Universidade Federal do Piauí
Centro de Educação Aberta e a Distância

LABORATÓRIO DE BANCO DE DADOS

Flávio Ferry de Oliveira Moreira

Ministério da Educação - MEC
Universidade Aberta do Brasil - UAB
Universidade Federal do Piauí - UFPI
Universidade Aberta do Piauí - UAPI
Centro de Educação Aberta e a Distância - CEAD

LABORATÓRIO DE BANCO DE DADOS

Flávio Ferry de Oliveira Moreira

UFPI
2013

PRESIDENTE DA REPÚBLICA	*Dilma Vana Rousseff Linhares*
MINISTÉRIO DA EDUCAÇÃO	*José Henrique Pain*
GOVERNADOR DO ESTADO	*Wilson Nunes Martins*
REITOR DA UNIVERSAIDADE FEDERAL DO PIAUÍ	*Luiz de Sousa Santos Júnior*
PRESIDENTE DA CAPES	*Jorge Almeida Guimarães*
COORDENADOR GERAL DA UNIVERSIDADE ABERTA DO BRASIL	*João Carlos Teatini de S. Clímaco*
DIRETOR DO CENTRO DE EDUCAÇÃO ABERTA E A DISTÂNCIA DA UFPI	*Gildásio Guedes Fernandes*

COORDENADORES DE CURSOS

ADMINISTRAÇÃO	*Francisco Pereira da Silva Filho*
CIÊNCIAS BIOLÓGICAS	*Maria da Conceição Prado de Oliveira*
FILOSOFIA	*Zoraida Maria Lopes Feitosa*
FÍSICA	*Miguel Arcanjo Costa*
MATEMÁTICA	*João Benício de Melo Neto*
PEDAGOGIA	*Vera Lúcia Costa Oliveira*
QUÍMICA	*Davi da Silva*
SISTEMAS DE INFORMAÇÃO	*Arlino Henrique Magalhães de Araújo*

EQUIPE DE DESENVOLVIMENTO

TÉCNICO EM ASSUNTOS EDUCACIONAIS	*Ubirajara Santana Assunção*
EDIÇÃO	*Roberto Denes Quaresma Rêgo*
PROJETO GRÁFICO	*Samuel Falcão Silva*
DIAGRAMAÇÃO	*Cleonildo F. de Mendonça Neto*
REVISÃO	*Beatriz Gama Rodrigues*
REVISÃO GRÁFICA	*Carmem Lúcia Portela Santos*

CONSELHO EDITORIAL DA EDUFPI

Prof. Dr. Ricardo Alaggio Ribeiro (Presidente)
Des. Tomaz Gomes Campelo
Prof. Dr. José Renato de Araújo Sousa
Profª. Drª. Teresinha de Jesus Mesquita Queiroz
Profª. Francisca Maria Soares Mendes
Profª. Iracildes Maria de Moura Fé Lima
Prof. Dr. João Renór Ferreira de Carvalho

M838l Moreira, Flávio Ferry de Oliveira
Laboratório de Banco de Dados./ Flávio Ferry de Oliveiera Moreira. Teresina: EDUFPI, 2013.
80 p.

1- Banco de Dados. 2 - Sistema de Informação. 3 - Educação a Distâcia. I. Título

C.D.D. - 371.3

© 2013 Universidade Federal do Piauí - UFPI. Todos os direitos reservados.

A responsabilidade pelo conteúdo e imagens desta obra é do autor. O conteúdo desta obra foi licenciado temporária e gratuitamente para utilização no âmbito do Sistema Universidade Aberta do Brasil, através da UFPI. O leitor se compromete a utilizar o conteúdo desta obra para aprendizado pessoal, sendo que a reprodução e distribuição ficarão limitadas ao âmbito interno dos cursos. A citação desta obra em trabalhos acadêmicos e/ou profissionais poderá ser feita com indicação da fonte. A cópia deste obra sem autorização expressa ou com intuito de lucro constitui crime contra a propriedade intelectual, com sansões previstas no Código Penal.
É proibida a venda ou distribuição deste material.

Apresentação

Este material é destinado àqueles que chegaram ao sexto módulo do curso de Sistemas de Informação pela Universidade Aberta do Piauí (UAPI) vinculada ao consórcio formado pela Universidade Federal do Piauí (UFPI), Universidade Estadual do Piauí (UESPI) e CInstituto Federal de Educação, Ciência e Tecnologia do Piauí (IFPI), com apoio do Governo do Estado do Piauí, através da Secretaria Estadual de Educação e Cultura (SEEDUC - PI). Nele mostraremos o desenvolvimento de um projeto completo de Banco de Dados para uma aplicação real.

Esta disciplina apresenta-se como a prática de fixação da disciplina de Fundamentos de Bancos de Dados e Banco de Dados, sendo esta última com menor ênfase.

O texto está dividido em três unidades, a saber:

Unidade 1 – Proposta de um Projeto-Modelo de Banco de Dados a ser desenvolvido.

Unidade 2 – Execução de um projeto de Bancos de Dados para a aplicação modelo e para o projeto individual do aluno

Unidade 3 – Implementação em MySQL dos projetos expostos no material e projetos individuais dos alunos .

Sumário

UNIDADE 01

Definição de Projetos

Resumo

Esta unidade trata da apresentação da metodologia que será aplicada ao uso prático deste material. Nesta unidade serão definidas as regras de desenvolvimento dos projetos individuais dos alunos que usem este material como referência. A ideia consiste em desenvolver um projeto de Banco de Dados e executá-lo ao longo dos demais capítulos. Em paralelo recomenda-se que cada aluno faça um projeto somente seu e a partir de um caso real.

1 DEFINIÇÃO DE PROJETOS

Definição do Projeto Piloto

Ao longo deste texto, será desenvolvido o projeto que é descrito a seguir:

Uma oficina de consertos em aparelhos eletrônicos quer desenvolver um sistema de informação para automatizar sua rotina. Os clientes trazem aparelhos defeituosos, que são recebidos pelo funcionário da recepção, que abre uma ordem de serviço (doravante chamada OS). Essa OS fica sob a responsabilidade de um técnico que terá vinte quatro horas para fornecer o orçamento do reparo. Uma vez pronto o orçamento, o cliente é contatado para aprovação ou não do conserto. Os orçamentos aprovados devem ser executados pelo técnico responsável pelo mesmo. Uma vez executado o conserto, o cliente é avisado e tem o prazo de quarenta dias para a retirada do aparelho com o devido pagamento do serviço. Entre quarenta e sessenta dias a oficina tem o direito de cobrar taxa de armazenagem do aparelho. Após 60 dias o aparelho é tido como abandonado pelo dono, podendo ser revendido a terceiros como forma de pagamento pelo conserto executado. A seguir , tem-se a listagem das principais entidades deste caso e seus respectivos atributos:

Entidade	Atributos
Clientes	CPF*, nome, endereço, fone, celular, fonetrab
Técnicos	Matrícula*, nome, celular

Entidade	Atributos
Clientes	CPF*, nome, endereço, fone, celular, fonetrab
Técnicos	Matrícula*, nome, celular
OS	Número*, CodCliente^{CE}, MatrTec^{CE}, dataAbertura, Orcamento, Status, CodAparelho^{CE}, defeitoReclamado, dataConserto, dataEntrega
Aparelhos	Código*, Descrição, Marca, Garantia, PrazoVencido

Talvez outras tabelas ou outros atributos sejam necessários para uma implementação mais ligada à rotina de uma oficina real.

Estratégia

No decorrer de todo este material, este sistema de oficina irá evoluir em todas as etapas de desenvolvimento ligadas a Banco de Dados. Em primeiro lugar é importante termos um esboço do Diagrama Entidade-Relacionamentos deste caso.

Nesta etapa, para uma melhor visão do sistema, recomenda-se o desenho manual do D.E-R deste caso, para mais tarde ser transferido para uma ferramenta CASE como o MySQL Workbench OSS. E posterior implementação no MySQL.

Softwares de Apoio

MySQL Workbench OSS

Agora é recomendável que todos tenham instalado o MySQL Workbench OSS em seus computadores de trabalho, pois na próxima etapa será necessário fazer uso desta ferramenta neste projeto. A figura 1.1, a seguir, ilustra a instalação do MySQL Workbench OSS.

Cod*	Nome	Fone
01	Flávio	9966-6363
02	Aline	9494-4477
120	Ana	8855-2121

Figura 1.1 – Instalação do MySQL Workbench OSS

MySQL (SGBD)

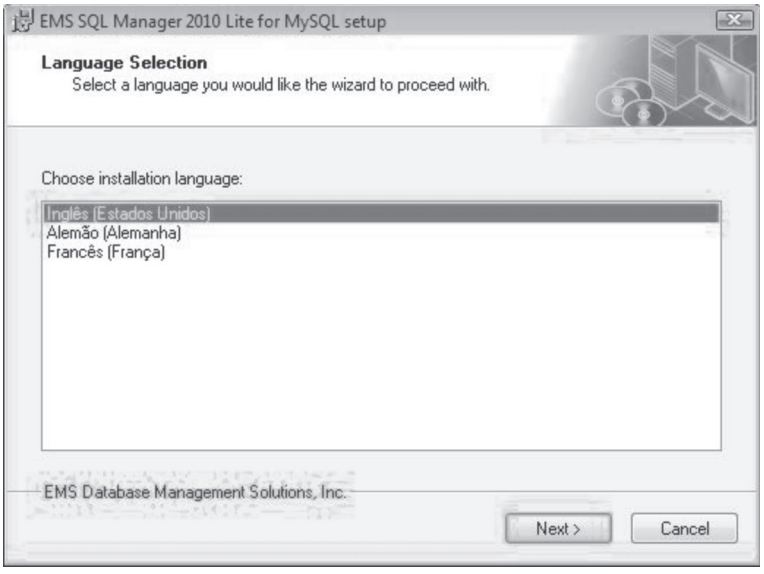
Após o projeto ficar pronto, será executado de forma que se possa dispor do Banco de Dados para esta aplicação em MySQL, sendo assim, é necessário também fazer a instalação deste SGBD caso ainda não esteja instalado em seu equipamento. Segue a figura 1.2 ilustrando a instalação do

Figura 1.2 – Instalação do SGBD MySQL

MySQL em seu computador.

SQL Manager 2010 Lite

Lembrando dos conceitos fundamentais de segurança em Banco de Dados, ao instalar o MySQL é fundamental trocar a senha do usuário root, pois este detém os poderes máximos dentro do SGBD e sua senha padrão é bastante conhecida por todos. Recomendamos, para tanto, usar uma das interfaces gráficas disponíveis para MySQL como: SQL Manager 2010 Lite ou



DB Tools ou ainda o SQL Yog.

Figura 1.3 – Instalação do SQL Manager 2010 Lite

A ferramenta gráfica a ser adotada neste material é o SQL Manager 2010 Lite, cuja tela de instalação pode ser vista na figura 1.3 logo acima. Executando o SQL Manager 2010 Lite, é apresentada a tela que pode ser vista na figura 1.4 a seguir. Onde pode-se fazer algumas configurações ou apenas aceitar o padrão sugerido clicando em OK.

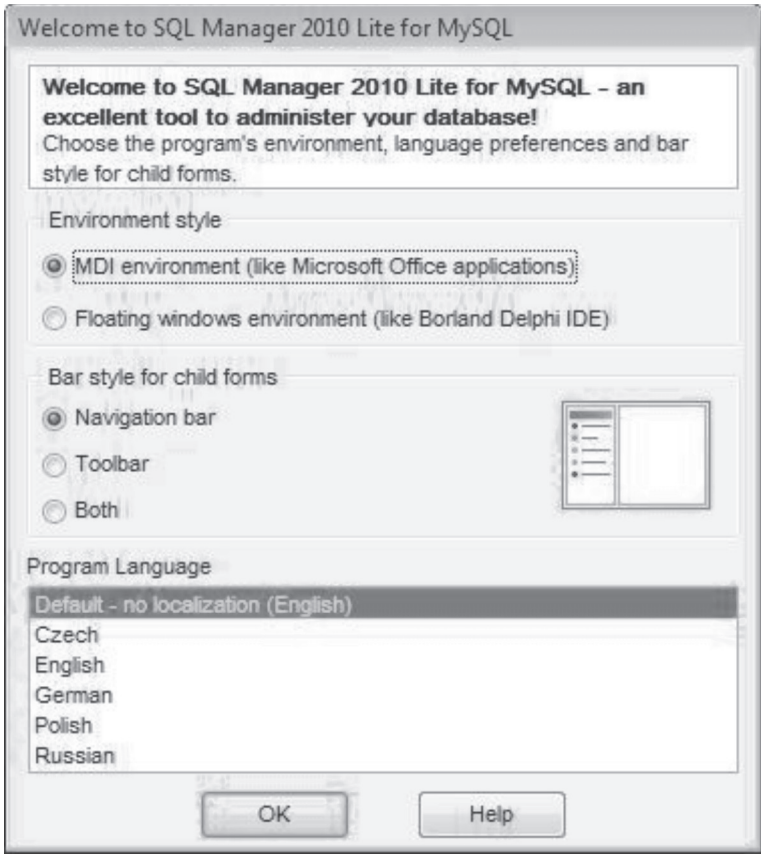


Figura 1.4 – Configurando o SQL Manager 2010 Lite

Após concluir a configuração, apresenta-se a tela a seguir. Conforme ilustra a figura 1.5, a dica do dia é apresentada, fica a seu critério mantê-la ativada ou não.

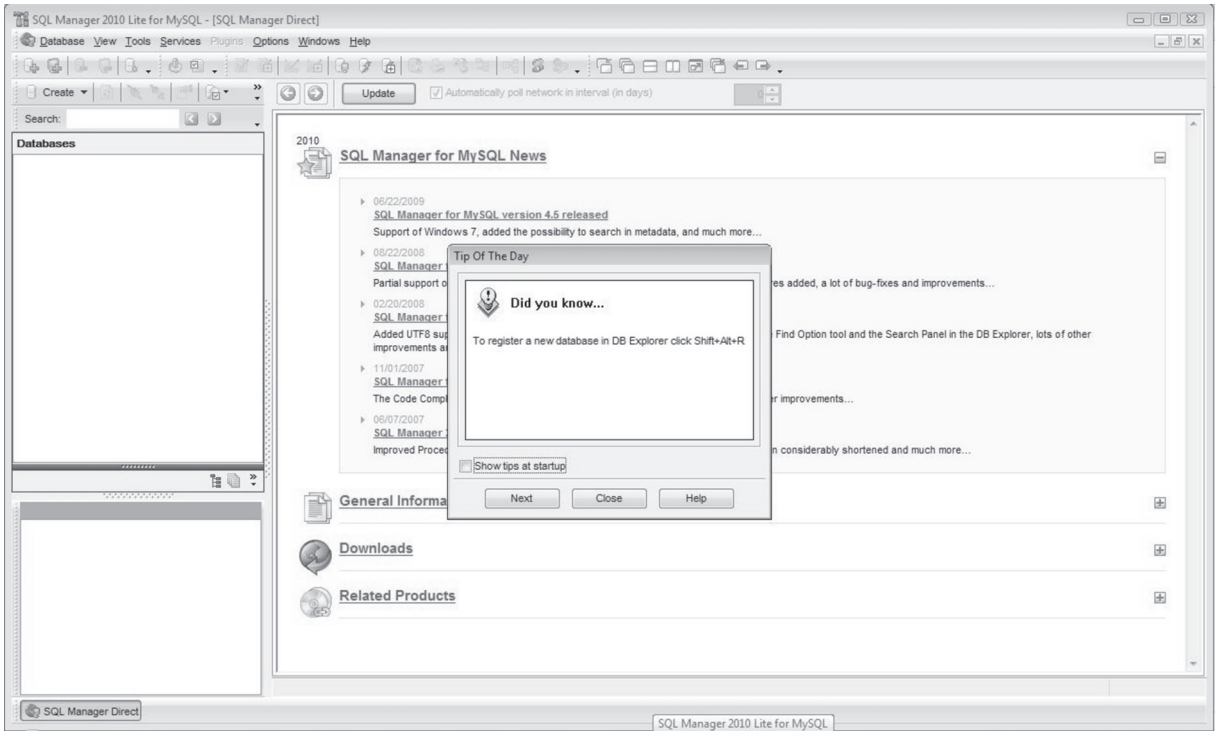


Figura 1.5 – Tela inicial do SQL Manager 2010 Lite

Agora ao procedimento de uso da ferramenta:

1. Clique no primeiro botão da barra de ferramentas, que simboliza um banco de dados amarelo com um sinal de + verde – esta ferramenta irá conectar o SQL Manager ao servidor MySQL instalado em sua máquina;
2. Use localhost para o host name, porta 3306, usuário root e a senha do root que está em uso atualmente (definida na instalação do MySQL);
3. Clique em next;
4. Escolha um database name (ou crie um novo database) e clique em Finish.

Uma boa ideia talvez fosse criar um database com o nome da disciplina ou uma sigla como LBD (Laboratório de Banco de Dados).

Para mudar a senha do root se for o caso, clique no menu TOOLS e na opção USER MANAGER, informe a seguir o login do usuário root e sua senha atual. Veja a figura 1.6 a seguir:

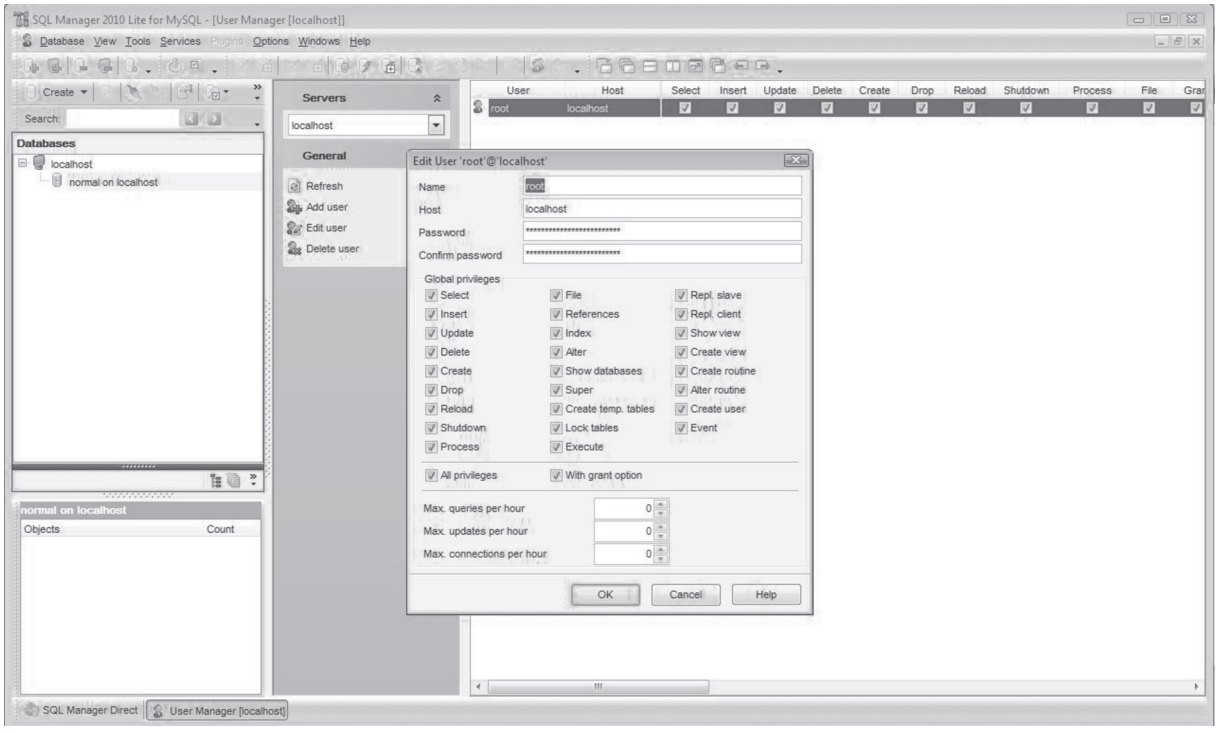


Figura 1.6 – alterando usuários

Em servers escolha localhost, clique no usuário pretendido: root, finalmente clique em EDIT USER, podendo alterar a senha do usuário que passará a valer imediatamente.

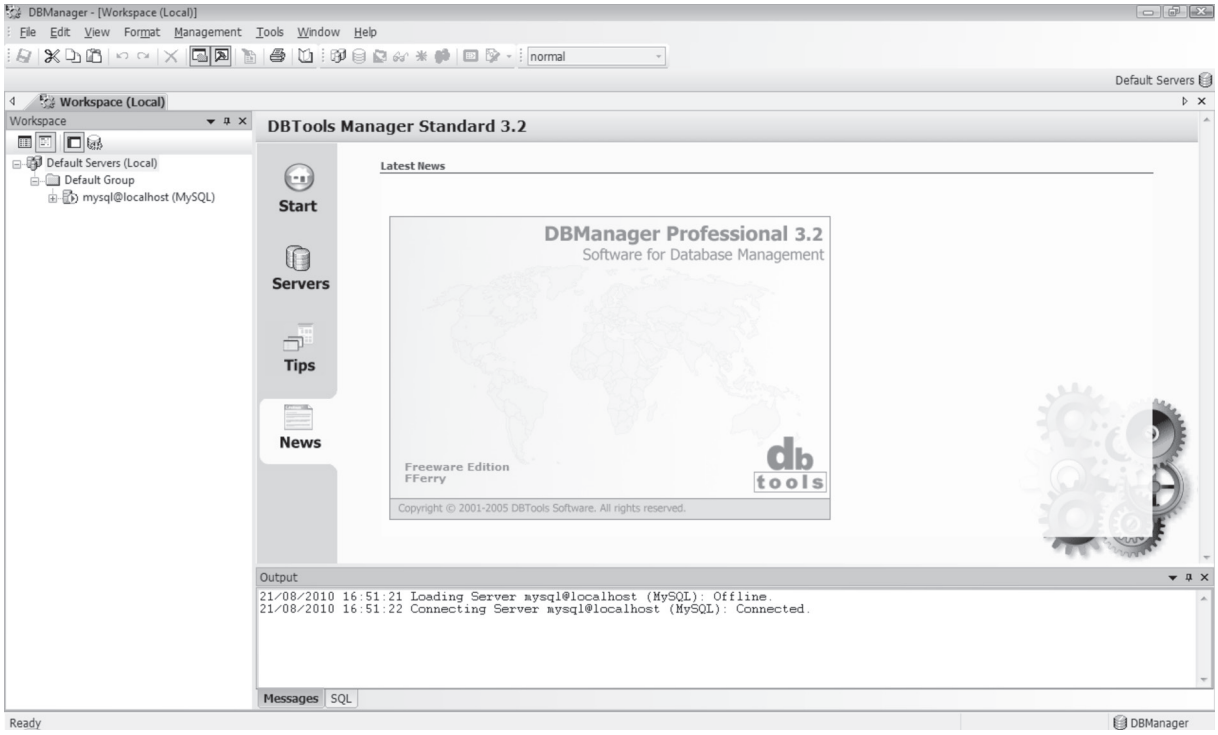
Nesta tela, também é possível criar novos usuários para seu MySQL, dando a estes as permissões necessárias. Ao criar um servidor “valendo mesmo”, recomenda-se a criação de um usuário menos poderoso que o root para a aplicação acessar o SGBD através deste.

Atenção!
NUNCA modifique os privilégios do usuário root. Este sempre deverá ter TODOS os privilégios disponíveis.

Mude a senha do root periodicamente para dificultar invasões em sua máquina através de redes, inclusive pela Internet.

DB *Tools Manager Standard Edition*

O SQL Manager 2010 Lite é muito bom para fazer o papel de DBA do SGBD no que diz respeito a gerenciamento de usuários e execuções de scripts previamente preparados de SQL. Mas, para práticas de SQL diretamente no Banco de Dados através da digitação de predicados com sua consequente visualização de resultados, recomenda-se o uso do DB Tools Manager Standard Edition devido a facilidade de uso de sua interface para



este fim. A figura a seguir ilustra a tela inicial do programa.

Figura 1.7 – Tela inicial do DB Tools Manager Standard Edition

Antes de chegar a esta tela, é solicitada uma chave de serial; para ativar versão full do programa, pode-se usar a opção “no thanks” para prosseguir na versão standard. Após o programa solicita o SGBD a ser trabalhado, escolha MySQL. Finalmente a tela acima aparece.

Clicando na estrutura de WORKSPACE ao lado esquerdo da tela, abre-se os Bancos de Dados do SGBD de localhost (seu computador). Veja a estrutura expandida na figura 1.8 a seguir:

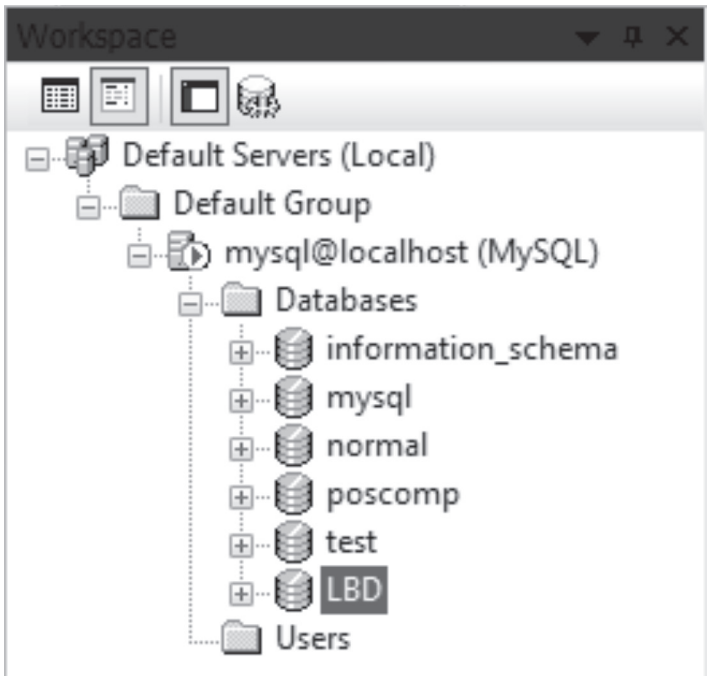


Figura 1.8 – Estrutura do SGBD MySQL

Lembre-se que esta estrutura vai variar de acordo com os DATABASES criados em sua máquina. Trabalharei com um database chamado “LBD” para esta apostila. Para tanto, pode-se clicar com o botão direito sobre a pasta “databases” e escolher “new database”, fornecendo o nome LBD. A seguir, selecione LBD, clique com o botão direito e escolha “new query”. Abre-se a tela que pode ser vista na figura 1.9 a seguir:

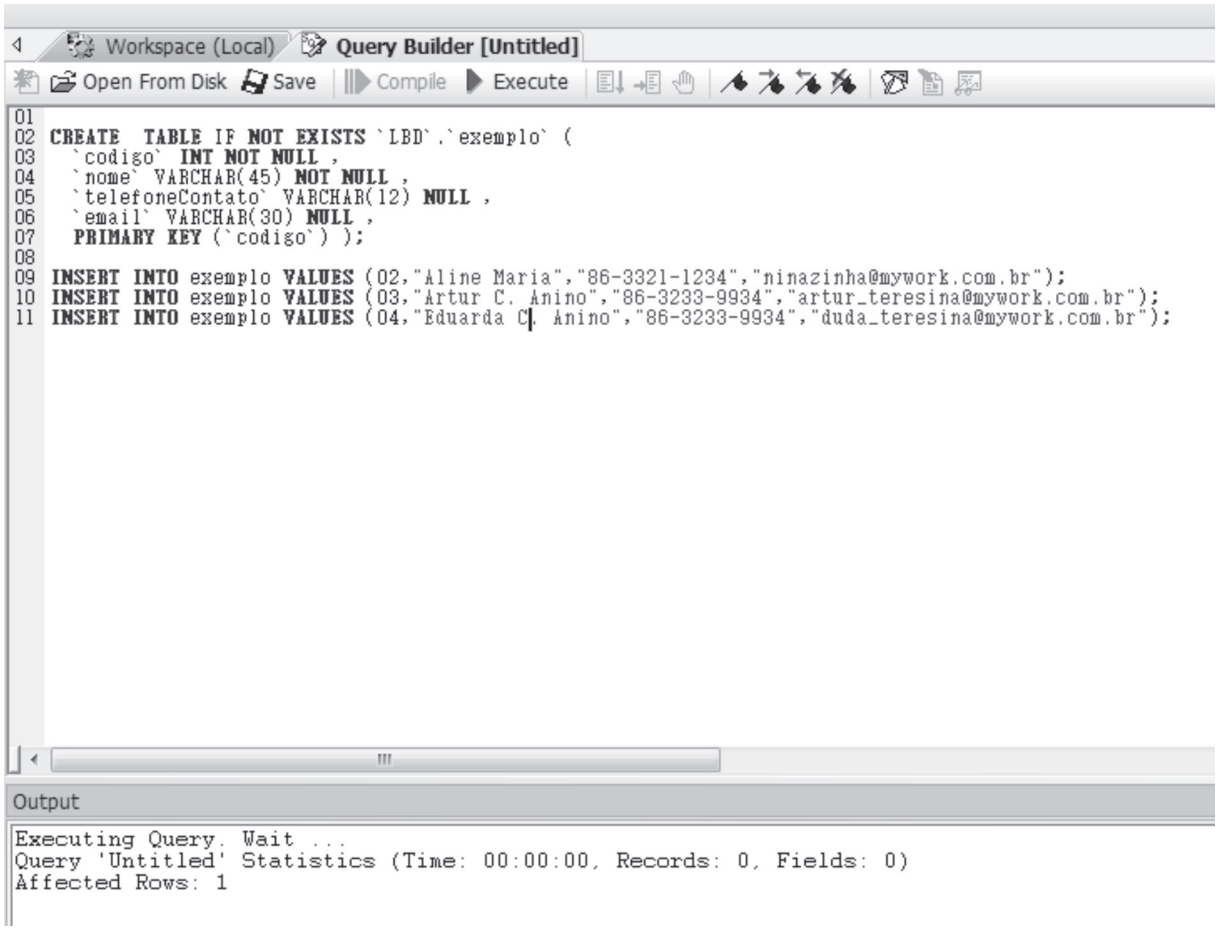


Figura 1.9 – Tela de Query do DB Manager Standard

O script usado na figura acima tem o seguinte conteúdo:

```
CREATE  TABLE IF NOT EXISTS `LBD`.`exemplo` (
  `codigo` INT NOT NULL ,
  `nome` VARCHAR(45) NOT NULL ,
  `telefoneContato` VARCHAR(12) NULL ,
  `email` VARCHAR(30) NULL ,
  PRIMARY KEY (`codigo`) );

INSERT INTO exemplo VALUES (01,”Flávio José”,”86-3322-2233”,”fj_man@mywork.com.br”);
INSERT INTO exemplo VALUES (02,”Aline Maria”,”86-
```

```
3321-1234",”ninazinha@mywork.com.br”);
      INSERT INTO exemplo VALUES (03,”Artur C. Anino”,”86-
3233-9934”,”artur_teresina@mywork.com.br”);
      INSERT INTO exemplo VALUES (04,”Eduarda C. Anino”,”86-
3233-9934”,”duda_teresina@mywork.com.br”);
```

Executado com o auxílio da tecla F5.

Fazendo a execução do comando SELECT * FROM exemplo, obtém-se o resultado visto na tabela 1.1 a seguir:

Tabela 1.1 – resultado da execução do select na tabela de exemplo

codigo	Nome	telefoneContato	Email
1	Flávio José	86-3322-2233	fj_man@mywork.com.br
2	Aline Maria	86-3321-1234	ninazinha@mywork.com.br
3	Artur C.Anino	86-3233-9934	artur_teresina@mywork.com.br
4	Eduarda C.Anino	86-3233-9934	duda_teresina@mywork.com.br

Agora que estamos com todas as ferramentas preparadas, vamos definir o trabalho final da disciplina.

Projetos Individuais

Cada aluno desta disciplina deverá apresentar um projeto individual de Banco de Dados como uma das avaliações. Para tanto, lembramos do trabalho final de Fundamentos de Banco de Dados. Será agora um projeto nos mesmos moldes.

Especificação

Procurar uma empresa de algum conhecido que queira colaborar com seu trabalho fazendo uma entrevista para um suposto Sistema de Informação.

Lembre-se de deixar claro para esta pessoa que o Sistema é trabalho final de uma disciplina da sua faculdade, e nosso interesse neste será apenas na parte de Banco de Dados.

Fazer a implementação total deste sistema, dependerá de um acerto seu com o empresário, e a parte de desenvolvimento do software e sua interface não são conteúdos desta disciplina, portanto não nos interessam.

Vamos focar apenas no Banco de Dados da aplicação.

O seu trabalho final, que será avaliado de zero a dez juntamente com a resolução dos exercícios do final da unidade 2 deste material, deverá ter os seguintes itens:

- Texto com descrição do estudo de caso;
- Diagrama de Entidades-Relacionamentos feito no MySQL Workbench OSS;
- Script para execução no MySQL para criação das estruturas do Diagrama E-R, bem como povoamento das tabelas e algumas consultas.

A entrega deste deverá acontecer até o dia de aplicação da prova da disciplina, via e-mail, para o seu tutor.

Na próxima unidade trataremos do Projeto de nossa Apostila. Sendo os conceitos vistos a serem aplicados também em seu projeto final.

REFERÊNCIAS NA WEB

- MySQL - http://www.mysql.com/;
- Downloads do MySQL - http://dev.mysql.com/downloads/
- MySQL Workbench OSS - http://dev.mysql.com/downloads/workbench/
- SQL Manager 2010 Lite - http://sqlmanager.net/products/mysql/manager/download
- DB Tools Manager – Standard Edition - http://www.dbtools.com.br/EN/downloads/

UNIDADE 02

Modelagem e-r dos Projetos

Resumo

Esta unidade trata da execução da modelagem dos projetos, do livro e do aluno, de acordo com as regras do modelo Entidade-Relacionamentos. Ao final deste capítulo existe um exercício proposto para relembrar os conceitos vistos em "Fundamentos de Banco de Dados", que são de vital importância para o sucesso nesta unidade.

2 MODELAGEM E-R DOS PROJETOS

Nesta unidade, faz-se o uso intensivo da ferramenta MySQL Workbench OSS para modelagens de Diagramas Entidades-Relacionamentos usando diretamente o computador.

Figura 2.0 – Tela de boas-vindas da ferramenta

Executando o MySQL Workbench

Procure no grupo MySQL, do Menu Iniciar, o atalho para MySQL Workbench OSS. Clique nele e aparecerá a tela ilustrada na figura 2.1 a seguir:

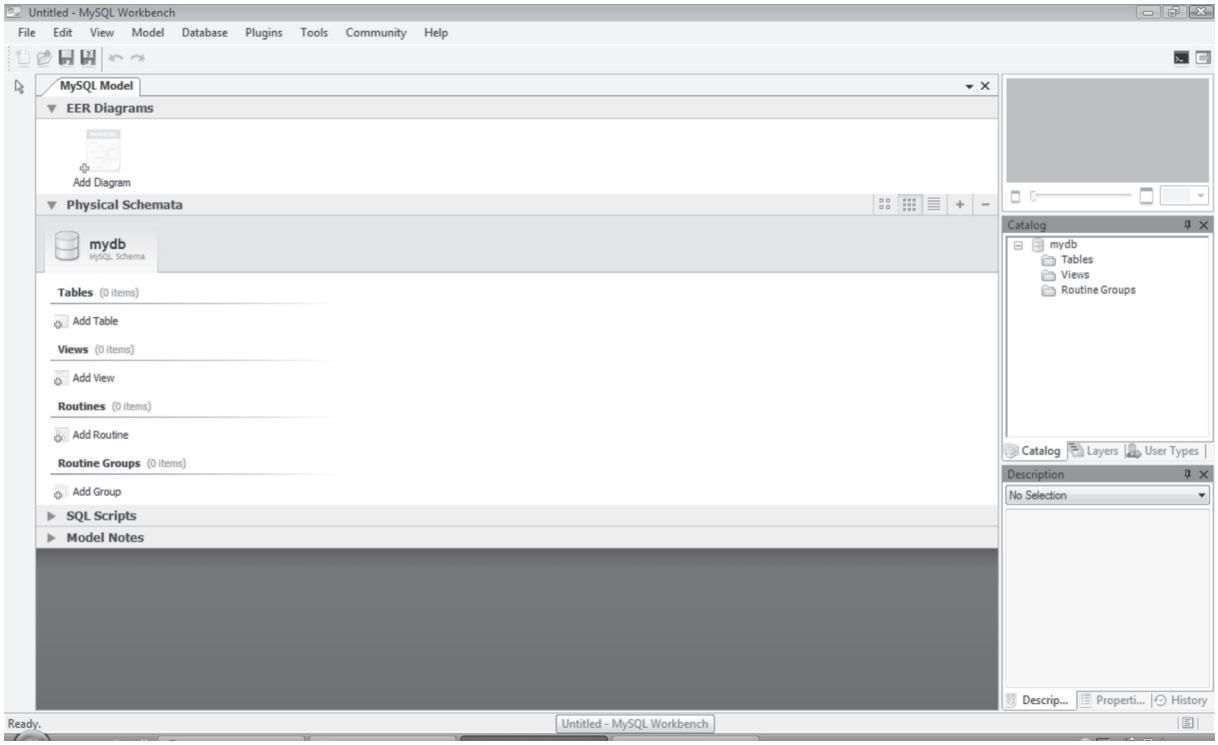


Figura 2.1 – Tela inicial do MySQL Workbench OSS

A partir desta tela faremos o diagrama de nosso projeto piloto. E também espera-se que cada um dos alunos faça a modelagem de seu projeto final em paralelo para praticar duplamente.

Criando o Diagrama E-R na Ferramenta

Neste momento, deve-se clicar duas vezes em “Add Diagram” que é um ícone que aparece no canto superior esquerdo da tela. Uma nova aba será aberta com o aspecto da figura 2.2 a seguir:

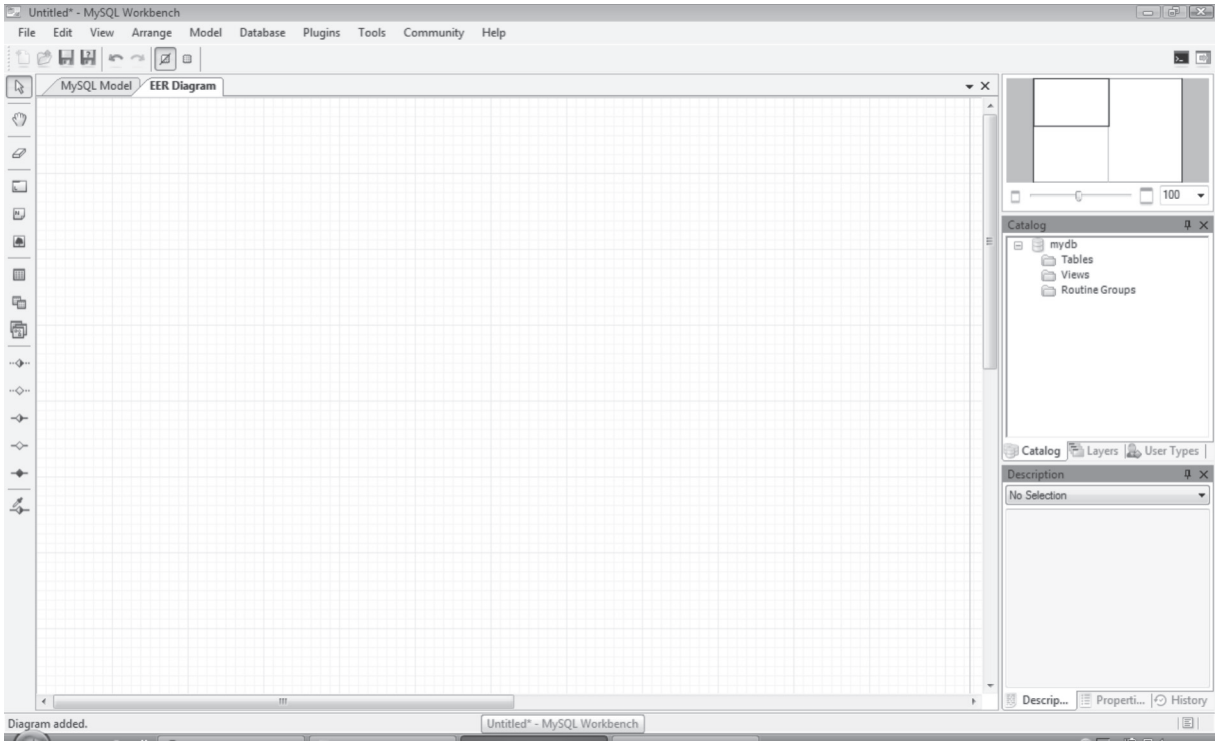


Figura 2.2 – Tela de criação do Diagrama E-R

Agora vamos chamar atenção para as ferramentas desta tela, presentes em uma barra localizada à esquerda. Os detalhamentos de cada botão estão na tabela 2.1 a seguir:

Tabela 2.1 – Detalhamento das Ferramentas E-R

Figura	Descrição
	Seleciona objetos – Serve para clicar em algum objeto do diagrama para reposicioná-lo.
	Move o diagrama – Faz de conta que a tela é uma folha de papel e move a área de trabalho como se o fosse.
	Deleta um objeto selecionado.

	Ferramentas de Desenho – Similares às encontradas em programas como photoshop, servem para colocar novas camadas de imagem, texto e figura respectivamente.
	Ferramentas de Armazenamento – Representam tabelas (entidades), visões e rotinas.
	Ferramentas de Relacionamento – Fazem relacionamentos não-identificadores (pontilhados) e identificadores com as mais diversas cardinalidades. Este texto fará mais detalhes a respeito destas ferramentas logo a seguir.
	Ferramenta Engenharia Reversa – Cria relacionamentos a partir de colunas já existentes nas tabelas.

Entendendo os diferentes relacionamentos

A ferramenta apresenta diferentes relacionamentos agrupados em duas grandes categorias: não-identificadores e identificadores.

Via de regra em um relacionamento tipo “um-para-muitos”, a chave primária da entidade do lado “um” é exportada para a entidade do lado “muitos” como chave estrangeira.

Os relacionamentos não-identificadores são aqueles onde a chave exportada aparece como chave estrangeira na outra tabela mas não fazendo parte da chave primária.

Nos relacionamentos identificadores, esta chave estrangeira, que chega, fará parte da chave primária.

O fato de o relacionamento não-identificador ser representado na ferramenta de forma tracejada não significa que este seja um relacionamento

de entidades fracas, conforme visto na teoria de Fundamentos de Bancos de Dados.

Um caso típico de relacionamentos identificadores acontece quando existe no D. E-R um relacionamento do tipo “muitos-para-muitos”.

Nestes casos, a regra chamada “Terceira Forma Normal – 3FN”, pertencente à Teoria das Dependências – Normalização, diz claramente que nestes casos deve-se criar uma tabela extra chamada “Entidade Associativa”, onde sua chave primária será composta e formada por duas chaves estrangeiras vindas de cada uma das tabelas originalmente participantes do relacionamento “muitos-para-muitos”. Ou seja, um relacionamento “muitos-para-muitos” é sempre substituído por uma nova tabela e dois relacionamentos “um-para-muitos” entre esta nova e as duas originais, sendo que o lado “muitos” sempre apontará para a nova tabela criada. A ferramenta em estudo agora faz a aplicação desta regra de maneira automática, sempre que for feita uma representação do tipo “identificador e m:n” é criada uma nova tabela e aplicada toda a regra descrita acima. Isto pode ser ilustrado na figura 2.3 a seguir:

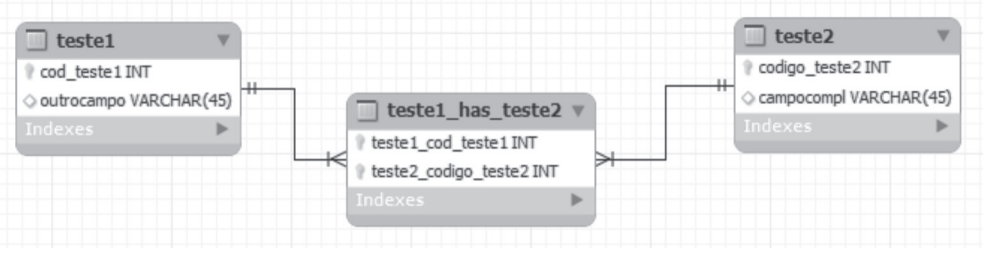


Figura 2.3 – MySQL Workbench OSS tratando relacionamentos m:n

Criando o Diagrama E-R do Projeto Piloto

Agora faremos o Diagrama Entidades-Relacionamentos do Projeto Piloto definido anteriormente na seção 1.2. Recomenda-se uma leitura e até algumas releituras do texto que descreve o caso. As entidades citadas devem estar obrigatoriamente no diagrama, o texto deve ser cuidadosamente lido para que não restem dúvidas sobre alguma entidade ainda a ser representada.

Procedimento para criação do D. E-R do Projeto Piloto

Seguem os passos para a criação do Diagrama Entidades-Relacionamentos de nosso Projeto Piloto:

1. Execute o MySQL Workbenck OSS e comece um novo diagrama;
2. Procure representar primeiro a entidade considerada mais importante para o caso – neste, consideremos a unidade “OS”;
3. Faça as demais entidades e ligue os relacionamentos sempre usando os “não-identificadores”;
4. Compare com as figuras mostradas a seguir.

Segue a figura 2.4 que ilustra a criação inicial da entidade OS, lembrando que as chaves estrangeiras não são criadas a principio, pois estas vêm de outra entidade automaticamente.

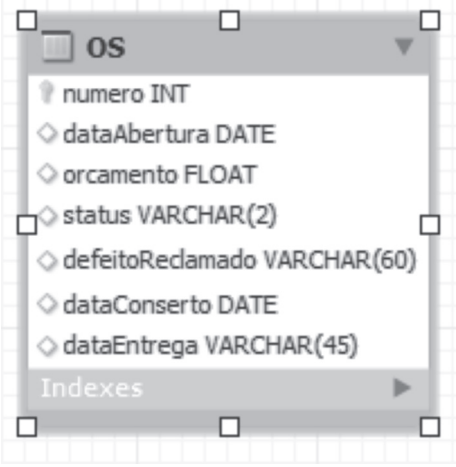


Figura 2.4 – Criação da entidade OS

Veja que o atributo status pode ter vários valores, porém estamos usando uma string de tamanho 2 para guardar apenas um código do status, que poderá ser deixado para definições a critério do implementador do sistema.

A seguir, a figura 2.5 ilustra o diagrama final de nosso projeto:

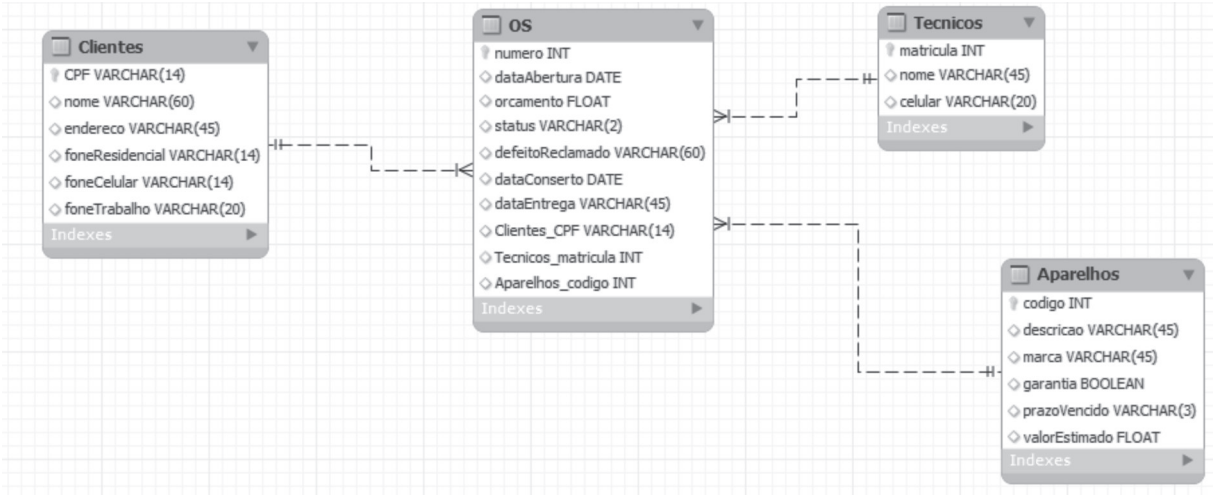


Figura 2.5 – Diagrama Entidades-Relacionamentos do Projeto

Veja que na entidade aparelhos foi necessário criarmos um novo atributo a partir da leitura do texto: valorEstimado, pois é dito que o aparelho abandonado pelo proprietário, após 60 dias, pode ser vendido para pagar o conserto e a armazenagem na oficina.

Não esqueça de salvar o modelo antes de fechar a ferramenta, pois na próxima unidade faremos uso deste diagrama já salvo em seu computador.

Cuide de seu projeto individual

Neste momento, fica a preocupação com os trabalhos finais dos alunos. Usando esta mesma ferramenta e partindo dos levantamentos feitos por cada um anteriormente, deve-se neste momento aproveitar a ferramenta e fazer os Diagramas de Entidades-Relacionamentos de cada um dos casos. Agindo assim os vossos projetos ficarão prontos juntamente com o Projeto Piloto desta apostila.

Um exercício importante agora será fazer o Diagrama E-R de seu caso individual nesta ferramenta.

EXERCÍCIOS

1. Fale da importância do SGBD para um Sistema de Informação.
2. Como proceder para garantir a segurança de um servidor de Banco de Dados por menor que ele seja?
3. Que importância tem o Diagrama Entidades-Relacionamentos para o projeto de uma aplicação?
4. No D. E-R, quais são as notações mais usadas?
5. Mostre as diferenças de notações existentes entre os modelos de Peter Chen e James Martin.
6. Conceitue Chave.
7. Como se resolve um caso de relacionamento “muitos-para-muitos”?
8. Por que é importante usar um SGBD gratuito e de confiança como o MySQL?
9. Em alguns casos, usa-se o relacionamento “um-para-um”. Seria possível, nestes casos, simplesmente aumentar o número de atributos de uma das entidades e abandonar a outra? Justifique sua resposta.
10. Na instalação de um SGBD é importante mudar a senha do root? Justifique sua resposta.

REFERÊNCIAS NA WEB

Texto sobre M E-R:
http://www.shammas.eng.br/acad/materiais/mer.pdf
Aula sobre E-R:
http://www.las.pucpr.br/mcfmello/BD/BD-Aula02-MER.pdf
M E-R:
http://www.scribd.com/doc/512604/Modelo-EntidadeRelacionamento
Outra aula de E-R:
http://www.ic.unicamp.br/~geovane/mo410-091/Ch02-MER_pt.pdf
Documentação do MySQL Workbench:
http://downloads.mysql.com/docs/workbench-en.pdf
Tutorial do MySQL Workbench:
http://dev.mysql.com/doc/workbench/en/wb-tutorials.html

UNIDADE 03

Implementação dos Projetos

Resumo

Nesta última unidade, os projetos serão realmente executados em um Sistema de Gerenciamento de Banco de Dados (SGBD). São explicados os passos básicos de instalação e configuração de um SGBD. Ao final deste capítulo, o SGBD do aluno deverá executar todos os comandos de SQL necessários para uso do projeto.

3 IMPLEMENTAÇÃO DOS PROJETOS

Esta unidade se destina a execução dos scripts de Banco de Dados gerados pelo MySQL Workbench OSS e uso da Linguagem SQL para gerar consultas e relatórios.

Lembrando que a definição de layouts de relatórios está fora do escopo desta disciplina por se tratar de uma técnica de desenvolvimento de programas.

Revisão de Linguagem SQL

Nesta seção, vamos revisar as principais cláusulas de SQL, para que sejamos mais felizes na implementação de nossos projetos de Banco de Dados.

CONSULTAS E RELATÓRIOS

a) Comando SELECT → Implementa algumas operações da álgebra relacional tais como: Seleção, Projeção e Produto Cartesiano (só para citar as mais usuais).

Sintaxe para Seleção:

```
SELECT * FROM nome_tabela
  WHERE
      <condição*>
```

*Onde a condição pode ser formada pela comparação de um atributo da tabela com um dado valor, seja por igualdade ou por desigualdade. No caso de campos tipo string, existe a possibilidade de comparação por aproximação

com o uso da palavra LIKE no lugar do operador de comparação.

Sintaxe para Projeção:

```
SELECT campo1, campoX,...,campoZ FROM nome_tabela
```

Sintaxe para Produto Cartesiano (Junção Natural):

```
SELECT * FROM nome_tabela1,nome_tabela2
    WHERE
        ChavePrim_tab1=ChaveEstr_tab2
```

DEFININDO ESTRUTURAS

b) Comando CREATE → Usado sempre em conjunto com outro parâmetro que indica o que será criado, podendo ser uma tabela, uma visão, uma trigger (gatilho) ou um índice secundário.

Segue um exemplo de uso do CREATE TABLE:

```
CREATE  TABLE IF NOT EXISTS `mydb`.`colecaoCD` (
      `codigo_CD` INT NOT NULL ,
      `artista` VARCHAR(60) NOT NULL ,
      `titulo` VARCHAR(45) NOT NULL ,
      `ano` INT NULL ,
      `qdeFaixas` INT NULL ,
PRIMARY KEY (`codigo_CD`) )
ENGINE = InnoDB;
```

c) Comando DROP → Apaga qualquer estrutura criada pelo commando anterior, sua sintaxe varia da mesma forma.

Exemplo (Eliminar toda uma tabela):

```
DROP TABLE nome_tabela;
```

d) Comando DELETE → Este comando apaga dados que atendam

determinado critério, ou seja, não atua sobre toda uma estrutura, mas atua sobre um grupo de um ou mais registros que atendam ao critério.

Exemplo (apagar um ou mais registros cujo bairro seja igual a “várzea”):

```
DELETE FROM nome_tabela
  WHERE
        Bairro=’várzea’;
```

Além destes comandos, existem muitos outros que implementam as mais diversas operações da álgebra relacional. Para uma melhor revisão de álgebra relacional, recomendamos a leitura do APÊNDICE 1 deste material.

Usando o Diagrama Salvo

Voltando ao MySQL Workbench OSS, abra o diagrama salvo anteriormente para gerarmos o script SQL que será executado no MySQL.

Com o diagrama reaberto, acesse o menu FILE e use a opção EXPORT, escolhendo a sub-opção “Forward Engenieer CREATE SQL Script...”.

Veja na figura 3.1 a seguir a tela que surgirá:

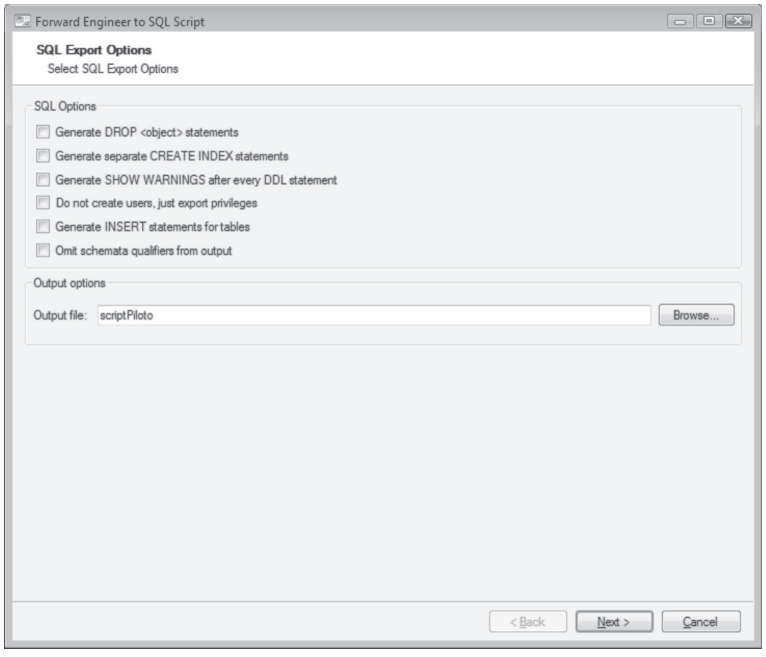


Figura 3.1 – Assistente de Criação de Script SQL

No output file, clique em BROWSE e informe um nome para seu arquivo de script, por exemplo, “scriptPiloto”. Este arquivo será criado em “Meus Documentos” e terá a extensão “.SQL”, para ser aberto em qualquer interface do MySQL. A figura 3.2 ilustra a etapa a seguir:

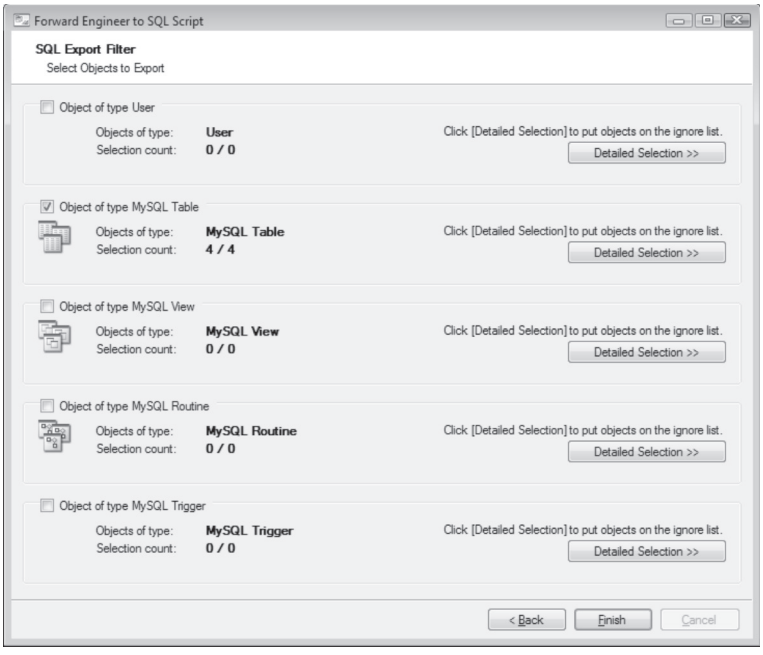


Figura 3.2 – Objetos a serem criados pelo script

De acordo com o diagrama que fizemos, vamos criar quatro tabelas relacionadas entre si por chaves estrangeiras. Portanto, a opção detectada pelo programa está correta. Podemos aprová-la clicando em FINISH.

Neste momento se você criou o database “LBD” em seu MySQL, no início desta disciplina, abra o script em um editor como o bloco de notas e mande substituir todas as ocorrências de “mybd” para “LBD” . Salve o script novamente.

Obs.: Em alguns editores o script poderá aparecer como se estivesse todo em uma linha só. Isto é normal e depende das configurações do seu editor. A execução no MySQL não será afetada.

Agora abra uma de nossas interfaces para MySQL. Usaremos na sequência deste texto o DB Tools Manager Standard Edition.

Na estrutura, vamos localizar o database LBD e clicar com o botão direito para criar uma nova tela de query. De acordo com a figura 3.3 a seguir:

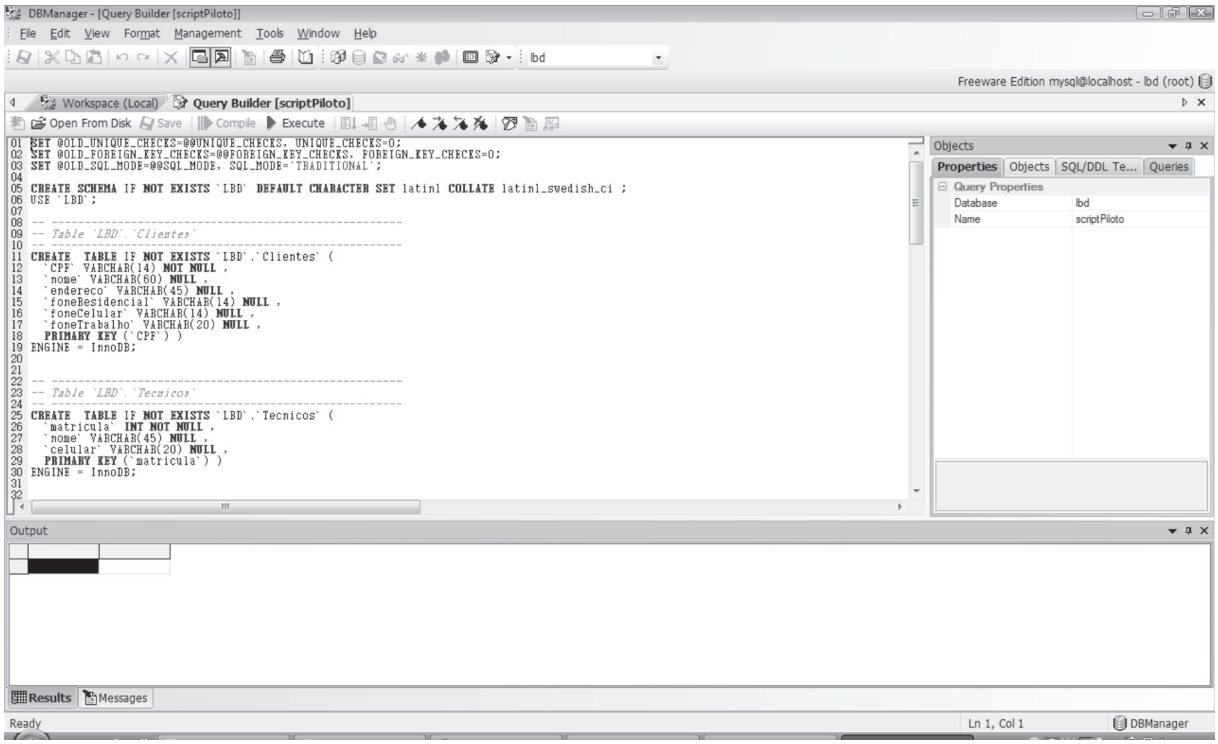


Figura 3.3 – DB Tools Manager com o script aberto

Usando o botão “open from disk”, abra o script SQL gravado em “Meus Documentos”. A tela ficará como visto acima na figura 3.3. Pressione a tecla F5 para executar o script e poderá observar a tela de output do programa com as mensagens de sucesso a seguir, conforme a figura 3.4.

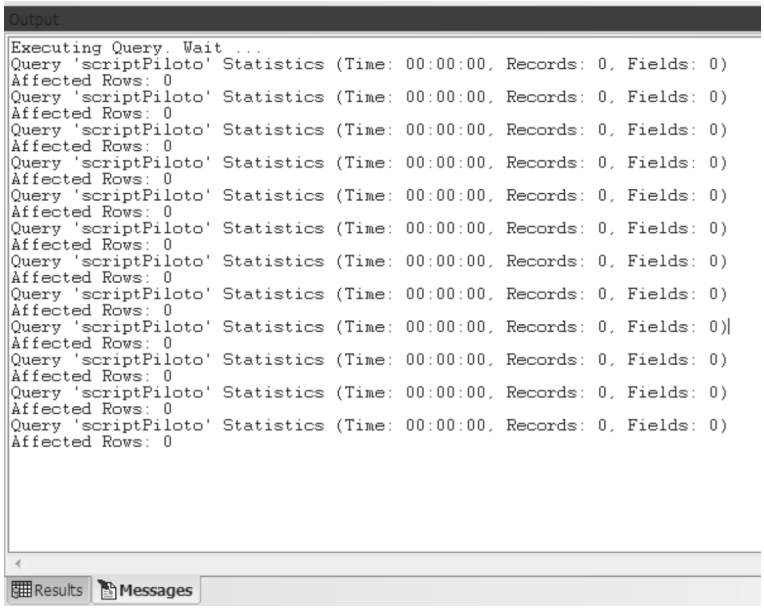


Figura 3.4 – Script executado com sucesso

Poderá também voltar na aba “Workspace (Local)” e comprovar nas tables criadas as novas tabelas em seu database. Conforme figura 3.5.

Figura 3.5 – Tabelas do database LBD

Segue uma listagem do script executado com sucesso, neste caso, para conferências, caso o seu tenha resultado em algum erro. Lembrando que, para uma nova execução deste script, será necessário usar o comando DROP TABLE para eliminar cada tabela criada pelo anterior que resultou em erro.

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL';

CREATE SCHEMA IF NOT EXISTS `LBD` DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci ;
USE `LBD`;

-- -----------------------------------------------------
-- Table `LBD`.`Clientes`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `LBD`.`Clientes` (
  `CPF` VARCHAR(14) NOT NULL ,
  `nome` VARCHAR(60) NULL ,
  `endereco` VARCHAR(45) NULL ,
```

```
  `foneResidencial` VARCHAR(14) NULL ,
  `foneCelular` VARCHAR(14) NULL ,
  `foneTrabalho` VARCHAR(20) NULL ,
  PRIMARY KEY (`CPF`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `LBD`.`Tecnicos`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `LBD`.`Tecnicos` (
  `matricula` INT NOT NULL ,
  `nome` VARCHAR(45) NULL ,
  `celular` VARCHAR(20) NULL ,
  PRIMARY KEY (`matricula`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `LBD`.`Aparelhos`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `LBD`.`Aparelhos` (
  `codigo` INT NOT NULL ,
  `descricao` VARCHAR(45) NULL ,
  `marca` VARCHAR(45) NULL ,
  `garantia` BOOLEAN NULL DEFAULT FALSE ,
  `prazoVencido` VARCHAR(3) NULL ,
  `valorEstimado` FLOAT NULL ,
  PRIMARY KEY (`codigo`) )
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `LBD`.`OS`
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS `LBD`.`OS` (
  `numero` INT NOT NULL ,
```

```
  `dataAbertura` DATE NULL ,
  `orcamento` FLOAT NULL ,
  `status` VARCHAR(2) NULL ,
  `defeitoReclamado` VARCHAR(60) NULL ,
  `dataConserto` DATE NULL ,
  `dataEntrega` VARCHAR(45) NULL ,
  `Clientes_CPF` VARCHAR(14) NULL ,
  `Tecnicos_matricula` INT NULL ,
  `Aparelhos_codigo` INT NULL ,
  PRIMARY KEY (`numero`) ,
  INDEX `fk_OS_Clientes` (`Clientes_CPF` ASC) ,
  INDEX `fk_OS_Tecnicos` (`Tecnicos_matricula` ASC) ,
  INDEX `fk_OS_Aparelhos` (`Aparelhos_codigo` ASC) ,
  CONSTRAINT `fk_OS_Clientes`
    FOREIGN KEY (`Clientes_CPF` )
    REFERENCES `LBD`.`Clientes` (`CPF` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_OS_Tecnicos`
    FOREIGN KEY (`Tecnicos_matricula` )
    REFERENCES `LBD`.`Tecnicos` (`matricula` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_OS_Aparelhos`
    FOREIGN KEY (`Aparelhos_codigo` )
    REFERENCES `LBD`.`Aparelhos` (`codigo` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Povoando as tabelas do Banco de Dados

Agora faremos o povoamento das tabelas do banco de dados. Lembrando que estes dados são hipotéticos e servem para fazer testes nas consultas e demais funções do aplicativo a ser desenvolvido.

Primeiro a tabela de Clientes:

INSERT INTO clientes VALUES (“111.222.333-00”,”José da Silva”,”Rua Pedro II,123”,”(86)3322-5599”,”(86)9999-4545”,”(86)3215-9988”);

INSERT INTO clientes VALUES (“231.223.333-00”,”Ana Maria Ferraz”,”Rua Piauí,123”,”(86)3222-5599”,”(86)9432-4545”,”(86)3215-9998”);

A tabela de Técnicos:

INSERT INTO tecnicos VALUES (1,”José Antônio”,”(86)9988-0099”);

INSERT INTO tecnicos VALUES (2,”Antônio Pedro”,”(86)8102-0099”);

Alguns aparelhos:

INSERT INTO aparelhos VALUES (1,”Televisão de LCD”,”CCE”,0,FALSE,1230.00);

INSERT INTO aparelhos VALUES (2,”Som”,”Tohiba”,0,FALSE,230.00);

INSERT INTO aparelhos VALUES (3,”FAX”,”Panasonic”,0,FALSE,830.00);

INSERT INTO aparelhos VALUES (4,”TV CRT29”,”CCE”,0,FALSE,630.00);

Agora finalmente o cadastro das Ordens de Serviços:

INSERT INTO os VALUES (1,’2010-08-10’,123.00,”AP”,”Nao liga e treme o som”,”2010-08-11”,”2010-08-12”,”111.222.333-00”,1,1);

INSERT INTO os VALUES (2,’2010-08-10’,123.00,”AP”,”sem som”,”2010-08-21”,”2010-08-22”,”111.222.333-00”,1,2);

INSERT INTO os VALUES (3,’2010-08-10’,123.00,”AP”,”Nao liga”,”2010-08-11”,”2010-08-22”,”111.222.333-00”,1,3);

INSERT INTO os VALUES (4,’2010-08-10’,123.00,”AP”,”Imagem sem foco”,”2010-08-22”,”2010-08-28”,”111.222.333-00”,1,4);

Veja que se o aplicativo estivesse pronto, a partir da tela de cadastro

de OS, deveria ser possível acessar todas as telas relacionadas às chaves estrangeiras como: Clientes, Aparelhos e Técnicos. Este acesso seria para buscar as chaves requeridas através de consultas ou para cadastrá-las caso não existam.

Algumas consultas com os dados existentes

Nesta etapa, ilustraremos algumas consultas que podem ser feitas com os dados criados para o projeto piloto.

a) Uma consulta envolvendo Ordens de Serviço e Aparelhos para que estes sejam identificados juntamente com suas OS:

- Para executar esta consulta, precisamos criar um produto cartesiano filtrado entre as tabelas OS e Aparelhos.

SELECT numero, dataAbertura, aparelhos_codigo, descricao, marca FROM OS,aparelhos WHERE os.Aparelhos_codigo=aparelhos.codigo;

O Resultado desta seria algo como:

Numero	dataAbertura	Aparelhos_codigo	Descricao	marca
1	10/08/2010	1	Televisão de LCD	CCE
2	10/08/2010	2	Som	Tohiba
3	10/08/2010	3	FAX	Panasonic
4	10/08/2010	4	TV CRT29	CCE

b) Uma listagem de proprietários de aparelhos:

- Para executar esta consulta, o produto cartesiano vai envolver três tabelas: OS, aparelhos e clientes. É importante “amarrar” os devidos códigos para evitar problemas com tuplas-fantasmas.

SELECT clientes.nome, OS.dataAbertura, aparelhos.descricao, aparelhos.marca FROM clientes,OS,aparelhos

```
WHERE
  clientes.CPF=os.Clientes_CPF AND
  aparelhos.codigo=os.Aparelhos_codigo;
```

O Resultado desta consulta é este:

Clientes.nome dataAbertura Aparelhos.descricao Aparelhos.marca
José da Silva 10/08/2010 Televisão de LCD CCE
José da Silva 10/08/2010 Som Tohiba
Ana Maria Ferraz 10/08/2010 FAX Panasonic
José da Silva 10/08/2010 TV CRT29 CCE

c) Fazendo atualizações em registros:

- Faz-se uso do comando UPDATE para atualizar dados em tabelas. Seguem exemplos de atualizações em alguns registros de OS:

Atualizar técnico responsável da OS número 4:

```
UPDATE OS
SET   Tecnicos_matricula=2
 WHERE numero = 4;
```

Atualizar o cliente proprietário da OS de número 3:

```
UPDATE OS
SET   clientes_CPF='231.223.333-00'
WHERE numero = 3;
```

Projetos Individuais

Toda a implementação deste capítulo deverá ser adaptada ao projeto individual de cada aluno. Desde o script de criação das tabelas, passando pelo povoamento das mesmas e chegando a algumas consultas.

Uma vez concluído o seu trabalho, todos os dados das tabelas

podem ser salvos para entregar ao seu tutor, basta fazer um DUMP do DATABASE. Na aba workspace do DB Manager, clique com o botão direito sobre o database pretendido e escolha a opção DUMP. Um assistente abrirá conforme as figuras 3.6 e 3.7 a seguir:

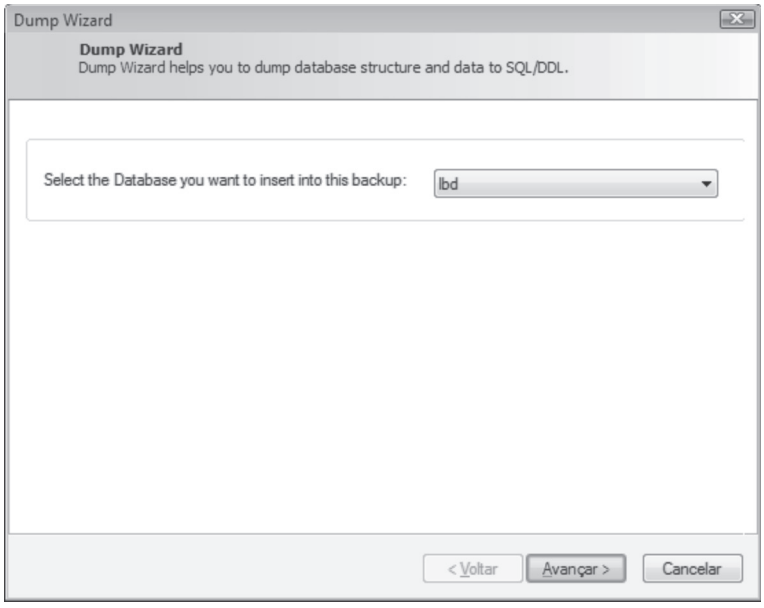


Figura 3.6 – Inicio do processo de DUMP do database LBD

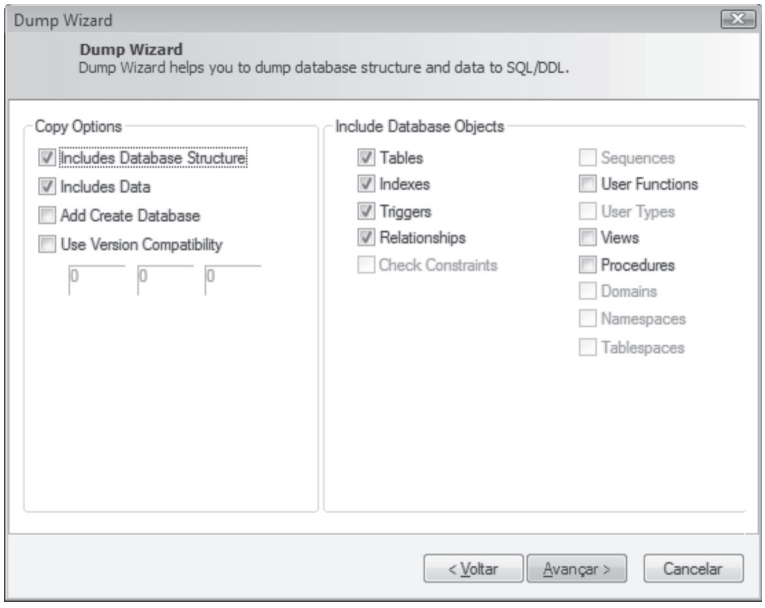


Figura 3.7 – Seleção dos objetos a serem copiados

Na etapa seguinte, devemos informar o nome do arquivo com extensão SQL a ser criado com os dados, usa-se o botão com reticências e informa-se o nome do arquivo a ser criado. Na última etapa, antes de pressionar concluir, pode-se salvar as definições deste DUMP para executá-lo novamente em outra ocasião.

Um DUMP de banco de dados pode ser usado como backup. Sendo com uma particularidade: um backup leva junto de si os dados de identificação do computador original, e a restauração somente é permitida neste computador. Um DUMP leva apenas dados que podem servir para povoar um Banco de Dados do mesmo tipo criado em qualquer outra máquina.

Listagem de arquivos a serem entregues

Nesta etapa da disciplina todos devem estar com seus projetos individuais prontos. Sendo assim, os seguintes arquivos devem ser entregues aos tutores:

a) A descrição do estudo de caso → texto digitado em formato ".doc";

b) O Diagrama Entidades-Relacionamentos → criado e salvo no MySQL Workbench OSS;

c) O Script ".SQL" exportado pelo MySQL Workbench OSS para criação do Banco de Dados no MySQL;

d) Os comandos de SQL usados para povoamento das tabelas e para fazer as consultas;

e) Finalmente, o DUMP do database onde foi feito o seu trabalho em sua máquina.

Lembre-se que este trabalho se constitui em uma das notas para compor sua média final da disciplina.

REFERÊNCIAS NA WEB

MySQL Reference Manual:
http://dev.mysql.com/doc/mysql/
Manual de Referência do MySQL:
http://hidouci.esi.dz/manual_MySQL.pdf

Referências

ALVES, William Pereira. **Fundamentos de Bancos de Dados**. São Paulo: Érica, 2004. 382p.

COUGO, Paulo. **Modelagem Conceitual e Projeto de Banco de Dados**. 1ª edição. 4ª tiragem. Ed. CAMPUS, São Paulo, 2000.

ELMASRI, Ramez & NAVATHE, Shamkant B. F**undamentos de Sistemas de Banco de Dados**. 4ª Edição. Ed. Pearson/Addinson Weslley, 2006.

GUIMARÃES, Célio C. **Fundamentos de Bancos de Dados**. Ed. Unicamp, Campinas SP, 2003.

KORTH, Henry et all. **Sistemas de Banco de Dados**. 5ª edição. Ed. CAMPUS, São Paulo, 2006.

MACHADO, Felipe Nery Rodrigues. **Banco de dados: projeto e implementação**. São Paulo: Érica, 2004. 398p.

APÊNDICE I – REVISÃO DE ÁLGEBRA RELACIONAL

Esta é uma linguagem de consulta procedural que será detalhada em suas operações fundamentais e avançadas a partir deste ponto. Já existem diversas evoluções de versão tanto da álgebra relacional quanto da SQL, neste material serão vistas as operações mais populares desta dupla.

a) Operação Seleção (σ)

Esta operação permite selecionar em uma tabela registros que atendam a um determinado critério. Seu símbolo é a letra sigma minúscula. A sua sintaxe é mostrada logo abaixo:

$\sigma_{(campoX = valor)}$ (nomeTabela)

Observe que o critério aparece subscrito ao símbolo da operação, o nome da tabela deve aparecer no mesmo nível da operação. Para a formulação do critério, pode-se usar qualquer campo da tabela, juntamente com qualquer símbolo de comparação válido, tal como: =, <>, < , >, <=, >= ; além do valor a ser comparado ao campo condizente com o domínio do campo, ou seja, para um campo inteiro deve-se fazer comparações com valores do tipo inteiro, e assim por diante.

Tomando a tabela a seguir como exemplo:

Alunos

Matr *	**Nome**	**Nome**	**Fone**	**Cod. Curso**[CE]
123	Flávio José	Rua Pedro, 2290	5859-6930	01
205	Aline Maria	Rua Pereira, 224	7452-8520	01
330	José Luiz	Rua Pedro, 450	3358-9665	02
440	Clarita Silva	Rua Riachuelo, 177	6598-8754	05

Fazendo a operação de seleção para o nome de aluno igual a ‘Aline Maria’, obtém-se:

$\sigma_{(alunos.nome = 'Aline\ Maria')}$ (Alunos)

205	Aline Maria	Rua Pereira, 224	7452-8520	01

Observe que todo o registro é retornado pela operação. Esta mesma operação em SQL seria escrita assim:

```
SELECT * FROM alunos
WHERE
    Alunos.nome = 'Aline Maria'
```

Algumas considerações a respeito desta cláusula SQL:

1. Os termos destacados em maiúsculas são apenas para chamar atenção para as palavras reservadas da linguagem, pois esta não é 'case-sensitive';
2. Tanto em álgebra relacional quanto em SQL, ao citar um campo de tabela é opcional colocar antes o nome da tabela e usar o ponto para referenciar o campo, mas esta é uma boa e elegante prática de programação, pois ajuda na leitura e compreensão do algoritmo;
3. Especificamente para campos string e assemelhados, a SQL possui a possibilidade de usar o critério de comparação LIKE associado ao caractere coringa %, sendo assim poderíamos fazer seleção de todos os alunos ,cuja rua tivesse por exemplo a string 'Pedro' na sua denominação. Segue o exemplo:

```
SELECT * FROM alunos
WHERE
    Alunos.endereço LIKE '%Pedro%'
```

Isto retorna registros cujo endereço contenha 'Pedro' em qualquer posição:

123	Flávio José	Rua Pedro, 2290	5859-6930	01
330	José Luiz	Rua Pedro, 450	3358-9665	02

b) Operação Projeção (π)

Esta operação simbolizada pela letra grega pi minúscula, lista apenas alguns campos da tabela. É útil quando, em alguns resultados, torna-se importante omitir certos valores em cada registro da tabela. A sintaxe da operação é a seguinte:

$\pi_{(campo\_1, \ldots, campo\_n)}(nomeTabela)$

Como exemplo, tome-se a tabela já anteriormente descrita para fazer a projeção dos campos de matrícula e de nome dos alunos.

$\pi_{(matr, nome)}(Alunos)$

Matr	Nome
123	Flávio José
205	Aline Maria
330	José Luiz
440	Clarita Silva

Em SQL, a cláusula para obter este mesmo resultado seria:

```
SELECT Matr, Nome FROM alunos
```

Observa-se que:

1. O comando SELECT não deve ser traduzido como seleção, pois serve para implementar várias operações da álgebra relacional;
2. Para implementar a projeção faz-se a substituição do tradicional coringa * após o select pela lista de campos a serem projetados;
3. Por questões de elegância, o nome da tabela pode também ser usado antes de cada referência aos seus campos.

c) Operação Produto Cartesiano (T1 x T2).

Nesta operação que envolve duas tabelas, acontece a inevitável volta ao passado na vida de todos, volta-se aos tempos da escolinha onde a professora, tia para alguns, desenhava dois diagramas de Venn cheios de elementos, e dizia que o Produto Cartesiano dos dois conjuntos significava apenas ligar cada elemento do primeiro conjunto a TODOS os elementos do segundo conjunto. Além disso certamente não havia utilidade para esta operação de conjuntos em sua vida. Portanto, aprenda agora e ensine depois para a sua ex-professora a grande utilidade desta operação.

No universo relacional é correto cada tabela ter seus devidos dados e todas as tabelas ficarem relacionadas entre si, direta ou indiretamente, através de suas chaves primárias e estrangeiras. Às vezes faz-se necessário ligar temporariamente duas tabelas para satisfazer uma consulta, para isto, o produto cartesiano é usado. Segue um exemplo de duas tabelas relacionadas e seu produto cartesiano:

Alunos

Matr*	**Nome**	**Endereço**	**Fone**	**Cód. Curso**[CE]
123	Flávio José	Rua Pedro, 2290	5859-6930	01
205	Aline Maria	Rua Pereira, 224	7452-8520	01
330	José Luiz	Rua Pedro, 450	3358-9665	02
440	Clarita Silva	Rua Riachuelo, 177	6598-8754	05

Cursos

Código*	**Descrição**	**Carga horária**
01	Informática	120
02	Veterinária	220
05	Agronomia	190

Onde Alunos.codCurso é referente a Cursos.código.

A sintaxe do produto cartesiano é:

Tabela_1 x Tabela_2

Sendo assim, para realizar o cruzamento das informações de alunos e cursos seria: Alunos x Cursos, obtendo-se ligações do tipo:

123	Flávio José	Rua Pedro, 2290	5859-6930	01	01	Informática	120

Porém, esta operação conta com um grande problema: as ligações indevidas que originam as chamadas duplas-fantasmas, ou seja, como existem três cursos, o aluno 'Flávio José' e todos os demais apareceriam ligados aos seus cursos originais e também aos outros cursos que não fazem, originando informações inverídicas. Tais como:

123	Flávio José	Rua Pedro, 2290	5859-6930	01	**02**	**Veterinária**	**220**
123	Flávio José	Rua Pedro, 2290	5859-6930	01	**05**	**Agronomia**	**190**

Para resolver este tipo de problema, usa-se uma operação de seleção conjugada ao produto cartesiano. Esta seleção é conhecida como seleção de filtro, pois fica responsável por mostrar apenas a verdade. A operação seria refeita da seguinte maneira:

$\sigma_{\text{Alunos. Cód. Curso = Cursos.código}}$ (Alunos x Cursos)

E o resultado seria o correto:

Matr*	**Nome**	**Endereço**	**Fone**	**Cód. Cursos**^{CE}	**Código**	**Descrição**	**Carga Horária**
123	Flávio José	Rua Pedro, 2290	5859-6930	01	01	Informática	120

Matr*	Nome	Endereço	Fone	Cód. Cursos^{CE}	Código	Descrição	Carga Horária
205	Aline Maria	Rua Pereira, 2247452-8520	7452-8520	01	01	Informática	120
330	José Luiz	Rua Pedro, 450	3358-9665	02	02	Veterinária	220
440	Clarita Silva	Rua Riachuelo, 177	6598-8754	05	05	Agronomia	190

Esta combinação de operações é tão necessária que se define uma operação avançada da álgebra relacional tendo exatamente este significado: a JUNÇÃO NATURAL, simbolizada por: ⋈

Então, a expressão anterior poderia ser substituída pela junção natural:

Alunos ⋈ Cursos

Obtendo-se o mesmo resultado.

Em SQL, a expressão para realizar esta junção natural é:

```
SELECT * FROM alunos, cursos
WHERE
    Alunos.codCurso = Cursos.código
```

Conceito: União-Compatível

Diz-se que duas tabelas são união-compatíveis quando ambas têm a mesma quantidade de campos com a mesma sequência de domínios, ou seja, para uma tabela de três campos sendo um inteiro, um varchar (string) e um lógico; a outra tabela terá obrigatoriamente um inteiro, um varchar e um lógico para ser união-compatível. Algumas operações da álgebra relacional são definidas sobre este conceito, tais como: união, diferença e intersecção.

d) União de Tabelas (T1 U T2)

Esta operação é definida entre duas tabelas união-compatíveis, consiste em juntar os conteúdos de duas tabelas em uma só.

A Sintaxe desta operação é:

Tabela1 U Tabela2

Segue um exemplo deste caso.

Clientes			**Funcionários**		
Cod*	**Nome**	**Fone**	**Matr***	**NomeF**	**FoneF**
01	Flávio	9966-6363	120	Ana	8855-2121
02	Aline	9494-4477	121	José	8101-8585

A operação *Clientes U Funcionarios* resulta em:

Cod*	**Nome**	**Fone**
01	Flávio	9966-6363
02	Aline	9494-4477
120	Ana	8855-2121
121	José	8101-8585

Em SQL, usa-se a palavra reservada UNION para unir duas consultas e obter resultado para esta operação.

```
SELECT * FROM CLIENTES
UNION
SELECT * FROM FUNCIONARIOS
```

e) Diferença de Tabelas (T1 – T2)

Desta vez a operação também é definida entre duas tabelas união-compatíveis. A semelhança fica ligada à diferença de conjuntos, pois no resultado ficam os dados que estão em uma tabela, mas não estão na outra. Segue um exemplo desta operação:

Clientes-Poupança			**Clientes-ContaCorrente**		
Cod*	**Nome**	**Fone**	**Matr***	**NomeF**	**FoneF**
01	Flávio	9966-6363	120	Ana	8855-2121
02	Aline	9494-4477	121	José	8101-8585
			01	Flávio	9966-6363

Fazendo a operação *Clientes-Poupança - Clientes-ContaCorrente*, obtém-se:

02	Aline	9494-4477

Ou seja, o cliente que tem conta-poupança, mas não tem conta-corrente.

f) Intersecção de Tabelas (T1 ∩ T2)
Esta operação, que também é definida entre tabelas união-compatíveis, retorna algo semelhante a intersecção de conjuntos, ou seja, tudo aquilo que é comum às duas tabelas.

Retomando o exemplo da operação anterior, pode-se investigar o cliente que tem conta-poupança e conta-corrente. Para tanto usa-se a operação *Clientes-Poupança* ∩ *Clientes-ContaCorrente*. Com o seguinte resultado:

01	Flávio	9966-6363

Em MySQL, esta operação é feita com a ajuda do comando INNER JOIN:

```
SELECT código, nome, cellular FROM Clientes-Poupanca P
      INNER JOIN clientes-Conta Corrente C ON (P.código = C.código)
ORDER BY P.código
```

Anexo 1 – Referência Rápida de MySQL encontrada na Internet

Este texto que segue é encontrado em diversos sites na Internet e é uma boa referência rápida para MySQL. A versão original deste texto está armazenada na URL: http://apostilas.netsaber.com.br/list_apostilas_c_63.html e seu autor não é citado.

Instrução SELECT

Instrui o programa principal do banco de dados para retornar a informação como um conjunto de registros.

Sintaxe

SELECT [predicado { * | tabela.* | [tabela.]campo1 [AS alias1] [, [tabela.]campo2 [AS alias2] [, ...]]}

FROM expressãotabela [, ...] [IN bancodedadosexterno]

[WHERE...]

[GROUP BY...]

[HAVING...]

[ORDER BY...]

[WITH OWNERACCESS OPTION]

A instrução SELECT tem as partes abaixo:

Parte Descrição

predicado Um dos seguintes predicados: ALL, DISTINCT, DISTINCTROW ou TOP.

Você usa o predicado para restringir o número de registros que retornam. Se nenhum for especificado, o padrão será ALL.

* Especifica que todos os campos da tabela ou tabelas especificadas são selecionados.

Tabela O nome da tabela que contém os campos dos quais os registros são selecionados.

campo1, campo2 Os nomes dos campos dos quais os dados serão recuperados. Se você incluir mais de um campo, eles serão recuperados na ordem listada.

alias1, alias2 Os nomes que serão usados como títulos de colunas em vez dos nomes originais das colunas na tabela.

expressãotabela O nome da tabela ou tabelas contendo os dados que você quer recuperar.

bancodedadosexterno O Nome do banco de dados que contém as tabelas em expressãotabela se não estiver no banco de dados atual.

Comentários

Para executar esta operação, o programa principal de banco de dados procura a tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que satisfazem o critério e classifica ou agrupa as linhas resultantes na ordem especificada.

A instrução SELECT não muda os dados no banco de dados. SELECT é normalmente a primeira palavra em uma instrução SQL. A maior parte das instruções SQL são instruções SELECT.

A sintaxe mínima da instrução SELECT é:

SELECT campos FROM tabela

Você pode usar um asterisco (*) para selecionar todos os campos na tabela. O exemplo abaixo seleciona todos os campos na tabela Funcionários:

SELECT * FROM Funcionários;

Se o nome de um campo estiver incluído em mais de uma tabela na cláusula FROM, preceda-o com o nome da tabela e o operador . (ponto). No exemplo abaixo, o campo Departamento está nas tabelas Funcionários e Supervisores. A instrução SQL seleciona Departamento da tabela Funcionários e NomeSupv da tabela Supervisores:

SELECT Funcionários.Departamento, Supervisores.NomeSupv;

FROM Funcionários INNER JOIN Supervisores;

WHERE Funcionários.Departamento = Supervisores.Departamento.

Ao criar um objeto Recordset, o programa principal de banco de dados do Jet usa o nome do campo da tabela como o nome do objeto Field no objeto Recordset. Se você quiser um nome de campo diferente ou um nome que não esteja implícito na expressão usada para gerar o campo, use a palavra reservada AS. O exemplo abaixo usa o título Nasc para nomear o objeto Field retornado no objeto Recordset resultante:

SELECT DataNasc AS Nasc FROM Funcionários;

Sempre que você usar funções aggregate ou consultas que retornem nomes de objetos Field ambíguos ou duplicados, você precisará usar a cláusula AS para fornecer um nome alternativo para o objeto Field. O exemplo abaixo usa o título Contagem para nomear o objeto Field retornado no objeto Recordset resultante:

SELECT COUNT(FuncionárioID) AS Contagem FROM Funcionários;

Você pode usar outras cláusulas na instrução SELECT para restringir e organizar posteriormente os seus dados retornados.

Cláusula GROUP BY

GROUP BY é opcional. Valores de resumo são omitidos se não houver qualquer função aggregate SQL na instrução SELECT. Os valores Null nos campos GROUP BY são agrupados e não omitidos. No entanto, os valores Null não são avaliados em qualquer função aggregate SQL. Use a cláusula WHERE para excluir linhas que você não quer agrupadas e use a cláusula HAVING para filtrar os registros após eles terem sido agrupados.

A não ser que contenha dados Memo ou OLE Object, um campo na lista de campos GROUP BY pode fazer referência a qualquer campo em qualquer tabela listada na cláusula FROM. Mesmo que o campo não esteja incluído na instrução SELECT, fornecida a instrução SELECT, inclua pelo menos uma função SQL. O programa principal de banco de dados do Jet não pode agrupar campos Memo ou OLE Objects.

Todos os campos na lista de campos SELECT devem ser incluídos na cláusula GROUP BY ou incluídos como argumentos em uma função aggregate SQL.

Cláusula HAVING

HAVING é opcional. HAVING é semelhante a WHERE, que determina quais registros são selecionados. Depois que os registros são agrupados com GROUP BY, HAVING determina quais registros são exibidos:

SELECT CategoriaID, Sum(UnidadesNoEstoque) FROM Produtos

GROUP BY CategoriaID

HAVING Sum(UnidadesNoEstoque) > 100 AND LIKE “BOS*”

Uma cláusula HAVING pode conter até 40 expressões vinculadas por operadores lógicos, como And ou Or.

Cláusula ORDER BY

ORDER BY é opcional. Entretanto, se você quiser exibir seus dados na ordem classificada, você deve utilizar ORDER BY. O padrão ordem de classificação é ascendente (A a Z, 0 a 9). Os dois exemplos abaixo classificam os nomes dos funcionários pelo sobrenome.

SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome;

SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome ASC;

Para classificar em ordem descendente (Z a A, 9 a 0), adicione a palavra reservada DESC ao final de cada campo que você quiser classificar em ordem descendente. O exemplo abaixo seleciona salários e os classifica em ordem descendente.

SELECT Sobrenome, Salário FROM Funcionários ORDER BY Salário DESC, Sobrenome;

Se você especificar um campo que contém dados Memo ou OLE Objects na cláusula ORDER BY, um erro ocorrerá. O programa principal de banco de dados do Jet não classifica campos deste tipo. ORDER BY é normalmente o último item em uma instrução SQL.

Você pode incluir campos adicionais na cláusula ORDER BY. Os registros são classificados primeiro pelo primeiro campo listado depois de ORDER BY. Os registros que tiverem valores iguais naquele campo são classificados pelo valor no segundo campo listado e assim por diante.

Cláusula WITH OWNERACCESS OPTION

A declaração WITH OWNERACCESS OPTION é opcional. O exemplo abaixo habilita o usuário a ver as informações de salário (mesmo que não tenha outra permissão para ver a tabela Folha de Pagamentos) desde que o proprietário da consulta tenha tal permissão:

SELECT Sobrenome, Nome, Salário FROM Funcionários ORDER BY

Sobrenome;

WITH OWNERACCESS OPTION;

Se, por outro lado, um usuário for impedido de criar ou anexar a uma tabela, você poderá usar WITH OWNERACCESS OPTION para habilitá-lo a executar uma consulta construção de tabela ou consulta anexação. Se você quiser reforçar as configurações de segurança do grupo de trabalho e as permissões dos usuários, não inclua a declaração WITH OWNERACCESS OPTION. Esta opção exige que você tenha acesso ao arquivo System. mda associado ao banco de dados. É realmente útil em implementações de multiusuários seguras.

Exemplo da instrução SELECT, cláusula FROM

Esse exemplo seleciona os campos “Sobrenome” e “Nome” de todos os registros da tabela “Funcionários”.

SELECT Sobrenome, Nome FROM Funcionários

Esse exemplo seleciona todos os campos da tabela “Funcionários”.

SELECT Funcionários.* FROM Funcionários;

Esse exemplo conta o número de registros que têm uma entrada no campo “CódigoPostal” e nomeia o campo retornado como “Tcp”.

SELECT Count(CódigoPostal) AS Tcp FROM Clientes;

Esse exemplo mostra qual seria o salário se cada funcionário recebesse um aumento de dez porcento. Não altera o valor original dos salários.

SELECT Sobrenome, Salário AS Atual, Salário * 1.1 AS Proposto FROM Funcionários;

Esse exemplo coloca o título Nome no topo da coluna “Sobrenome”. O título Salário é exibido no topo da coluna “Salário”.

SELECT Sobrenome AS Nome, Salário FROM Funcionários;

Esse exemplo mostra o número de funcionários e os salários médio e máximo.

SELECT Count(*) AS [Total de Funcionários], Avg(Salário) AS [Salário Médio], Max(Salário) AS [Salário Máximo] FROM Funcionários;

Para cada registro, mostra Sobrenome e Salário no primeiro e último campos. A sequência de caracteres “tem um salário de” é retornada como o campo do meio de cada registro.

SELECT Sobrenome, ‘tem um salário de’, Salário FROM Funcionários;

Exemplo de cláusula GROUP BY

Esse exemplo cria uma lista de nomes de departamentos únicos e o número de funcionários em cada um destes departamentos.

SELECT Departamento, Count([Departamento]) AS Tbc FROM Funcionários

GROUP BY Departamento;

Para cada título de função único, calcula o número de funcionários do departamento de Vendas que têm este título.

SELECT Título, Count(Título) AS Tbc FROM Funcionários

WHERE Departamento = ‘Vendas’ GROUP BY Título;

Esse exemplo calcula o número de itens em estoque para cada combinação de número e cor do item.

SELECT Item, Sum(Unidades) AS Tbc FROM ItensEmEstoque

GROUP BY Item, Cor;

Exemplo de cláusula HAVING

Esse exemplo seleciona os títulos de cargos do departamento de Produção atribuídos a mais de cinquenta funcionários.

SELECT Título, Count(Título) FROM Funcionários WHERE Departamento = 'Produção'

GROUP BY Título HAVING Count(Título) > 50;

Esse exemplo seleciona os departamentos que tenham mais de cem funcionários.

SELECT Departamento, Count([Departamento]) FROM Funcionários

GROUP BY Departamento HAVING Count(Departamento) > 100;

Exemplo de cláusula ORDER BY

As instruções SQL mostradas abaixo usam a cláusula ORDER BY para classificar os registros em ordem alfabética e depois por categoria.

Esse exemplo ordena os registros pelo sobrenome, em ordem descendente (Z-A).

SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome DESC;

Esse exemplo ordena, primeiro, por categoria ID e depois por nome do produto.

SELECT CategoriaID, ProdutoNome, PreçoUnit FROM Produtos

ORDER BY CategoriaID, NomeProduto;

Instrução INSERT

Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.

Sintaxe

Consulta anexação de vários registros:

INSERT INTO destino [IN bancodedadosexterno] [(campo1[, campo2[, ...]])]

SELECT [origem.]campo1[, campo2[, ...]

FROM expressãodetabela

Consulta anexação de um único registro:

INSERT INTO destino [(campo1[, campo2[, ...]])]

VALUES (valor1[, valor2[, ...])

A instrução INSERT INTO tem as partes abaixo:

Parte Descrição

destino O nome da tabela ou consulta em que os registros devem ser anexados.

nobancodedadosexterno O caminho para um banco de dados externo. Para uma descrição do caminho, consulte a cláusula IN.

origem O nome da tabela ou consulta de onde os dados devem ser copiados.

campo1, campo2 Os nomes dos campos aos quais os dados devem ser anexados, se estiverem após um argumento destino ou os nomes dos campos dos quais se deve obter os dados, se estiverem após um argumento origem.

Expressãodetabela O nome da tabela ou tabelas das quais registros são inseridos. Este argumento pode ser um único nome de tabela ou uma combinação resultante de uma operação INNER JOIN, LEFT JOIN ou RIGHT JOIN ou de uma consulta gravada.
alor1, valor2 Os valores para inserir em campos específicos do novo registro.

Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante. Você deve separar os valores com uma vírgula e colocar os campos de textos entre aspas (“ “).

Comentários

Você pode usar a instrução INSERT INTO para adicionar um único registro a uma tabela usando a sintaxe de consulta anexação de um único registro como mostrado acima. Neste caso, seu código especifica o nome e o valor de cada campo do registro. Você precisa especificar cada um dos campos do registro para os quais um valor deve ser designado e um valor para este campo. Quando você não especifica cada campo, o valor padrão ou Null é inserido nas colunas omitidas. Os registros são adicionados no final da tabela.

Você também pode usar INSERT INTO para anexar um conjunto de registros de outra tabela ou consulta usando a cláusula SELECT ... FROM como é mostrado acima na sintaxe consulta anexação de vários registros. Neste caso, a cláusula SELECT especifica os campos que serão acrescentados na tabela destino especificada.

A tabela de origem ou de destino pode especificar uma tabela ou uma consulta. Se uma consulta for especificada, o programa principal de banco de dados do Microsoft anexa a qualquer e a todas as tabelas especificadas pela consulta.

INSERT INTO é opcional, mas quando incluída, precede a instrução SELECT.

Se sua tabela de destino contém uma chave primária, você deve acrescentar valores únicos, não Null ao campo ou campos da chave primária. Caso contrário, o programa principal de banco de dados do Jet não anexará os registros.

Se você anexar registros a uma tabela com um campo Counter e quiser numerar novamente os registros anexados, não inclua o campo

Counter em sua consulta. Inclua o campo Counter na consulta se quiser manter os valores originais do campo.

Use a cláusula IN para anexar registros a uma tabela de outro banco de dados. Para achar quais registros serão anexados, antes de você executar a consulta anexação, primeiro execute e veja os resultados de uma consulta seleção que use o mesmo critério de seleção.

Uma operação de consulta anexação copia os registros de uma ou mais tabelas em outra. As tabelas que contêm os registros que você anexa não são afetadas pela operação de consulta anexação.

Em lugar de acrescentar registros existentes de outra tabela, você pode especificar o valor de cada campo em um único registro novo usando a cláusula VALUES. Se você omitir a lista de campo, a cláusula VALUES deve incluir um valor para cada campo na tabela; caso contrário, um erro ocorrerá em INSERT. Use uma instrução adicional INSERT INTO com uma cláusula VALUES para cada registro adicional que você quiser criar.

Exemplo de instrução INSERT

Esse exemplo seleciona todos os registros de uma tabela hipotética “Novos Clientes” e os adiciona à tabela “Clientes” (quando não são designadas colunas individuais, os nomes das colunas das tabelas SELECT devem corresponder exatamente aos da tabela INSERT INTO).

INSERT INTO Clientes SELECT [Novos Clientes].*

FROM [Novos Clientes];

Esse exemplo cria um novo registro na tabela “Funcionários”

INSERT INTO Funcionários (Nome,Sobrenome, Título)

VALUES (“André”, “Pereira”, “Estagiário”);

Esse exemplo seleciona todos os estagiários de uma tabela hipotética

“Estagiários” que foram contratados há mais de trinta dias e adiciona seus registros à tabela “Funcionários”.

INSERT INTO Funcionários SELECT Estagiários.*

FROM Estagiários WHERE DataContrato < Now() - 30;

Instrução UPDATE

Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.

Sintaxe

UPDATE tabela

SET valornovo

WHERE critério;

A instrução UPDATE tem as partes abaixo:

Parte Descrição

tabela O nome da tabela cujos dados você quer modificar.

valornovo Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.

critério Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizados.

Comentários

UPDATE é especialmente útil quando você quer alterar muitos registros ou quando os registros que você quer alterar estão em várias tabelas. Você pode alterar vários campos ao mesmo tempo. O exemplo abaixo aumenta o

Valor do Pedido em dez por cento e o valor do Frete em três por cento para embarques do Reino Unido:

UPDATE Pedidos SET ValorPedido = ValorPedido * 1.1, Frete = Frete * 1.03

WHERE PaísEmbarque = 'RU';

UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

Exemplo de instrução UPDATE

Esse exemplo muda os valores no campo "RelatórioPara" para 5 para todos os registros de funcionários que atualmente têm valores de RelatórioPara de 2.

UPDATE Funcionários SET RelatórioPara = 5 WHERE RelatórioPara = 2;

Esse exemplo aumenta o "PreçoUnit" de todos os produtos não suspensos do fornecedor 8 em dez porcento.

UPDATE Produtos SET PreçoUnit = PreçoUnit * 1.1

WHERE FornecedorID = 8 AND Suspenso = No;

Esse exemplo reduz o PreçoUnit de todos os produtos não suspensos fornecidos pela Tokyo Traders em cinco porcento. As tabelas "Produtos" e "Fornecedores" têm uma relação um para vários.

UPDATE Fornecedores INNER JOIN Produtos

ON Fornecedores.FornecedorID = Produtos.FornecedorID SET PreçoUnit = PreçoUnit * .95

WHERE NomeEmpresa = 'Tokyo Traders' AND Suspenso = No;

Instrução DELETE

Cria uma consulta exclusão que remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE.

Sintaxe

DELETE [tabela.*]

FROM tabela

WHERE critério

A instrução DELETE tem as partes abaixo:

Parte Descrição

tabela.* O nome opcional da tabela da qual os registros são excluídos.

tabela O nome da tabela da qual os registros são excluídos.

critério Uma expressão que determina qual registro deve ser excluído.

Comentários

DELETE é especialmente útil quando você quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, você pode usar o método Execute com uma instrução DROP.

Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa DELETE, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Você pode usar DELETE para remover registros de tabelas que estão em uma relação um por vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado “vários”

da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta. Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

Uma consulta de exclusão exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para Null.

Importante

Após remover os registros usando uma consulta exclusão, você não poderá desfazer a operação. Se quiser saber quais arquivos foram excluídos, primeiro examine os resultados de uma consulta seleção que use o mesmo critério e então, execute a consulta exclusão. Mantenha os backups de seus dados. Se você excluir os registros errados, poderá recuperá-los a partir dos seus backups.

Exemplo de instrução DELETE

Esse exemplo exclui todos os registros de funcionários cujo título seja Estagiário. Quando a cláusula FROM inclui apenas uma tabela, não é necessário indicar o nome da tabela na instrução DELETE.

DELETE *FROM Funcionários WHERE Título = 'Estagiário';

Esse exemplo exclui todos os registros de funcionários cujo título seja Estagiário e que também tenham um registro na tabela "FolhadePagamento". As tabelas "Funcionários" e "FolhadePagamento" têm uma relação um por um.

DELETE Funcionários.* FROM Funcionários INNER JOIN FolhaDePagamento

ON Funcionários.FuncionárioID = FolhadePagamento.FuncionárioID

WHERE Funcionários.Título = ‘Estagiário’;

Subconsultas SQL

Uma subconsulta é uma instrução SELECT aninhada dentro de uma instrução SELECT, INSERT, DELETE ou UPDATE ou dentro de uma outra subconsulta.

Sintaxe

Você pode usar três formas de sintaxe para criar uma subconsulta:

comparação [ANY | ALL | SOME] (instruçãosql)

expressão [NOT] IN (instruçãosql)

[NOT] EXISTS (instruçãosql)

Uma subconsulta tem as partes abaixo:

Parte Descrição

comparação Uma expressão e um operador de comparação que compara a expressão com o resultado da subconsulta.

expressão Uma expressão para a qual o resultado definido da subconsulta é procurado.

instruçãosqlt Uma instrução SELECT de acordo com as mesmas regras e formato de qualquer outra instrução SELECT. Ela deve estar entre parênteses.

Comentários

Você pode usar uma subconsulta em vez de uma expressão na lista de campo de uma instrução SELECT ou em uma cláusula WHERE ou HAVING. Em uma subconsulta, você usa uma instrução SELECT para fornecer um

conjunto de um ou mais valores específicos para avaliar as expressões das cláusulas WHERE ou HAVING.

Use o predicado ANY ou SOME, que são sinônimos, para recuperar registros na consulta principal que satisfaçam a comparação com quaisquer registros recuperados na subconsulta. O exemplo abaixo retorna todos os produtos cujo preço unitário é maior que o preço de qualquer produto vendido com um desconto de vinte e cinco por cento ou mais:

```
SELECT * FROM Produtos WHERE PreçoUnit > ANY

(SELECT PreçoUnit FROM PedidoDetalhes WHERE Desconto >=
.25);
```

Use o predicado ALL para recuperar apenas os registros na consulta principal que satisfaçam a comparação com todos os registros recuperados na subconsulta. Se você mudou ANY para ALL no exemplo acima, a consulta retornaria apenas os produtos cujo preço unitário fosse maior que o de todos os produtos vendidos com um desconto de vinte e cinco por cento ou mais. Isto é muito mais restritivo.

Use o predicado IN para recuperar apenas os registros na consulta principal para os quais alguns registros na subconsulta contêm um valor igual. O exemplo abaixo retorna todos os produtos com um desconto de 25 por cento ou mais:

```
SELECT * FROM Produtos WHERE ProdutoID IN

(SELECT ProdutoID FROM PedidoDetalhes WHERE Desconto >=
.25);
```

De maneira contrária, você pode usar NOT IN para recuperar apenas os registros na consulta principal para os quais não existam registros com valores iguais na subconsulta. Utilize o predicado EXISTS (com a palavra reservada NOT opcionalmente) em comparações true/false para determinar se a subconsulta retorna algum registro.

Você também pode usar aliases de nomes de tabelas em uma subconsulta para fazer referência a tabelas listadas em uma cláusula FROM fora da subconsulta. O exemplo abaixo retorna os nomes dos funcionários cujos salários sejam iguais ou superiores à média de salários de todos os funcionários na mesma função. Para a tabela Funcionários é dada o alias "T1":

SELECT Sobrenome, Nome, Título, Salário FROM Funcionários AS T1

WHERE Salário >= (SELECT Avg(Salário)

FROM Funcionários WHERE T1. T1.Título = Funcionários.Título) Order by Title;

No exemplo acima, a palavra reservada AS é opcional. Algumas subconsultas são aceitas em consultas de tabela cruzada especialmente como predicados (as da cláusula WHERE). Subconsultas como saída (as da lista SELECT) não são aceitas em tabelas de referência cruzada.

Exemplos de subconsultas SQL

Esse exemplo lista o nome, título e salário de todos os representantes de vendas cujos salários sejam superiores aos de todos os gerentes e diretores.

SELECT Sobrenome, Nome, Título, Salário FROM Funcionários

WHERE Título LIKE "*Repr Vendas*" AND Salário > ALL

(SELECT Salário FROM Funcionários WHERE (Título LIKE "*Gerente*")

OR (Título LIKE "*Diretor*"));

Esse exemplo lista o nome e preço unitário de todos os produtos cujo preço unitário seja igual ao do Licor de Cacau.

SELECT NomeProduto, PreçoUnit FROM Produtos

WHERE PreçoUnit = (SELECT PreçoUnit FROM [Produtos]

WHERE NomeProduto = “Licor de Cacau”);

Esse exemplo lista a empresa e o contato de cada empresa de todos os clientes que fizeram pedidos no segundo trimestre de 1995.

SELECT NomeContato, NomeEmpresa, ContatoTítulo, Fone FROM Clientes

WHERE ClienteID IN (SELECT ClienteID FROM Pedidos

WHERE DataPedido BETWEEN #1/04/95# AND #1/07/95#);

Esse exemplo lista os funcionários cujo salário seja maior que a média dos salários de todos os funcionários.

SELECT Sobrenome, Nome, Título, Salário FROM Funcionários T1

WHERE Salário >= (SELECT AVG(Salário) FROM Funcionários

WHERE Funcionários.Título = T1.Título) ORDER BY Título;

Esse exemplo seleciona o nome de todos os funcionários que tenham registrado pelo menos um pedido. Isto também poderia ser feito com INNER JOIN.

SELECT Nome, Sobrenome FROM Funcionários AS E

WHERE EXISTS (SELECT * FROM Pedidos AS O

WHERE O.FuncionárioID = E.FuncionárioID);

Altera o campo Efetuado do arquivo de serviços para 2, caso o mesmo

tenha parecer técnico da entidade encaminhanhamento diferente de nulo.

```
UPDATE servico SET efetuado = 2

WHERE numero_servico = ANY (SELECT servico.numero_servico

FROM servico INNER JOIN encaminhamento

ON (servico.numero_servico = encaminhamento. numero_servico)

AND (servico. ano_servico = encaminhamento.ano_servico)

WHERE (((servico.efetuado) Is Null) AND ((encaminhamento.
parecer_tecnico) Is Not Null))

GROUP BY servico.numero_servico ORDER BY servico.numero_
servico);
```

Minicurrículo

Flávio Ferry de Oliveira Moreira

Possui Graduação em Bacharelado em Ciência da Computação pela Universidade Federal do Piauí (1998) e Mestrado em Ciências da Computação pela Universidade Federal de Pernambuco (2003). Atualmente é professor assistente da Universidade Federal do Piauí. Tem experiência na área de Ciência da Computação, com ênfase em Banco de Dados, atuando principalmente nos seguintes temas: Banco de Dados e DELPHI.

www.uapi.ufpi.br