulnerabilidades no nível do aplicativo. As principais ameaças para um servidor de Banco de Dados são:

Inclusão de código SQL

Com um ataque de inclusão de código SQL, o invasor explora as vulnerabilidades do código de acesso a dados. Outro ponto é a validação da entrada do aplicativo para executar comandos arbitrários que usa o contexto de segurança do aplicativo da Web.

Espionagem na rede

A arquitetura de implantação da maioria dos aplicativos inclui uma separação física entre o código de acesso a dados e o servidor de Banco de Dados. Consequentemente, os dados confidenciais, como dados específicos do aplicativo ou credenciais de logon ao Banco de Dados, devem ser protegidos contra a espionagem na rede.

Acesso não autorizado ao servidor

O acesso direto ao servidor de Banco de Dados deve ser restrito a computadores cliente para impedir o acesso não autorizado ao servidor. Ataques de conexão direta existem para usuários autenticados, e também para usuários sem nome de usuário e sem senha.

Quebra de senha

Uma primeira linha de ataque comum é tentar quebrar as senhas de nomes de conta conhecidos, como a conta do administrador do SGBD.

A figura a seguir mostra as principais ameaças e vulnerabilidades que podem resultar em um servidor de Banco de Dados. E visto também a possibilidade de destruição ou no roubo de dados confidenciais.

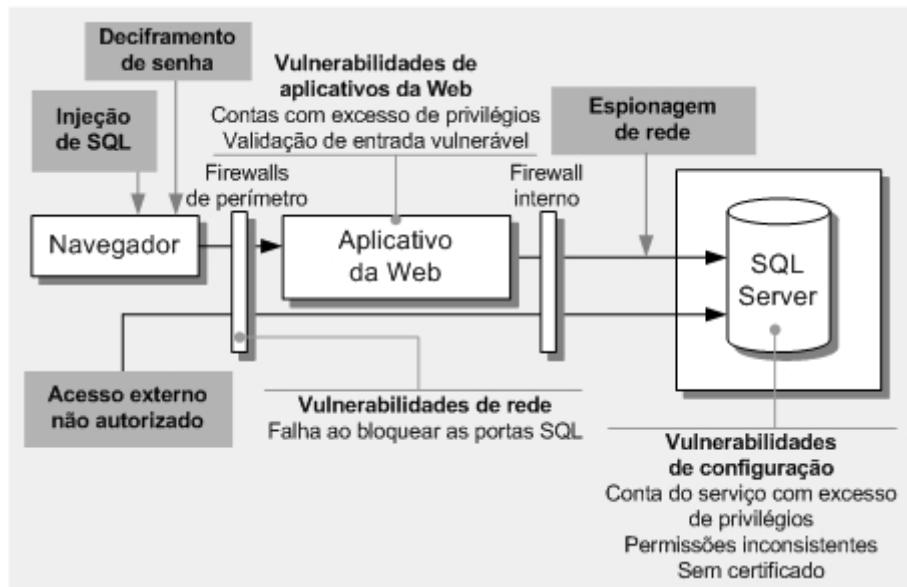


Figura - Principais ameaças e vulnerabilidades de um SGBD

Algumas medidas de segurança do servidor de Banco de Dados têm como base as práticas recomendadas obtidas em experiências reais, na validação de clientes e em estudos de implantações de segurança. Algumas medidas de segurança recomendadas são:

Patches e atualizações

Muitas ameaças de segurança existem devido a vulnerabilidades nos Sistemas Operacionais e aplicativos. Geralmente, quando novas vulnerabilidades são descobertas, o código de ataque é publicado na Internet, poucas horas após o primeiro ataque. O patch e a atualização do software do servidor são a primeira etapa para a proteção do servidor de Banco de Dados.

Serviços

Os serviços são os pontos de vulnerabilidade preferenciais dos invasores que exploram os privilégios e os recursos do serviço para acessar o servidor. Alguns serviços foram criados para funcionar usando contas com privilégios. Se esses serviços estiverem comprometidos, o invasor poderá efetuar operações privilegiadas. Por padrão, os servidores de Banco de

Dados não precisam de todos os serviços ativados. Ao desativar os serviços desnecessários, e não utilizados, você reduz de maneira rápida e fácil a área de superfície de ataque.

Protocolos

Limite o intervalo de protocolos que computadores cliente podem usar para estabelecer conexão com o servidor de Banco de Dados e verifique se é possível proteger esses protocolos.

Contas

Restrinja o número de contas do Windows acessíveis pelo servidor de Banco de Dados ao conjunto necessário de contas de usuário e de serviço. Use contas com menos privilégios e com senhas de alta segurança. Uma conta com menos privilégios, usada para executar o SQL Server, limita os recursos de um invasor que compromete o SQL Server e que consegue executar comandos do sistema operacional.

Arquivos e diretórios

Use as permissões do sistema de arquivos NTFS para proteger programas, Bancos de Dados e arquivos de log contra o acesso não autorizado. Quando você usa ACLs (Listas de Controle de Acesso) junto com a auditoria do Windows, é possível detectar a ocorrência de atividades suspeitas ou não autorizadas.

Compartilhamentos

Remova todos os compartilhamentos de arquivos desnecessários, incluindo os compartilhamentos de administração padrão caso não sejam necessários. Proteja quaisquer compartilhamentos restantes com permissões NTFS restritas.

Portas

As portas não utilizadas estão fechadas no firewall, mas é necessário que os servidores subjacentes ao firewall também bloqueiem ou restrinjam as portas com base no uso. Para um SGBD dedicado, bloqueie todas as portas, exceto as portas necessárias para autenticação.

Auditoria e log

A auditoria é uma ajuda vital para a identificação de intrusos e de ataques em andamento, bem como para o diagnóstico de sinais de ataque. Configure um nível mínimo de auditoria para o servidor de Banco de Dados usando uma combinação de recursos de auditoria do Windows e do SGBD.



Estudo Complementar

<http://www.microsoft.com/brasil/security/guidance/default.mspx>

<http://forums.microsoft.com/technet-br>

<http://www.activedelphi.com.br/modules.php?op=modload&name=News&file=article&sid=293&mode=thread&order=0&thold=0>





Atividades

Pesquise na Internet casos de invasão em Banco de Dados



UNIDADE 30

Objetivo: Fazer um comparativo dos vários SGBD usados atualmente no mundo corporativo.

Características dos vários SGBDs de uso Comercial

Vamos aqui mencionar algumas características dos principais SGBDs hoje utilizados nos mais variados segmentos da atividade humana. Entre os vários SGBDs usados atualmente para diversas utilidades e propósitos, podemos dividi-los em SGBDs free (licença gratuita) e os SGBDs proprietários.

Referente aos SGBDs free vamos analisar o MySQL e o PostgreSQL. Referente aos SGBDs proprietários vamos mencionar o SQL Server da Microsoft, o Oracle e o DB2 da IBM. Com relação ao Access do pacote Office da Microsoft, muitos não consideram como sendo um SGBD devido a sua relativa simplicidade.

MySQL

O MySQL é um SGBD, que utiliza a linguagem SQL como interface. É atualmente um dos Bancos de Dados mais populares. O MySQL foi criado na Suécia por dois suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius. Todos esses desenvolvedores trabalham juntos desde a década de 1980. Hoje em seu desenvolvimento e manutenção, empregam aproximadamente 70 profissionais no mundo inteiro. Mais de mil pessoas contribuem testando o software, integrando-o a outros produtos, e escrevendo a respeito do mesmo.

O sucesso do MySQL deve-se em grande parte à fácil integração com o PHP. Essa linguagem é oferecida atualmente, quase que obrigatoriamente, nos pacotes de hospedagem de sites na Internet. O Wikipédia é um exemplo de utilização do MySQL com grande eficiência e robustez. Algumas de suas características são:

- Portabilidade (suporta praticamente qualquer plataforma atual);
- Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como Delphi, Java, C/C++, Python, Perl, PHP e Ruby);
- Excelente desempenho e estabilidade;
- Pouco exigente quanto a recursos de hardware;
- Facilidade de uso;
- Software Livre;
- Suporte a vários tipos de tabelas (como MyISAM e InnoDB).

No entanto, mesmo com todas essas características positivas, ainda faltam alguns recursos avançados quando comparados como outros Banco de Dados, como o PostgreSQL.

PostgreSQL

PostgreSQL é um Sistema Gerenciador de Banco de Dados Objeto Relacional (SGBDOR), desenvolvido como projeto software livre. Inicialmente foi desenvolvido na Universidade de Berkeley, Califórnia. A partir de 1996, o PostgreSQL firmou-se como um projeto de software open source e começou a ser desenvolvido por uma comunidade de voluntários sob a coordenação do PostgreSQL Global Development Group.

Embora as atividades do grupo sejam patrocinadas por diversas organizações de todo o mundo, seu desenvolvimento é feito por um grupo de desenvolvedores, em sua maioria voluntários, espalhados por todo o mundo e que se comunicam via Internet. Hoje, o PostgreSQL é um dos SGBD de código aberto mais avançados, contando com recursos especiais tais como:

- Consultas complexas;

- Chaves estrangeiras;
- Integridade transacional;
- Controle de concorrência multiversão;
- Suporte ao modelo híbrido objeto-relacional;
- Triggers;
- Views;
- Stored procedures em várias linguagens.

SQL Server

O SQL Server é um Gerenciador de Banco de Dados Relacional feito pela Microsoft. É um Banco de Dados robusto e usado por sistemas corporativos dos mais diversos portes. Entre os novos recursos está a integração com o Framework .Net, que possibilita construir rotinas utilizando as linguagens do .Net como VB.Net e C#.

Existe a necessidade de se destacar que não se deve confundir esse específico SGBD, com a linguagem SQL que é geral e criada pela IBM. Por estratégias de marketing, espertamente a Microsoft incluiu o nome dessa linguagem dentro do seu SGBD.

O SQL Server funciona apenas sob as várias versões do Sistema Operacional Windows, da própria Microsoft. Os seus grandes concorrentes: Oracle e Postgres, por sua vez são mais flexíveis, pois funcionam em diversas plataformas e sistemas operacionais diferentes.

Oracle

O Oracle é um SGBD que surgiu no final dos anos 70, quando Larry Ellison vislumbrou uma oportunidade que outras companhias ainda não haviam percebido. Esse grande

empreendedor encontrou um protótipo funcional de um Banco de Dados relacional e descobriu que nenhuma empresa tinha se empenhado em comercializar essa tecnologia.

Ellison e os cofundadores da Oracle Corporation, Bob Miner e Ed Oates, perceberam que havia um tremendo potencial de negócios no modelo de Banco de Dados Relacional tornando assim a maior empresa de software empresarial do mundo. O SGBD da Oracle se tornou líder de mercado. O Oracle 9i foi pioneiro no suporte ao modelo Web. O Oracle 10g, mais recente, se baseia na tecnologia de grid.

DB2

O DB2 é o Sistema Gerenciador de Banco de Dados Relacionais (SGDBR) produzido pela IBM. Existem diferentes versões do DB2 que rodam desde num simples computador de mão, até em potentes mainframes. Funcionam em servidores baseados em sistemas Unix, Windows, ou Linux.

O projeto DB2 começou no inicio dos anos 70 quando Edgar Frank Codd, trabalhando para IBM, descreveu e publicou a teoria dos Bancos de Dados Relacionais. Para aplicar o modelo, Codd criou uma linguagem que chamou de Alpha. Entretanto, a IBM não acreditava no potencial das suas ideias, deixando-o fora da supervisão do grupo de programadores, que violaram diversas ideias fundamentais do modelo relacional de Codd. O resultado foi a linguagem SEQUEL, que depois foi mudado para seu acrônimo SQL porque SEQUEL já era uma marca registrada.

Por muitos anos, DB2 foi feito exclusivamente para rodar nos mainframes da IBM. Posteriormente a IBM introduziu o DB2 para outras plataformas de servidores, incluindo o Unix e o Windows, para então colocar no Linux e PDAs.



Estudo Complementar

Veja um comparativo entre o MySQL e o PostgreSQL

<http://articles.techrepublic.com.com/5100-22-1050671.html>

<http://www.devx.com/dbzone/Article/29480>



Atividades

Analise o SGBD usado pelo Metrô de São Paulo e o motivo de fazerem esta opção:

<https://extranet.metrosp.com.br/downloads/outros/usopostgres.pdf>



Atividades

Para concluir seus estudos é fundamental que você entre no site da ESAB e, em sua Sala de Aula, faça a Atividade 3, no link “Atividades”.





Atividades

Atividade Dissertativa

Desenvolva uma pesquisa gerando um texto, de 2 a 3 folhas adicionando imagens, de uma das unidades da nossa apostila, de sua livre escolha, permitindo a expansão da temática selecionada.

Atenção: Qualquer bloco de texto igual ou existente na internet será devolvido para o aluno realize novamente.



GLOSSÁRIO

Como dito anteriormente, utilizamos o glossário virtual <http://pt.wikipedia.org>.

BIBLIOGRAFIA

DATE, C.J. **Introdução a Sistemas de Bancos de Dados** (tradução da 8a ed.). Rio de Janeiro: Elsevier, 2003.

KORTH, H.F.; Silberschatz, A. **Sistema de Banco de Dados**. 3a ed. São Paulo: Makron Books, 1999.

FANDERUFF, Damaris. **Dominando o Oracle 9i: Modelagem e Desenvolvimento**. São Paulo: Pearson Education do Brasil, 2003.

SILVA, Luciano Carlos da. **Banco de Dados para WEB: do Planejamento à Implementação**. São Paulo: Érica, 2001.