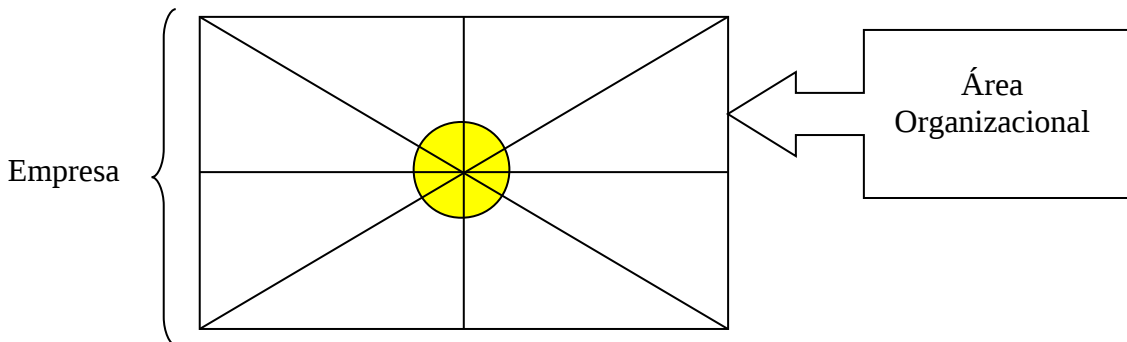

Banco de Dados

1. Visão Geral de Projeto do Banco de Dados:

1.1 **Análise Institucional**

Para definição de um projeto a primeira coisa a ser considerada é a Instituição (Empresa), onde devemos conhecer muito bem sua história, seu crescimento, sua estrutura, seu mercado, sua atuação, etc.



1.2 **Análise de Necessidades**

Levantar junto aos usuários quais suas necessidades para o desenvolvimento de suas atividades diárias, considerando-se a dependência de informações geradas ou tratadas por outras áreas da organização e a exportação de informações para outras áreas ou até mesmo para fora da empresa.

1.3 **Análise das Funcionalidades**

Diante do conhecimento da empresa e das necessidades de seus usuários, podem-se mapear as funcionalidades previstas para atender aos usuários do sistema proposto.

Esse mapa deve ser representado através de diagramas que podem apontar como o sistema tratará seus dados gerando as informações necessárias para seus usuários trabalharem. Os diagramas devem representar a visão macro das atividades a serem tratadas pelo sistema. Para isso temos DFD nível zero, na metodologia estruturada, e o use case (ou caso de uso) na orientação à objetos.

1.4 **Análise de Dados**

Nessa etapa do projeto devemos levantar todos os dados necessários para os usuários trabalharem, tanto os de entrada como os de saída. O tipo de dados, tamanho e estrutura dos dados devem ser bem definidos.

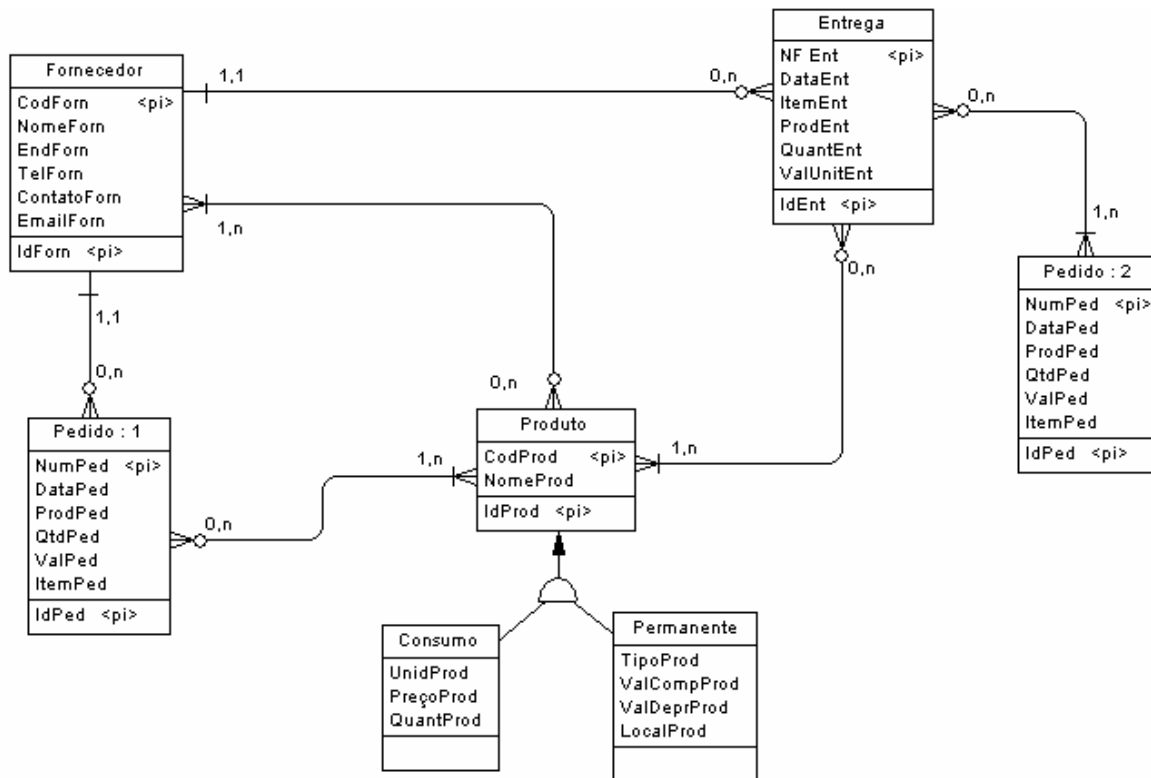
Os dados de entrada no sistema proposto fazem parte dos documentos de digitação, como por exemplo os dados do fornecedor. Todos os campos existentes nos documentos devem ser considerados e devidamente estruturados para serem implementados no banco de dados.

Já os dados de saída são aqueles que fazem parte dos relatórios ou telas de consulta, sendo que deve ser analisada sua origem, pois eles devem vir de um documento de entrada ou de um banco de dados já existente.

Os dados estruturados devem ser padronizados no seu tipo de dados e tamanho, por exemplo: todos as descrições devem ser do tipo caracter de tamanho variável com até 60 bytes.

As chaves de identificação devem ser definidas para cada conjunto de dados referentes a um mesmo assunto, como exemplo tem a chave de identificação para o conjunto de dados do cadastro de fornecedores o código de identificação interno, código criado, e/ou o externo, CNPJ.

O produto da análise de dados é o MER (Modelo de Entidade e Relacionamento), onde o conjunto de dados estruturados formará as Entidades e o Relacionamento será a ligação entre as Entidades quando existir referencia entre elas, isto é, um cliente não se relaciona com fornecedor, mas se relaciona com o contas a receber.



2. Etapas de construção do MER:

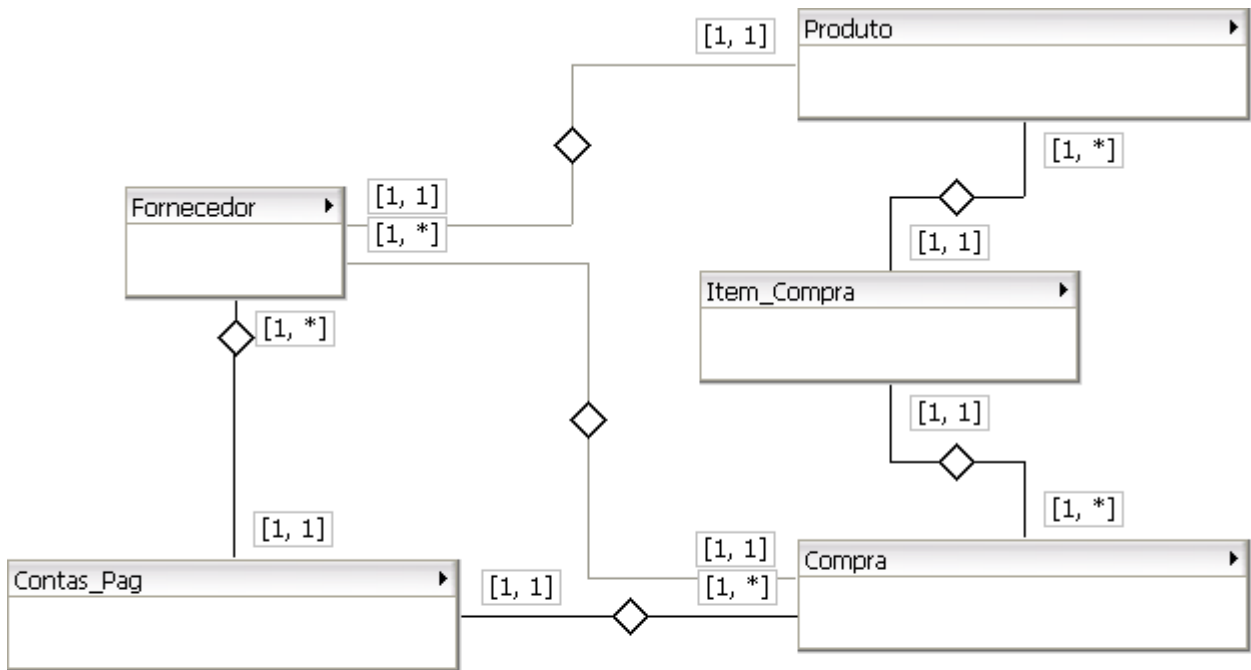
2.1 MER Conceitual:

É a representação do conceito da guarda dos dados representado no modelo por Entidades e o Relacionamento entre elas, isto é: quando é levantado a necessidade da guarda dos dados dos clientes e das compras feitas por eles, deve-se representar as entidades CLIENTE e VENDA, sem que haja a preocupação com os dados necessários para essas entidade.

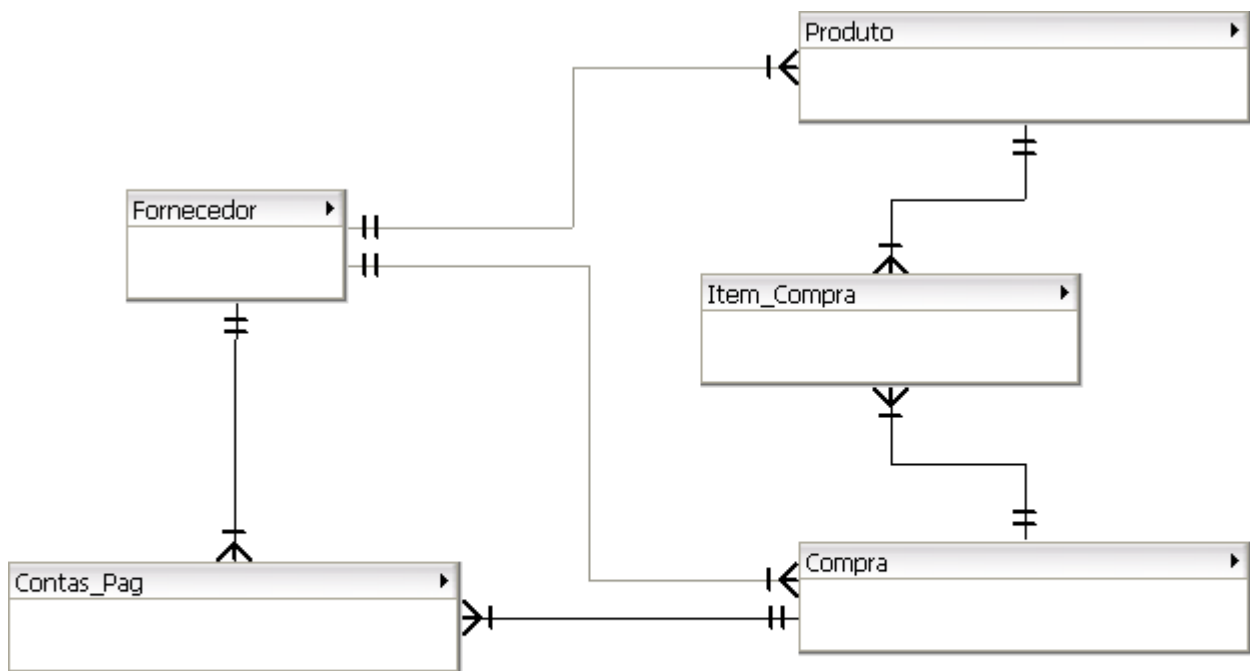
O relacionamento a ser definido deve ter a cardinalidade indicando a ligação entre as entidades, podendo ter os seguintes valores: 1 para 1, 1 para N e M para N, sendo que para essa definição deve-se identificar quantas ocorrências podem ter no par para cada ocorrência da outra entidade e a identificação deve ser feita para os dois pares de entidades. Por exemplo: para cada ocorrência em CLIENTE pode-se ter N ocorrências em VENDA e para cada ocorrência em VENDA tem-se somente uma ocorrência em CLIENTE, isso é o mesmo que dizer que cada cliente participa de N vendas e cada venda é feita por somente 1 cliente.

O relacionamento deve ser representado com usa cardinalidade utilizando-se a notação do Peter Chen ou do James Martin, conforme representação abaixo:

Notação Peter Chen:



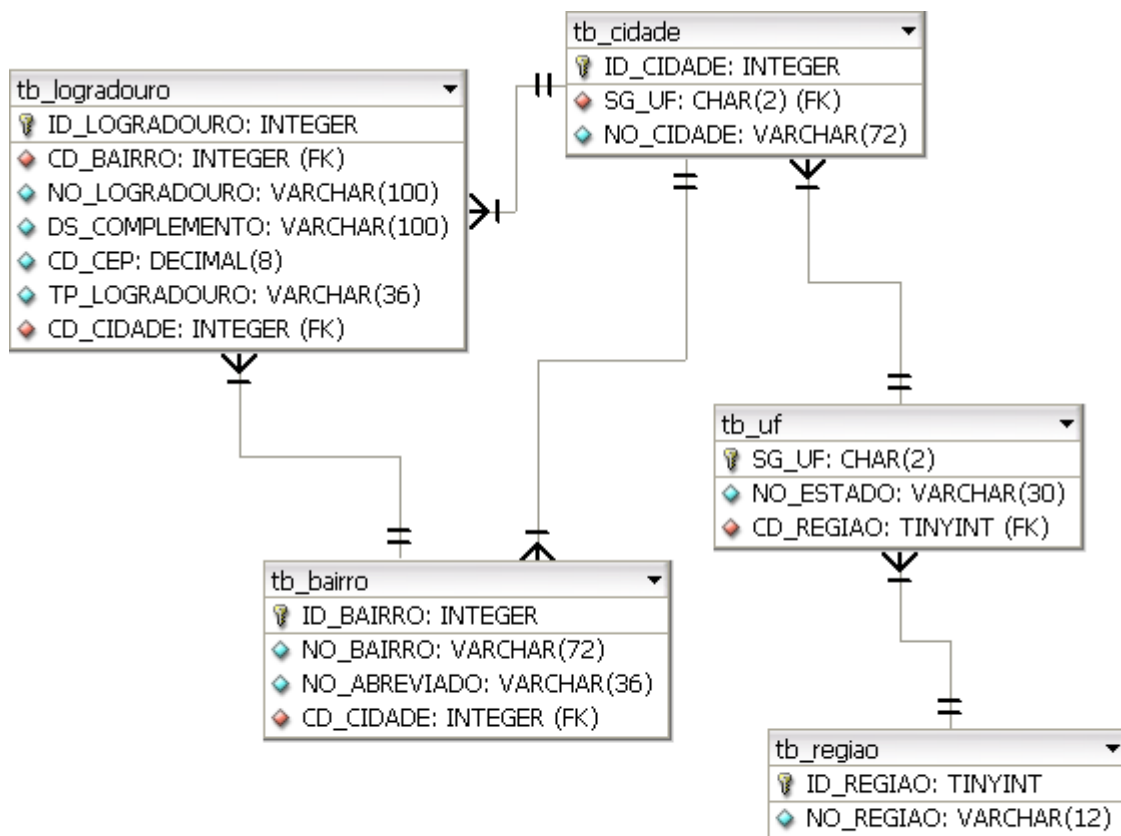
Notação James Martin:



2.2 MER Lógico:

O MER Lógico é uma evolução do MER Conceitual, e nada mais é do que a definição dos atributos que devem compor cada entidade. Os atributos devem ser definidos com as seguintes características:

- ✓ Nome;
- ✓ Tipo de dados;
- ✓ Tamanho;
- ✓ Casas decimais;
- ✓ Indicação de obrigatoriedade;
- ✓ Indicação de atributo identificador;
- ✓ Valor default;
- ✓ Se deve ser considerado o sinal positivo e negativo;
- ✓ Valores aceitos;
- ✓ Se os zeros não significativos serão considerados.



2.3 Normalização:

O MER Lógico, em um segundo momento, deve ser Normalizado nas suas principais Formas Normais e todos os relacionamentos do tipo 1:N e N:M devem aparecer.

O MER Lógico pode ser utilizado como ponto de partida para a aplicação do processo de normalização. Ele pode sintetizar graficamente, um conjunto de relações normalizadas em 3FN. O emprego conjunto dessas duas ferramentas é próprio do método de análise de dados.

- ✓ Cada entidade do modelo, com seus respectivos atributos será submetida às regras de normalização, com exceção daquelas consideradas meramente conceituais.
- ✓ A técnica de normalização permite o refinamento do MER, portanto, após a normalização, uma entidade poderá dar origem a mais de uma tabela.

A Normalização é uma técnica de modelagem de dados que proporciona critérios objetivos para determinar quando uma relação apresenta problemas para ser implementada em bancos de dados relacional. Ela permite a adaptação das estruturas de dados oriundas de modelos conceituais à realidade tabular dos bancos relacionais. Qualquer um que projete um sistema a ser implementado em banco de dados relacional deve estar familiarizado com as técnicas básicas da normalização, o que não significa que o projeto seja baseado somente nos princípios da normalização.

O processo de normalização deu origem a três formas normais: primeira (1FN), segunda (2FN) e terceira (3FN) que em um projeto de banco de dados devem ser aplicadas na sua totalidade.

As formas normais são aplicadas para evitar redundância de dados, inconsistências e anomalias de atualização. Isso porque, redundância de dados é um fato gerador de inconsistências. Já as anomalias de atualização criam dificuldades operacionais para manutenção do banco de dados, quebrando a confiabilidade no dado ou ferindo a integridade dos dados.

1ª Forma Normal (1FN):

Uma entidade está na primeira forma normal se todos os seus atributos são simples, ou seja, contem valores atômicos. Para colocarmos uma relação na 1FN devemos eliminar de sua estrutura os atributos não atômicos (matriz, itens de grupo vetores), de modo que, cada célula da tabela possua apenas um valor de atributo.

Mesmo que uma entidade esteja na 1FN ela pode apresentar redundâncias e anomalias de atualização. Para eliminar ou minimizar estes problemas, devemos prosseguir no processo de normalização, decompondo a entidade em tantas entidades quantas forem os grupos repetitivos.

2ª Forma Normal (2FN):

Uma entidade está na 2FN se, e somente se, ela estiver em 1FN e todos os seus atributos não chave forem dependentes completa e exclusivamente dela, ou seja, não contenha atributos que dependente funcionalmente de subconjuntos da chave primária composta.

Para passarmos uma entidade da 1FN para a 2FN, devemos eliminar as dependências parciais. Para tanto são geradas novas entidades contendo colunas que se encontram em dependência parcial com a chave primária

3ª Forma Normal (3FN):

Uma entidade está na 3FN se, e somente se, estiver na 2FN e todos os seus atributos dependerem completa e exclusivamente da chave primária, ou seja, não pode haver relação de identificação entre os atributos que não façam parte da chave primária (dependência transitiva).

Para passarmos uma relação da 2FN para a 3FN, devemos eliminar as dependências transitivas, gerando uma nova tabela para cada uma delas.

3. Passagem do MER para o Modelo do Banco de Dados:

Com o MER Normalizado o projeto do banco de dados está pronto para passar da sua fase lógica, Projeto Lógico, para a fase física, Projeto Físico ou de Implementação ou, simplesmente e mais correto, Modelo do Banco de Dados.

O MER é a representação do modelo lógico do projeto de banco de dados, sendo que todas as regras de criação de representação dos conceitos dos dados necessários ao projeto devem ser consideradas e o Modelo do Banco de Dados tem sua representação física, deixando de existir Entidades/Relacionamentos e passando a existir Tabelas, Índices, Stored Procedures, etc.

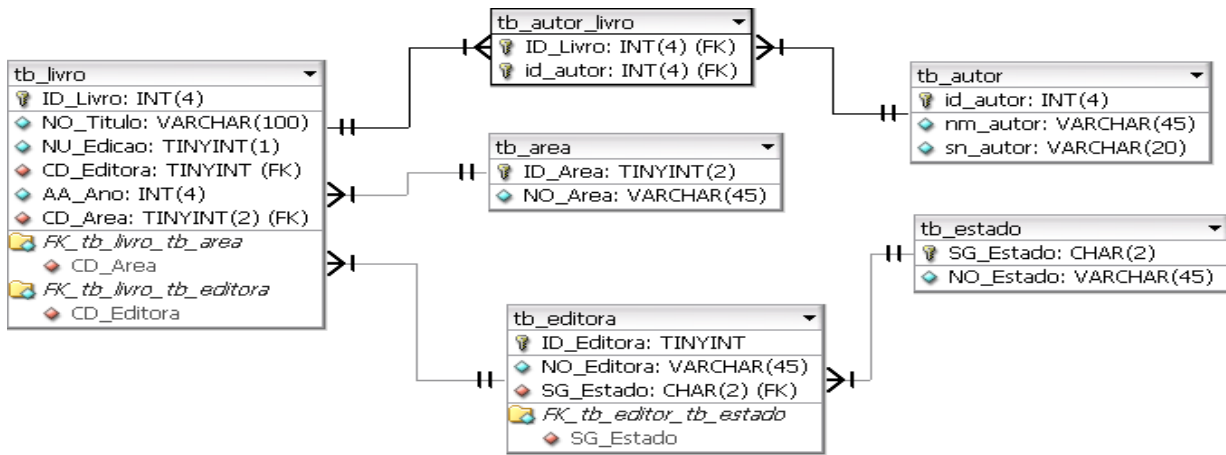
MER	Modelo de Dados
Entidade	Tabela
Atributos	Colunas
Identificador	Chave Primária
Relacionamento	Referencia ou Dependência
Cardinalidade	Chave Primária (PK) e Chave Estrangeira (FK)
	Índice
	Stored Procedure
	Trigger
	Constraint (check)
	User
	View

O MER, na sua evolução do MER Lógico ao MER Normalizado, não depende da tecnologia de banco de dados a ser utilizada pela empresa, já o Modelo do Banco de Dados depende, posto que cada SGBD tem suas características próprias.

O relacionamento 1:1 deve ser evitado, pois os dados das duas tabelas se completam, a não ser no caso de uma especialização (super tipo) onde os dados de uma tabela podem ser completados pelos dados de outras tabelas com características diferenciadas, conforme o caso, ou quando um determinado dado ocupa muito espaço e é de pouco uso, como é o caso de fotos.

A relação existente entre duas tabelas é feita através da definição da chave estrangeira (FK) na tabela dependente referenciado a chave primária (PK) na tabela principal, isto é, na tabela dependente do relacionamento deve-se definir a chave estrangeira (FK), sendo que esta deve ter a mesma estrutura física da PK da tabela principal, já que a referência é feita através da FK à PK. Em outras palavras podemos dizer que a ligação física entre duas tabelas utiliza a filosofia de Pai e Filho, onde o Filho (FK) depende do Pai (PK) e um Pai (PK) pode ter "N" Filhos (FK) mas um Filho (FK) só pode ter um Pai (PK). Com isso temos uma relação de 1:N fisicamente.

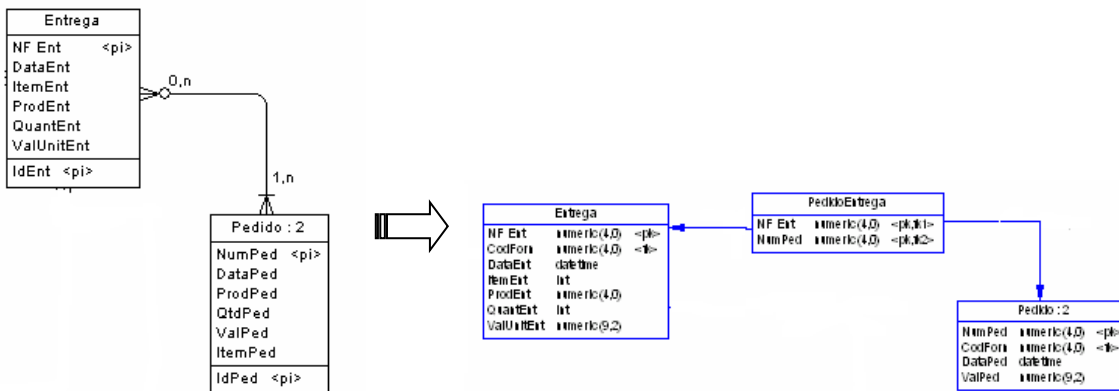
Fisicamente essa relação ocorre através das ocorrências (ou linhas) existentes nas tabelas relacionadas, onde é correto dizer que para cada ocorrência em uma tabela ter um número X de ocorrências na outra tabela com a mesma chave de casamento (valor da PK = valor da FK).



Também esta correto dizer que um Pai (PK) pode não ter Filhos (FK) e no modelo de dados isso significa que uma ocorrência na tabela Pai pode não ter ocorrências referente a sua ligação na tabela Filho (FK).

Uma tabela pode fazer referencia a várias tabelas e em algum momento uma de suas ocorrências pode não ter relação com uma das tabelas, o que faz com que o valor da coluna FK, dessa relação, fique NULL (valor nulo). Isso quer dizer que uma FK pode ter valor nulo, porém, só lembrando, uma PK não pode ter valor nulo.

O relacionamento M:N é o mesmo que dizer que um Pai pode ter “N” Filhos e um Filho pode ter “N” Pais e se olharmos pelo lado do relacionamento “M” ou “N” verifica-se que nessa relação só existem Filhos, o que não é permitido em um banco de dados. Para resolver este problema, deve-se criar uma tabela para quebrar esse relacionamento, tabela esta que é chamada de “Associativa”.



O Modelo de Dados de Implementação é aquele que dará condições plenas para o DBA criar o DataBase que atenderá o sistema proposto. Com isso todas as regras de integridade devem estar definidas, assim como viws necessárias e índices para agilizar a disponibilização dos dados, etc.

3.1 Os seguintes aspectos serão observados na validação de modelos de dados:

- ✓ Escopo do sistema;
- ✓ Adequação das entidades ao escopo do sistema;
- ✓ Possibilidade de integração e compartilhamento de dados com outros sistemas;
- ✓ Possibilidade de utilização de tabela corporativa do ambiente;
- ✓ Regras de Integridade expressas no modelo;
- ✓ Relacionamentos e cardinalidades;
- ✓ Uso de especializações e agregações;
- ✓ Normalização das entidades até a 3FN;
- ✓ Redundância de dados;
- ✓ Chaves primárias e chaves estrangeiras;
- ✓ Tamanho e tipo dos atributos;
- ✓ Descrição de validades do domínio de um atributo;
- ✓ Utilização de domínios de uso comum (corporativos);
- ✓ Nomenclatura dos elementos do modelo (conforme padrão).

3.2 Tabelas Estatísticas e com Totalizadores

São tabelas que armazenam totais ou estatísticas resultantes do processamento de dados analíticos.
Ex: Percentual de Carteiras emitidas em uma Unidade Organizacional, quantidade de acessos a um site do MTB

Critérios para tratamento:

- Quando os totalizadores forem resultantes de processamento de dados analíticos que são excluídos da base depois de totalizados.
- Quando o processamento da totalização em tempo real comprometer o tempo de resposta da aplicação.
Ex: Saldos de contas com extenso movimento de lançamentos

3.3 Critérios para Limpeza de Tabelas

Para as tabelas voláteis que compõem o banco de dados do sistema, devem ser definidos critérios para exclusão periódica dos dados, a fim de evitar o crescimento indefinido da tabela. Os critérios são periodicidade e parâmetros de seleção de registros para exclusão.

4. Tipos de Dados:

Os tipos de dados variam conforme o SGBD, apresentando diferenças no tamanho e valor máximo.

Tipo de Dado	Descrição	Observação
TINYINT	Valores numéricos inteiros variando de 0 até 256	
SMALLINT	Valores numéricos inteiros variando de – 32.768 até 32.767	
INT	Valores numéricos inteiros variando de -2.147.483.648 até 2.147.483.647	
BIGINT	Valores numéricos inteiros variando de –92.23.372.036.854.775.808 até 9.223.372.036.854.775.807	

BIT	Somente pode assumir os valores 0 ou 1. Utilizado para armazenar valores lógicos.	
DECIMAL(I,D) e NUMERIC(I,D)	Armazenam valores numéricos inteiros com casas decimais utilizando precisão. I deve ser substituído pela quantidade de dígitos total do número e D deve ser substituído pela quantidade de dígitos da parte decimal (após a vírgula). DECIMAL e NUMERIC possuem a mesma funcionalidade, porém DECIMAL faz parte do padrão ANSI e NUMERIC é mantido por compatibilidade. Por exemplo, DECIMAL(8,2) armazena valores numéricos decimais variando de – 999999,99 até 999999,99	Lembrando sempre que internamente armazena o separador decimal como ponto (.) e o separador de milhar como vírgula (,). Essas configurações INDEPENDEM de como o Windows está configurado no painel de controle e para DECIMAL E NUMERIC, somente o separador decimal (.) é armazenado.
SMALLMONEY	Valores numéricos decimais variando de -214.748,3648 até 214.748,3647	
MONEY	Valores numéricos decimais variando de -22.337.203.685.477,5808 até 922.337.203.685.477,5807	
REAL	Valores numéricos aproximados com precisão de ponto flutuante, indo de -3.40E + 38 até 3.40E + 38	
FLOAT	Valores numéricos aproximados com precisão de ponto flutuante, indo de -1.79E + 308 até 1.79E + 308	
SMALLDATETIME	Armazena hora e data variando de 1 de janeiro de 1900 até 6 de junho de 2079. A precisão de hora é armazenada até os segundos.	
DATETIME	Armazena hora e data variando de 1 de janeiro de 1753 até 31 de Dezembro de 9999. A precisão de hora é armazenada até os centésimos de segundos.	
CHAR(N)	Armazena N caracteres fixos (até 8.000) no formato não Unicode.	Se a quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo é preenchido com espaços em branco.
VARCHAR(N)	Armazena N caracteres (até 8.000) no formato não Unicode.	Se a quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo não é preenchido.

TEXT	Armazena caracteres (até 2.147.483.647) no formato não Unicode.	Se a quantidade de caracteres armazenada no campo for menor que 2.147.483.647, o resto do campo não é preenchido. Procure não utilizar este tipo de dado diretamente, pois existem funções específicas para trabalhar com este tipo de dado.
NCHAR(N)	Armazena N caracteres fixos (até 4.000) no formato Unicode.	Se a quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo é preenchido com espaços em branco.
NVARCHAR(N)	Armazena N caracteres (até 4.000) no formato Unicode.	Se a quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo não é preenchido.
NTEXT	Armazena caracteres (até 1.073.741.823) no formato Unicode.	Se a quantidade de caracteres armazenada no campo for menor que 1.073.741.823, o resto do campo não é preenchido. Procure não utilizar este tipo de dado diretamente, pois existem funções específicas para trabalhar com este tipo de dado.

4.1 Características dos Dados:

Alguns tipos de dados têm características próprias e tratamentos diferenciados, principalmente entre SGBDs. São elas:

- Chave Primária não pode ter valor nulo
- Um dado pode ter seu valor incrementado de forma seqüencial e automática
- Uma coluna pode ter valor padrão (default)
- Os dados do tipo inteiro podem ser sinalizados ou não
- Os dados do tipo decimal são definidos com seu tamanho total e a indicação do número de casas decimais
- Os dados do tipo inteiro podem ter os zeros não significativos (zeros a esquerda) considerados
- Os dados do tipo TIMESTAMP tem seu valor atualizado sempre que a linha do dado for atualizada
- Os dados do tipo ENUM validam valores a serem inseridos
- Os dados do tipo variável têm seu tamanho máximo carregado em memória e seu tamanho de preenchimento gravado em disco

5. Introdução ao SGBD

5.1 O MySQL:

“A MySQL Enterprise subscription is the most comprehensive offering of MySQL database software, services and support; it ensures that your business achieves the highest levels of reliability, security, and uptime.

An Enterprise Subscription includes:

1. The MySQL Enterprise Server – the most reliable, secure, and up-to-date version of the world’s most popular open source database
2. The MySQL Monitoring and Advisory Service – An automated virtual DBA assistant that monitors all your MySQL Servers around-the-clock, identifies exceptions to MySQL best practices, and provides expert advice on fixing any problems discovered
3. MySQL Production Support – Technical and consultative support when you need it, along with regularly scheduled service packs, hot-fixes, and more

For more information, visit <http://www.mysql.com/enterprise>. “

5.2 Forma de armazenamento dos dados:

Um SGBD - Sistema de Gerenciamento de Banco de Dados - é uma coleção de programas, chamados de **objetos**, que permitem ao usuário definir, construir e manipular Bases de Dados para as mais diversas finalidades.

Esses objetos são: Tabelas, Índices, Chaves (Key), Views, Stored Procedures, Triggers, Referencias (Relacionamentos) e Users.

Os dados são armazenados pelos SGBD da seguinte forma:

- Dados: Local onde os dados efetivos ficam armazenados.
- Log: Local onde os dados ficam armazenados temporariamente, até sua efetivação. A efetivação dos dados é concretizada através do comando Commit, que efetiva os dados da transação em andamento para a área de dados.
- Índice: Local onde os dados que indexam uma tabela ficam guardados. Os usuários do SGBD não têm acesso à esses arquivos.

Os índices podem ser do tipo:

- Índice normal
- Índice único
- Full text index
- No formato sequencial
- Em forma de árvore binária

5.3 Forma de manipulação dos dados:

As características de um SGBD são:

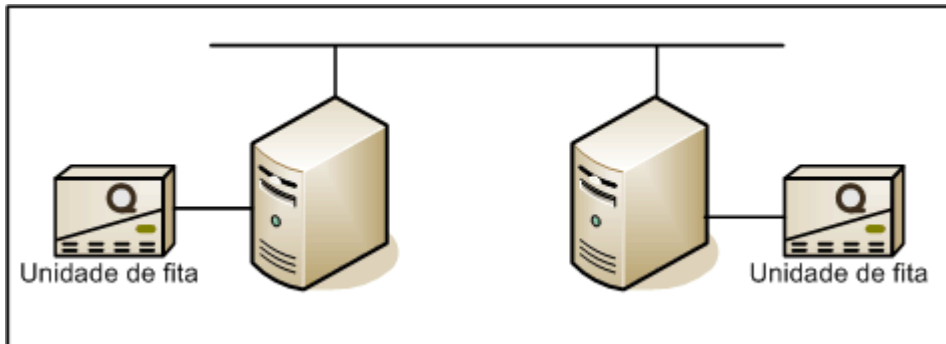
- ✓ Controle de Redundâncias
 - A redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Em um Banco de Dados as informações só se encontram armazenadas em um único local, não existindo duplicação descontrolada dos dados. Quando existem replicações dos dados, estas são decorrentes do processo de armazenagem típica do ambiente Cliente-Servidor, totalmente sob controle do Banco de Dados
- ✓ Compartilhamento dos Dados
 - O SGBD deve administrar o controle de concorrência ao acesso dos dados, garantindo em qualquer tipo de situação a escrita/leitura de dados sem erros
- ✓ Controle de Acesso
 - O SGBD deve dispor de recursos que possibilitem selecionar a autoridade de cada usuário. Assim um usuário poderá realizar qualquer tipo de acesso, outros poderão ler alguns dados e atualizar outros e outros ainda poderão somente acessar um conjunto restrito de dados para escrita e leitura
- ✓ Interfaceamento
 - Um Banco de Dados deverá disponibilizar formas de acesso gráfico, em linguagem natural, em SQL ou ainda via menus de acesso, não sendo uma "caixa-preta" somente sendo passível de ser acessada por aplicações
 - Esquematização
 - Um Banco de Dados deverá fornecer mecanismos que possibilitem a compreensão do relacionamento existente entre tabelas e de sua eventual manutenção
- ✓ Controle de Integridade
 - Um Banco de Dados deverá impedir que aplicações ou acessos pelas interfaces possam comprometer a integridade dos dados.
- ✓ Backup
 - O SGBD deverá apresentar facilidade para recuperar falhas de hardware e software, através da existência de arquivos de "pré-imagem" ou de outros recursos automáticos, exigindo minimizar a intervenção de pessoal técnico

Os SGBD utilizam DataBases e tabelas para manter a administração dos DataBases criados pelos usuários.

Os dados lidos são carregados em memória em formato de Página de 64KB, sendo que seus valores são carregados de forma seqüencial extraídos dos índices ou área de dados/log..

5.3.1 Rotina de Backup/Restore

Os SGBD estão munidos de utilitário para execução de backup e restore, sendo que alguns só fazem para o database completo.



Existem tipos diferenciados de backup, são eles:

- FULL – backup completo do database
- Diferencial – backup somente com as diferenças existentes no database referente ao ultimo backup feito
- Transaction Log – backup da área de log de transações

Backup deve ser feito **diariamente**, onde deve ficar uma cópia no disco do servidor (preferencialmente em disco diferente ao do database), uma em mídia removível na sala dos servidores e outra mídia removível em local distante e seguro.

Se a quantidade de dados da empresa for em número pequeno o backup full deve ser feito diariamente, mas ser for grande deve ser feito um backup full uma vez por semana e diariamente um diferencial.

Dependendo da criticidade dos dados da empresa o backup de log de transações deve ser feito em espaço curto de tempo (por exemplo de hora em hora).

A rotina de restauração dos dados deve ser escrita conforme a rotina de backup, isto é: se o backup for full diariamente a restauração deve ser do ultimo backup, mas se for diferencial deve-se baixar o backup full e depois baixar cada backup diferencial na ordem em que foram tirados. No caso do backup de log a rotina de baixa segue a mesma lógica da baixa de backup diferencial.

Existem softwares no mercado que tem como finalidade efetuar backup da forma que o usuário deseja.

5.3.1.1 Noções básicas sobre backup



Existem muitas maneiras de perder informações em um computador involuntariamente. Uma criança usando o teclado como se fosse um piano, uma queda de energia, um relâmpago, inundações. E algumas vezes o equipamento simplesmente falha.

Se você costuma fazer cópias de backup dos seus arquivos regularmente e os mantém em um local separado, você pode obter uma parte ou até todas as informações de volta caso algo aconteça aos originais no computador.

A decisão sobre quais arquivos incluir no backup é muito pessoal. Tudo aquilo que não pode ser substituído facilmente deve estar no topo da sua lista. Antes de começar, faça uma lista de verificação de todos os arquivos a serem incluídos no backup. Isso o ajudará a determinar o que precisa de backup, além de servir de lista de referência para recuperar um arquivo de backup-. Eis algumas sugestões para ajudá-lo a começar:

- Dados bancários e outras informações financeiras
- Fotografias digitais
- Software comprado e baixado através da Internet
- Música comprada e baixada através da Internet
- Projetos pessoais
- Seu catálogo de endereços de e-mail
- Seu calendário do Microsoft Outlook
- Seus favoritos do Internet Explorer

5.3.1.2 Design de serviços de backup e recuperação

As organizações dependem de dados críticos para a operação com êxito de seus negócios. As tecnologias de backup e recuperação são a base das estratégias de proteção de dados que ajudam as organizações a atender aos seus requisitos de disponibilidade e acessibilidade de dados. Os data centers podem usar componentes redundantes e tecnologias de tolerância a falhas (como clusters de servidor e espelhamento de software ou hardware) para garantir a alta disponibilidade replicando dados cruciais. No entanto, apenas essas tecnologias não podem resolver os problemas causados por corrupção ou exclusão de dados, que podem ocorrer devido a erros em aplicativos, vírus, violações de segurança ou erros do usuário. Você também pode precisar manter informações em formato arquivado, por motivos de auditoria industrial ou legal. Esse requisito pode abranger dados transacionais, documentos e informações de colaboração, como e-mail. Você deve, portanto, ter uma estratégia de proteção de dados que inclua um esquema abrangente de backup e recuperação para proteger os dados contra qualquer tipo de pane ou desastre não previsto e que atenda a qualquer requisito industrial relevante para a retenção de dados.

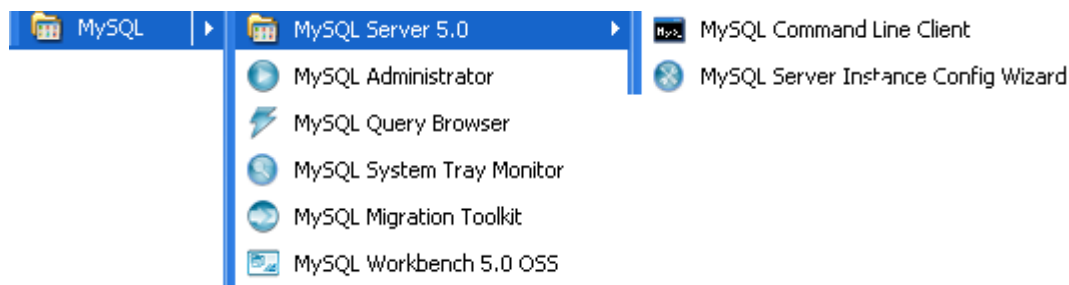
5.4 Linguagem SQL:

Para trabalhar os Databases e seus dados os SGBD utilizam a linguagem SQL (Structured Query Language) com as seguintes características:

- Ela segue o padrão ANSI (American National Standards Institute) e ISO (International Standards Organization);
 - Tais padrões têm como objetivo projetar uma versão padrão da linguagem para todos os SGBDs;
- O 1º padrão da linguagem SQL foi lançada em 1986 pela IBM;
- A linguagem SQL contém três sub-linguagens:
 - Uma linguagem de definição de dados (DDL):
 - Usada para descrever o esquema das tabelas do banco de dados;
 - Utiliza instruções do tipo: CREATE objeto, ALTER objeto e DROP objeto;
 - Não necessita COMMIT;
 - Uma de manipulação de dados (DML):
 - Abrange todas as declarações que manipulam dados;
 - Utiliza operações genéricas para manipular dados: recuperação (SELECT), inserção (INSERT), exclusão (DELETE) e modificação (UPDATE);
 - Necessita COMMIT;
 - E outra de controle (DCL):
 - Abrange as declarações que controlam a autorização de uso dos dados;
 - Utiliza instruções do tipo: GRANT e REVOKE

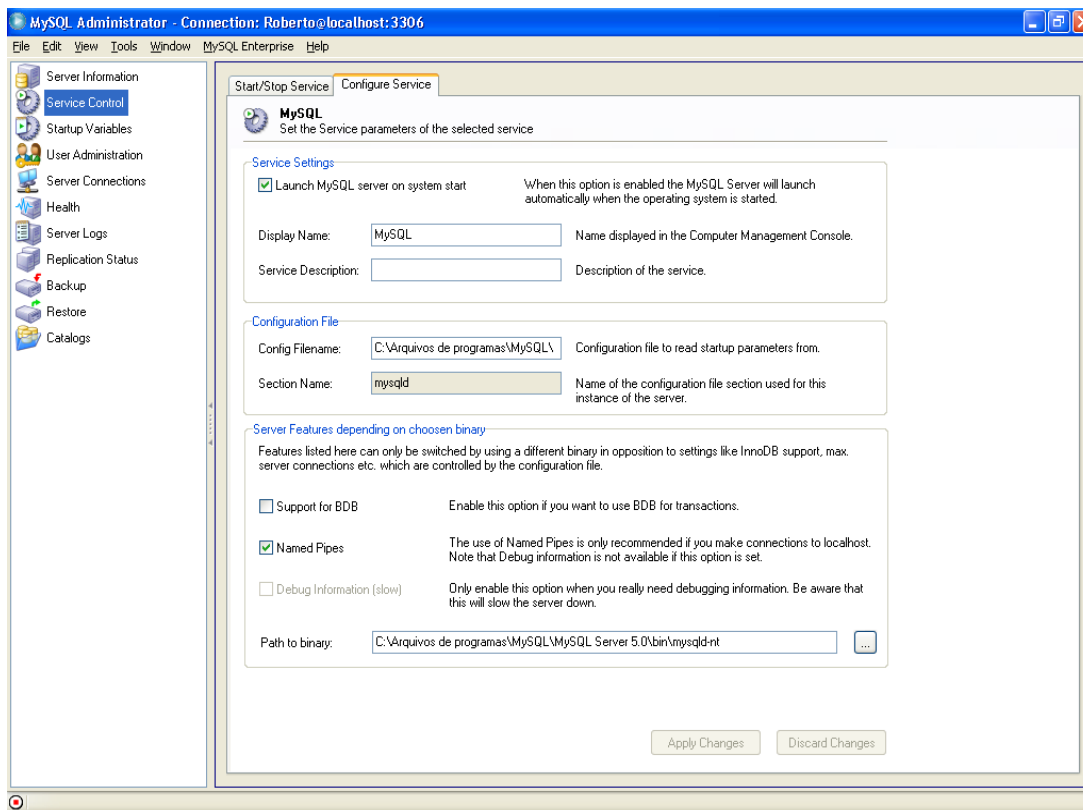
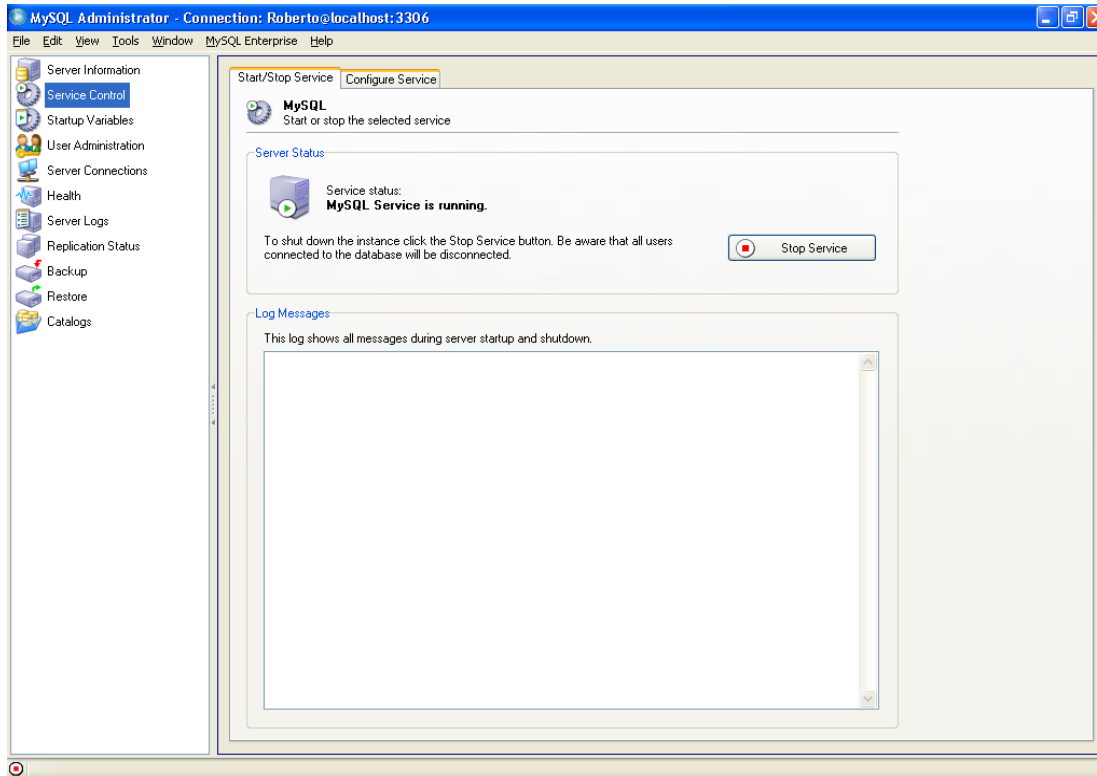
6. Apresentação do SGBD

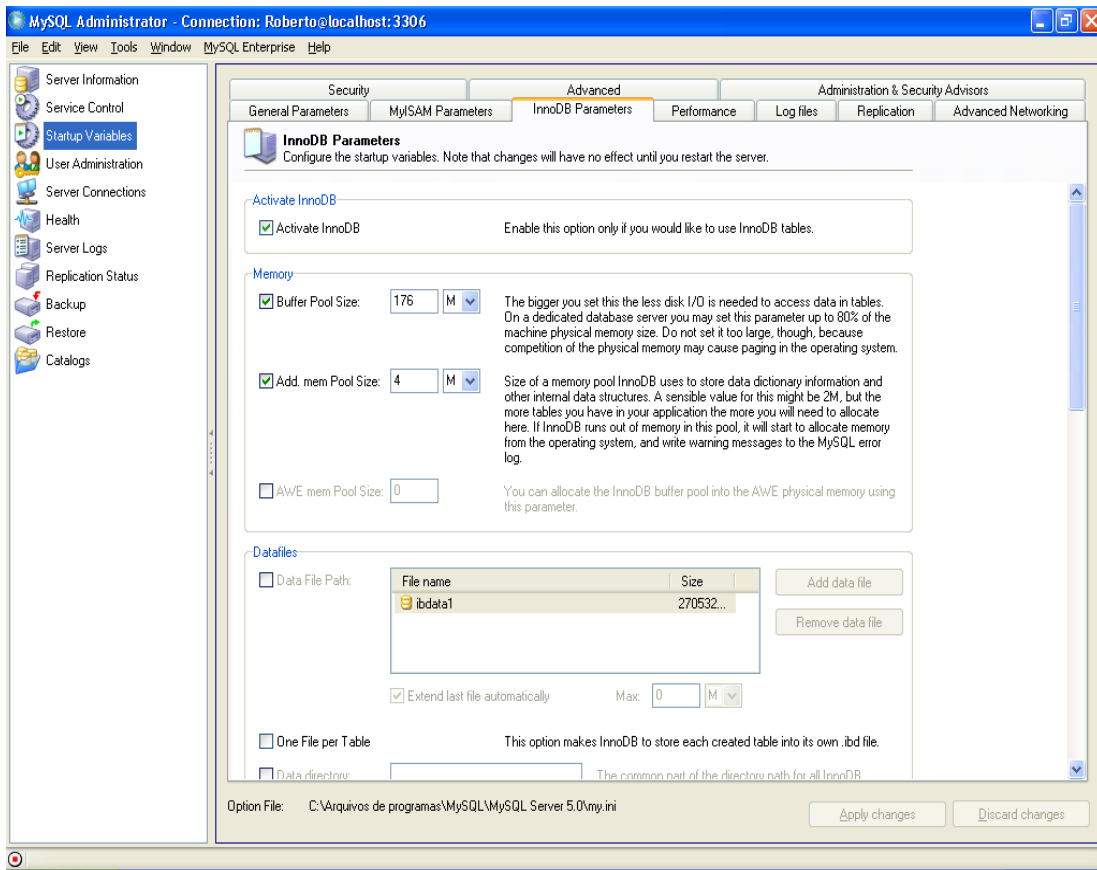
6.1 Utilitários do MySQL:



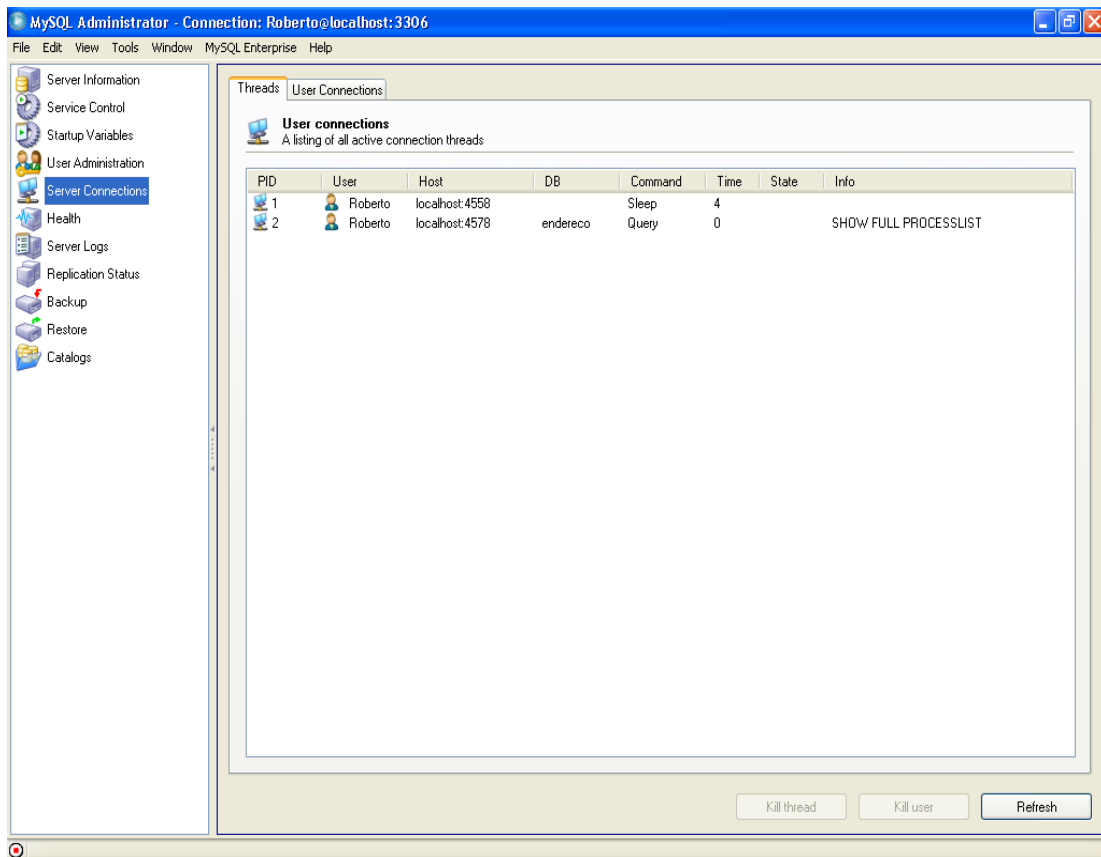
6.2 Uso do SGBD pelo DBA:

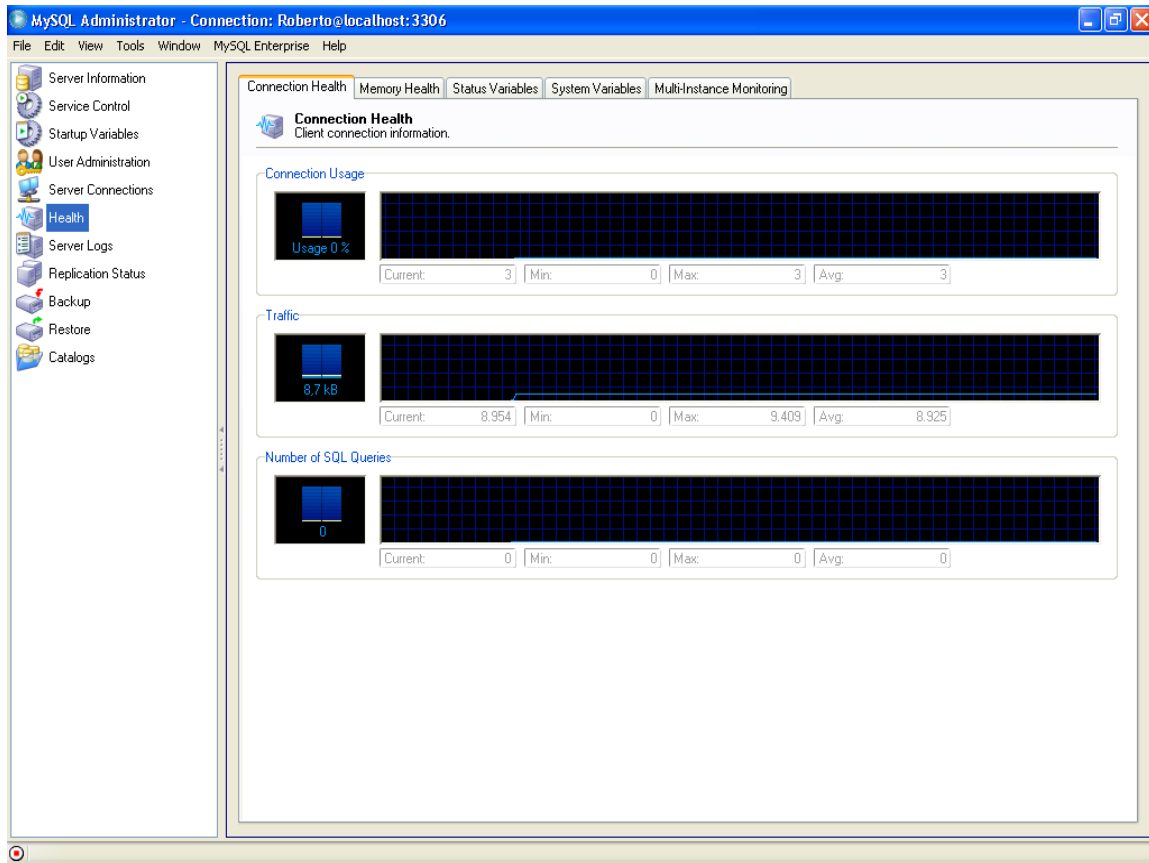
6.2.1 Configurações do SGBD:



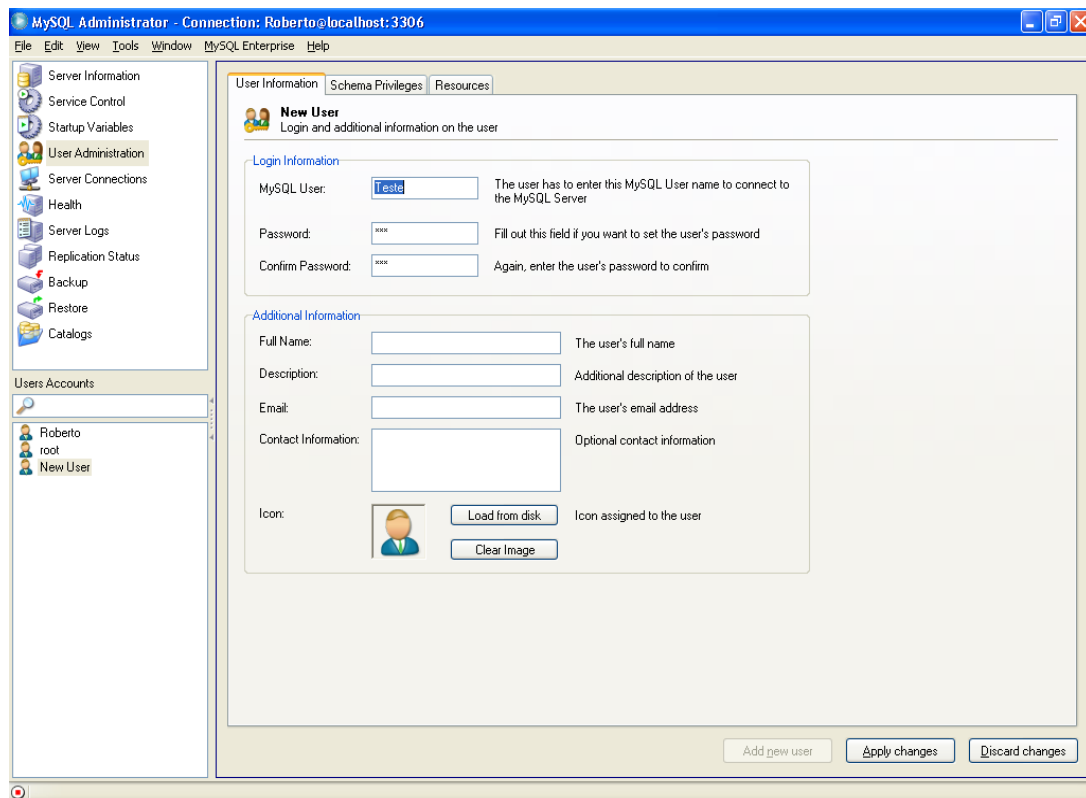


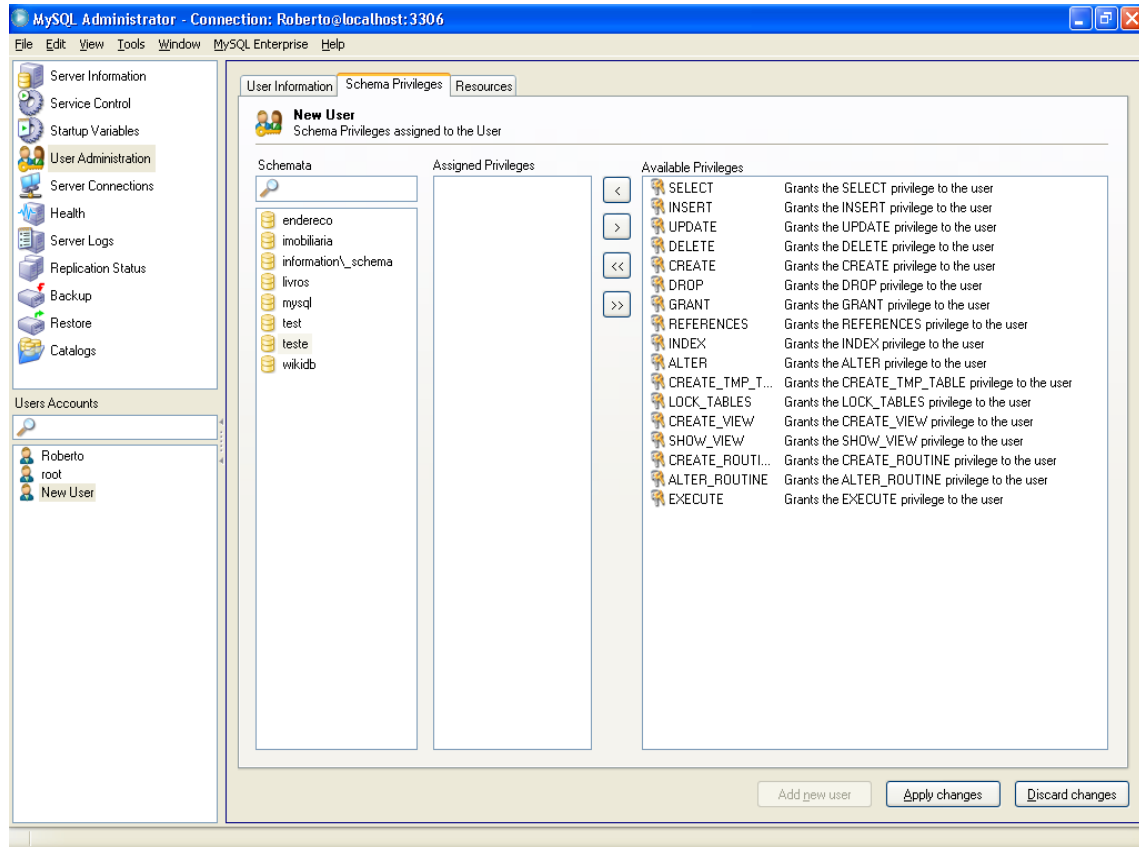
6.2.2 Administração de Uso do SGBD:



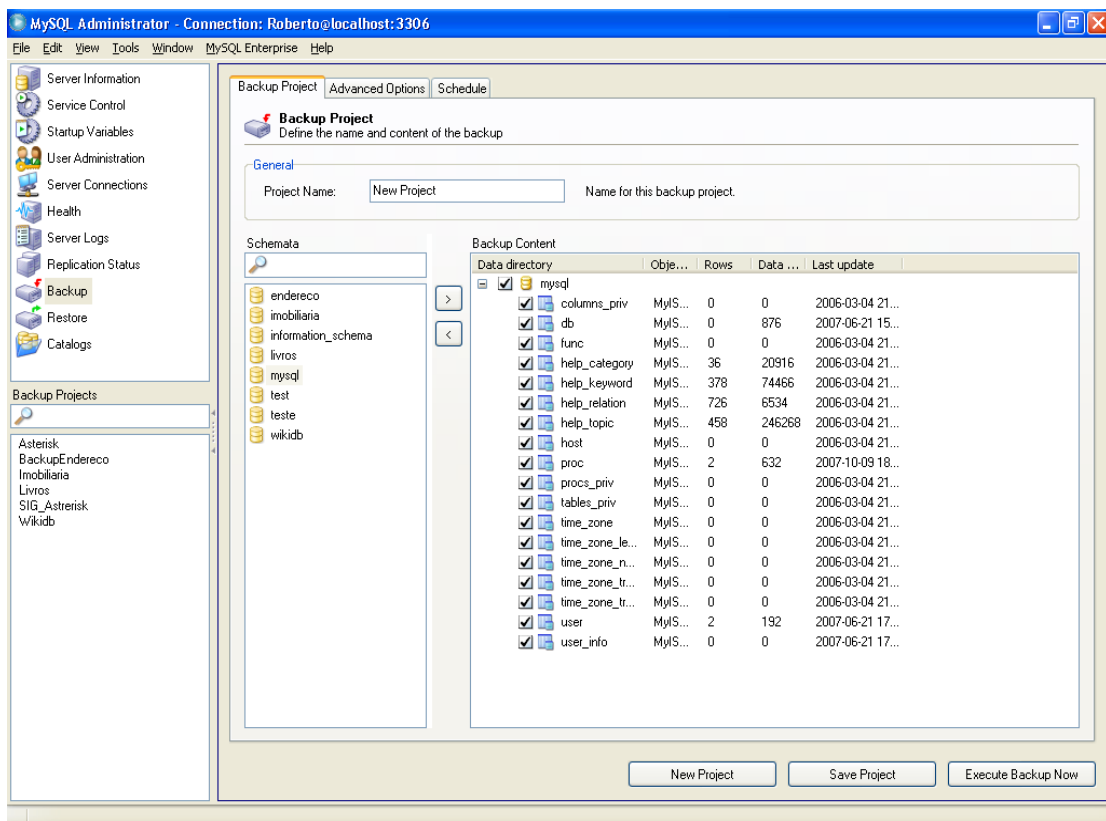


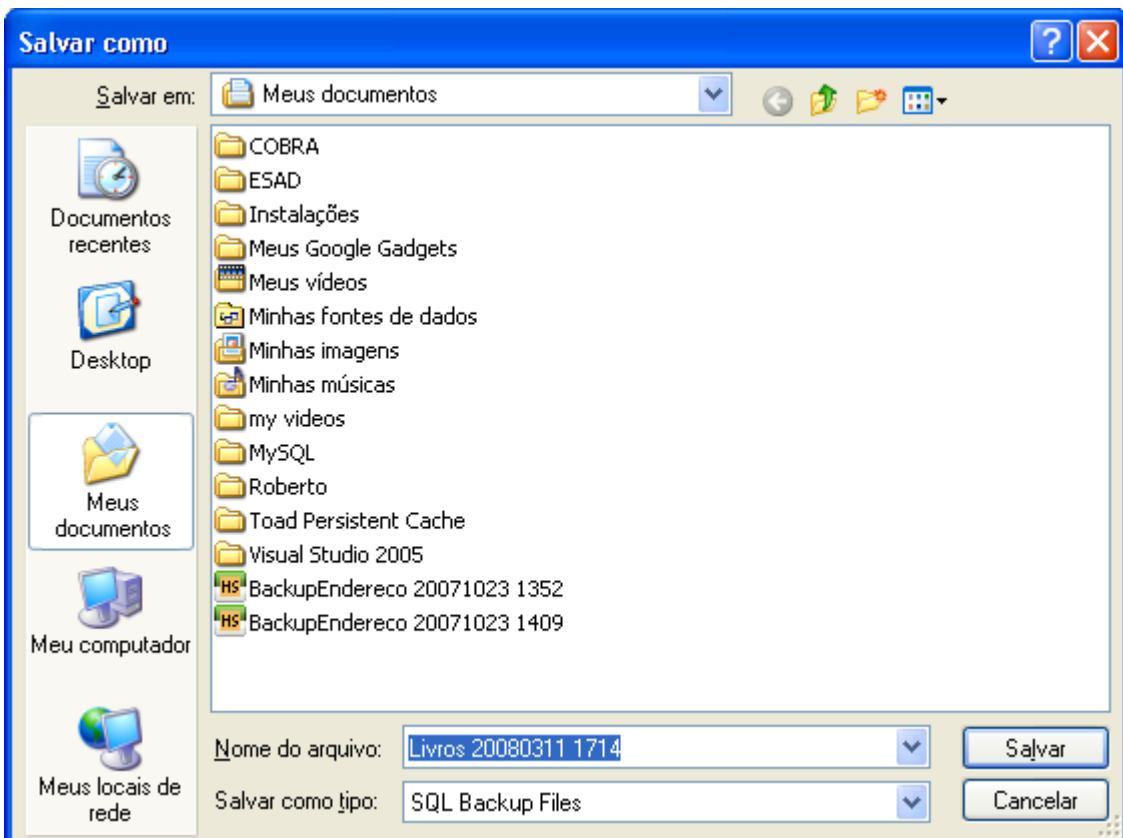
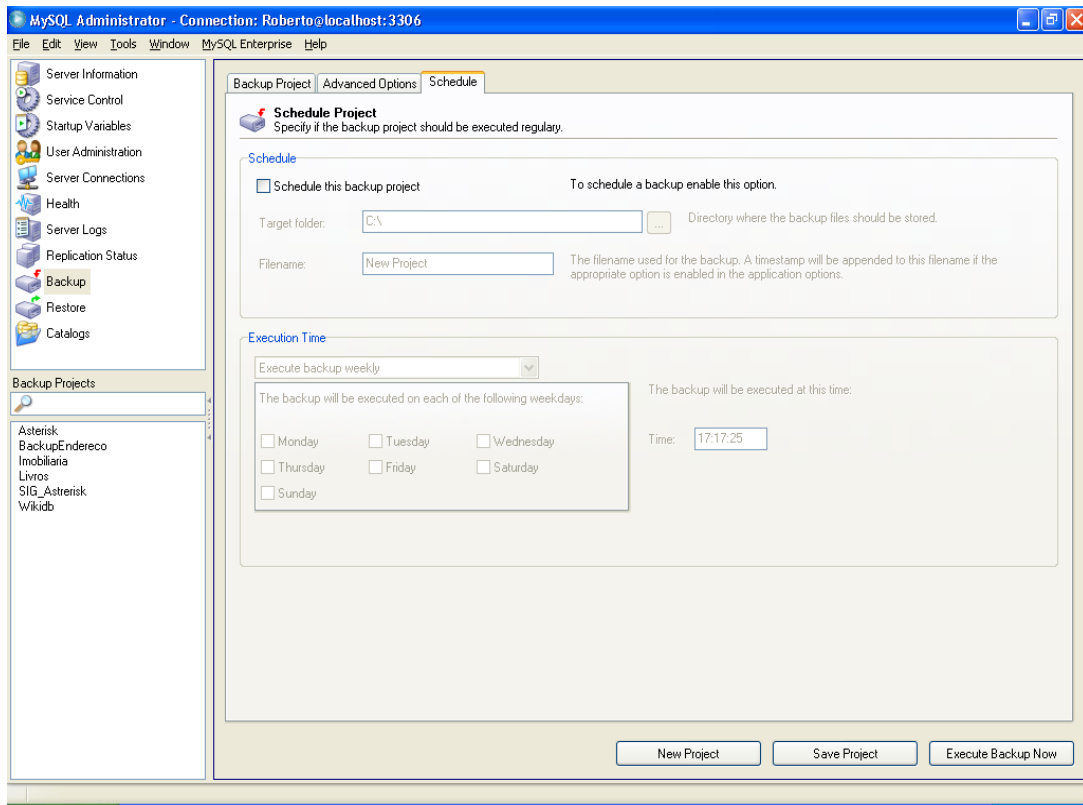
6.2.3 Administração de Usuários:



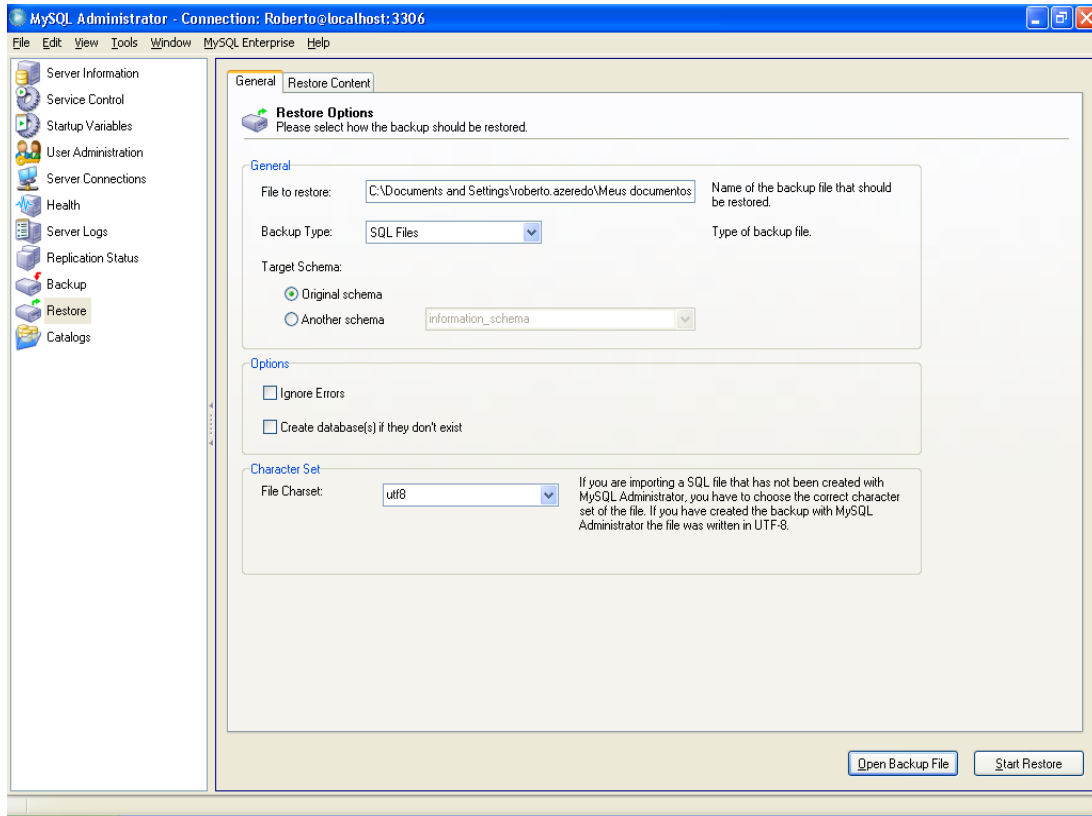


6.2.4 Rotina de Backup:





6.2.4.1 Restauração de Backup:

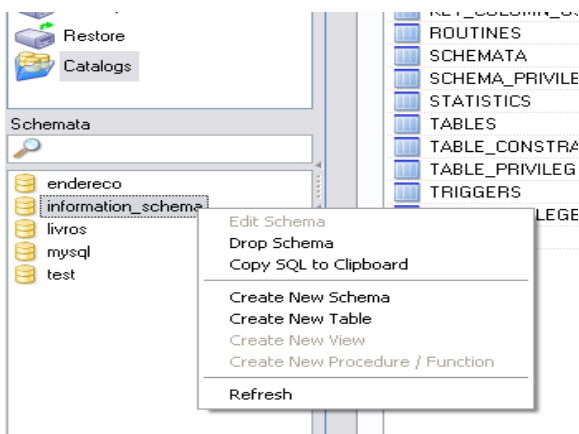


7. Criação e manutenção do banco de dados

7.1 Sintaxe dos comandos da sub-linguagem DDL:

7.1.1 Criação de Database/Schema

```
CREATE DATABASE [IF NOT EXISTS] db_name
  [create_specification [, create_specification] ...]
create_specification:
  [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```



7.1.2 Criação de Tabela

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
  [( LIKE old_tbl_name )];
create_definition:
  column_definition
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
| KEY [index_name] [index_type] (index_col_name,...)
| INDEX [index_name] [index_type] (index_col_name,...)
| [CONSTRAINT [symbol]] UNIQUE [INDEX]
  [index_name] [index_type] (index_col_name,...)
| [FULLTEXT|SPATIAL] [INDEX] [index_name] (index_col_name,...)
| [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (index_col_name,...) [reference_definition]
| CHECK (expr)
column_definition:
  col_name type [NOT NULL | NULL] [DEFAULT default_value]
  [AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string']
  [reference_definition]
```

7.1.3 Criação de Índice

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name [index_type]
  ON tbl_name (index_col_name,...)
index_col_name:
  col_name [(length)] [ASC | DESC]
```

7.1.4 Criação de View

```
CREATE VIEW view_name [ ( column [ ,...n ] ) ]
AS
select_statement
```

7.1.5 Alteração de Coluna

```
ALTER [IGNORE] TABLE tbl_name
  alter_specification [, alter_specification] ...

alter_specification:
  ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
| ADD [COLUMN] (column_definition,...)
| ADD INDEX [index_name] [index_type] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
  PRIMARY KEY [index_type] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
  UNIQUE [index_name] [index_type] (index_col_name,...)
| ADD [FULLTEXT|SPATIAL] [index_name] (index_col_name,...)
| ADD [CONSTRAINT [symbol]]
  FOREIGN KEY [index_name] (index_col_name,...)
```

```

[reference_definition]
| ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
| CHANGE [COLUMN] old_col_name column_definition
  [FIRST|AFTER col_name]
| MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP INDEX index_name
| DROP FOREIGN KEY fk_symbol
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col_name
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
| [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
| DISCARD TABLESPACE
| IMPORT TABLESPACE
| table_options

```

7.1.6 Eliminação de DataBase

```
DROP DATABASE [IF EXISTS] db_name
```

7.1.7 Eliminação de Tabela

```

DROP [TEMPORARY] TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]

```

7.1.8 Eliminação de Índice

```
DROP INDEX index_name ON tbl_name
```

7.1.9 Eliminação de View

```
DROP VIEW { view }
```

7.2 Sintaxe dos comandos da sub-linguagem DML:

7.2.1 Inserção de Dados

```

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
  [INTO] tbl_name [(col_name,...)]
  VALUES ({expr | DEFAULT},...),(...),...
  [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
Or:
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
  [INTO] tbl_name
  SET col_name={expr | DEFAULT}, ...
  [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]

```

```
Or:  
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]  
  [INTO] tbl_name [(col_name,...)]  
SELECT ...
```

7.2.2 Alteração de Dados

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name  
  SET col_name1=expr1 [, col_name2=expr2 ...]  
  [WHERE where_definition]  
  [ORDER BY ...]  
  [LIMIT row_count]
```

7.2.3 Eliminação de Dados

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name  
  [WHERE where_definition]  
  [ORDER BY ...]  
  [LIMIT row_count]
```

7.2.4 Seleção de Dados

```
SELECT  
  [ALL | DISTINCT | DISTINCTROW ]  
  [HIGH_PRIORITY]  
  [STRAIGHT_JOIN]  
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]  
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]  
  select_expr, ...  
  [INTO OUTFILE 'file_name' export_options  
  | INTO DUMPFILE 'file_name']  
  [FROM table_references  
  [WHERE where_definition]  
  [GROUP BY {col_name | expr | position}  
  [ASC | DESC], ... [WITH ROLLUP]]  
  [HAVING where_definition]  
  [ORDER BY {col_name | expr | position}  
  [ASC | DESC], ...]  
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]  
  [PROCEDURE procedure_name(argument_list)]  
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

7.3 Consultas Simples e Complexas

```
SELECT select_list
FROM table_source
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

7.3.1 Seleção de todas as colunas de uma tabela

```
SELECT * FROM nome_da_tabela
```

O exemplo utiliza o coringa “*” para selecionar os dados de todas as colunas da tabela definida na cláusula FROM

7.3.2 Clausula WHERE

A cláusula "where" corresponde ao operador restrição da álgebra relacional. Contém a condição de seleção. Ela pode comparar valores em colunas, literais, expressões aritméticas ou funções.

Os operadores lógicos e complementares a serem utilizados nas expressões:

<u>Operador</u>	<u>Significado</u>
=	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 (‘)

Para facilitar o entendimento será exposto através de exemplos formas de seleção de dados:

Operação aritmética na lista de colunas – Seleção de todos os departamentos cujo orçamento mensal seja maior que 100000, apresentando o nome de tal departamento e seu orçamento anual, que será obtido multiplicando-se o orçamento mensal por 12.

Neste exemplo é preciso 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 DEPNOOME, DEPORCA * 12
FROM DEPT
WHERE DEPORCA > 100000;
```

Apelido (ALIAS) para nome de coluna – A instrução anterior apresentando ao invés dos cabeçalhos “DepNome e DepOrca”, os títulos “Departamento e Orçamento”.

Neste exemplo colunas são denominadas 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 CodCli significa Código do Cliente. Para melhor visualização e entendimento fica melhor apelidar as colunas na hora da disponibilização dos dados.

```
SELECT DEPNOOME AS "DEPARTAMENTO", DEPORCA * 12 AS "ORCAMENTO"
FROM DEPT
WHERE DEPORCA > 100000;
```

Eliminação de resultados iguais – Apresentação de todos os salários existentes na empresa, porém omitindo eventuais duplicidades.

A cláusula “Distinct” elimina duplicidades das linhas que apresentam valores iguais nas colunas selecionadas.

```
SELECT DISTINCT EMPSERV
FROM EMP;
```

Ordenação do resultado da seleção – Apresentação dos nomes e funções da cada funcionário contidos na tabela empresa, porém classificados alfabeticamente (A..Z) e depois alfabeticamente invertido (Z..A).

A cláusula “Order By” modificará a ordem de apresentação do resultado da pesquisa (ascendente ou descendente). A ordenação default é a ascendente.

```
SELECT EMPNOME, EMPSERV
FROM EMP
ORDER BY EMPNOME;
```

```
SELECT EMPNOME, EMPSERV
FROM EMP
ORDER BY EMPNOME DESC;
```

Nota 1: Também é possível fazer com que o resultado da pesquisa venha classificado por várias colunas.

Nota 2: Sem a clausula "order by" as linhas serão exibidas na seqüência que o SGBD determinar.

Demais Operadores

<u>Operador</u>	<u>Significado</u>
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 como “qualquer coisa”, isto é, um coringa.

Operadores Negativos

<u>Operador</u>	<u>Descrição</u>
<>	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

Seleção dos empregados cujos salários sejam menores que 1000 ou maiores que 3500.
Exemplificamos aqui com a utilização de expressão negativa.

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

Apresentação de todos os funcionários com salários entre 200 e 700 e que sejam Vendedores.
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';
```

Apresentação de todos os funcionários com salários entre 200 e 700 ou que sejam Vendedores.

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

Apresentação de todos os funcionários com salários entre 200 e 700 e que sejam Vendedores ou Balconistas.

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

Funções Agregadas (ou de Agrupamento)

<u>Função</u>	<u>Retorno</u>
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

Apresentação da Média, o Maior, o Menor e também a Somatória dos Salários pagos aos empregados.

```
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 linhas. A cláusula "GROUP BY" do comando "SELECT" é utilizada para dividir linhas em grupos menores.

Apresentação da média de salário pago por departamento, agrupando por número do departamento.

```
SELECT DUPNUM, AVG(EMPSALA)
FROM EMP
GROUP BY DEPNUM;
```

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.

Having

A cláusula "HAVING" pode ser utilizada para especificar quais grupos deverão ser exibidos, portanto restringindo-os, isto é, tem o mesmo papel da cláusula "WHERE" quando não se está utilizando a cláusula "Group By".

O exemplo anterior com a resposta somente para departamentos com mais de 10 empregados.

```
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".

A cláusula "WHERE" não pode ser utilizada para 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)
```

Seqüência no comando "SELECT":

```
SELECT      coluna(s)
FROM        tabela(s)
WHERE       condição(ões) da(s) tupla(s)
GROUP BY   condição(ões) do(s) grupo(s) de tupla(s)
HAVING     condição(ões) do(s) grupo(s) de tupla(s)
ORDER BY   coluna(s);
```

Equi-Junção (Junção por igualdade) ou JOIN ou RELACIONAMENTO

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

Relação com o Nomes do Empregados, Cargos e Nome do Departamento onde o empregado trabalha.

Observemos que dois dos três dados solicitados estão na Tabela Emp, enquanto o outro dado está na Tabela Dept. Deve-se 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 pelo JOIN.

```
SELECT A.EMPNO, A.EMPNO, B.DEPNO
FROM EMP A, DEPT B
WHERE A.DEPNO = B.DEPNO;
```

Obs.: As tabelas quando contém colunas com o mesmo nome, usa-se um apelido "alias" para substituir o nome da tabela associado a coluna. Imagine que alguém definiu 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 no exemplo anterior de forma adequada, pois o aliás se encarregaria de evitar que uma ambigüidade fosse verificada.

Relação dos Códigos do Cada Funcionário, seus Nomes, seus Cargos e o nome do Gerente ao qual este se relaciona.

Precisa-se 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 linhas da tabela a outras linhas da mesma tabela.

```
SELECT A.EMPNO, A.EMPNO, A.EMPNO, B.EMPNO
FROM EMP A, EMP B
WHERE A.MANAGER = B.EMPNO;
```

Sub-Consultas

Uma sub-consulta é um comando "SELECT" que é aninhado dentro de outro "SELECT" e que devolve resultados intermediários.

Relação de todos os nomes de funcionários e seus respectivos cargos, desde que o orçamento do departamento seja igual a 300000.

```
SELECT EMPNO, EMPNO
FROM EMP A
WHERE 300000 IN ( SELECT DEPTORCA
                  FROM DEPT
                  WHERE DEPT.DEPNO = A.DEPNO );
```

Nota: A cláusula IN torna-se verdadeira quando o atributo indicado está presente no conjunto obtido através da subconsulta.

Relação de todos os departamentos que possuem empregados com remuneração maior que 3500.

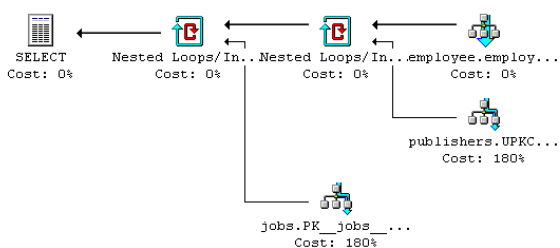
```
SELECT DEPTNO
FROM DEPT A
WHERE EXISTS (SELECT *
              FROM EMP
              WHERE EMPSALA > 3500 AND EMP.DEPNO = A.DEPNO');
```

Nota: A cláusula EXISTS indica se o resultado de uma pesquisa contém ou não linhas. Pode-se, também, verificar a não existência (NOT EXISTS) caso esta alternativa seja mais conveniente.

7.3.3 Clausula JOIN

Para resgatar dados de duas ou mais tabelas simultaneamente é preciso fazer o chamado JOIN, ou seja, indicar ao SGBD como ele encontrará os dados relacionados entre si nas tabelas envolvidas no processo. Esses relacionamentos estão indicados nas tabelas através da chave estrangeira (Foreign Key – FK) que faz referencia a chave primária (Primary Ky – PK) da tabela relacionada com ela.

A definição ou tradução de JOIN é dada por alguns dicionários como: “...unir; (a) reunir várias coisas; (b) combinar dois ou mais trechos de informação para produzir uma única unidade de informação; ± join files = unir arquivos = instrução que produz um novo arquivo consistindo de um arquivo adicionado ao final de um outro; (c) função lógica que produz uma saída verdadeira se qualquer entrada é verdadeira.”



O JOIN entre tabelas pode ser feito através das cláusulas WHERE ou JOIN, sendo que nos 2 casos o plano feito pelo SGBD para executar a transação é o mesmo, fazendo com que a preferência pelo uso de uma ou outra clausula seja particular. Eu, particularmente, dou preferência a WHERE, pois acho sua forma de escrita e entendimento mais simples.

Por exemplo:

```
SELECT  dbo.employee.emp_id, dbo.employee.fname, dbo.employee.lname,
        dbo.publishers.pub_name, dbo.jobs.job_desc
FROM    dbo.jobs INNER JOIN
        dbo.employee ON dbo.jobs.job_id = dbo.employee.job_id INNER JOIN
        dbo.publishers ON dbo.employee.pub_id = dbo.publishers.pub_id
```

ou

```
SELECT  dbo.employee.emp_id, dbo.employee.fname, dbo.employee.lname,
        dbo.publishers.pub_name, dbo.jobs.job_desc
FROM    dbo.jobs, dbo.employee, dbo.publishers
WHERE   dbo.jobs.job_id = dbo.employee.job_id AND
        dbo.employee.pub_id = dbo.publishers.pub_id
```

No uso da clausula WHERE o SGBD necessita saber quando a FK de uma tabela tem o mesmo valor da PK da tabela referenciada (relacionada) e no uso de mais de duas tabelas utiliza-se o operador de ligação AND.

Por exemplo:

WHERE PK da Tabela 1 = FK da Tabela 2

Ou

WHERE PK da Tabela 1 = FK da Tabela 2 AND PK da Tabela 2 = FK da Tabela 3 AND ...

A clausula JOIN é utilizado dentro da clausula FROM do comando de seleção dos dados e para isso existem várias formas de se extrair os dados: INNER JOIN, CROSS JOIN, LEFT OUTER JOIN e RIGHT OUTER JOIN.

No uso do JOIN dentro da clausula FROM defini-se a 1ª tabela, a clausula JOIN desejada, a tabela referenciada pela 1ª, a palavra ON, a FK da 1ª tabela, o operador de igualdade (=) e a PK da tabela referenciada.

Por exemplo:

```
FROM      Tabela 1 INNER JOIN Tabela 2
         ON      PK da Tabela 1 = FK da Tabela 2
INNER JOIN Tabela 3
         ON      FK da Tabela 2 = PK da Tabela 3
```

Os JOIN retornam os seguintes resultados:

- ✓ INNER JOIN retorna as linhas das tabelas que sejam comuns entre si;
- ✓ LEFT OUTER JOIN retorna todas as linhas da tabela a esquerda do relacionamento que não possuem linhas equivalentes na outra e as linhas correspondentes;
- ✓ RIGHT OUTER JOIN retorna todas as linhas da tabela a direita do relacionamento que não possuem linhas equivalentes na outra e as linhas correspondentes;
- ✓ CROSS JOIN retorna todas as linha das tabelas relacionadas. O resultado é o produto cartesiano do número de linhas das duas tabelas relacionadas (n° de linhas da 1ª tabela multiplicado pelo n° de linhas da 2ª tabela).