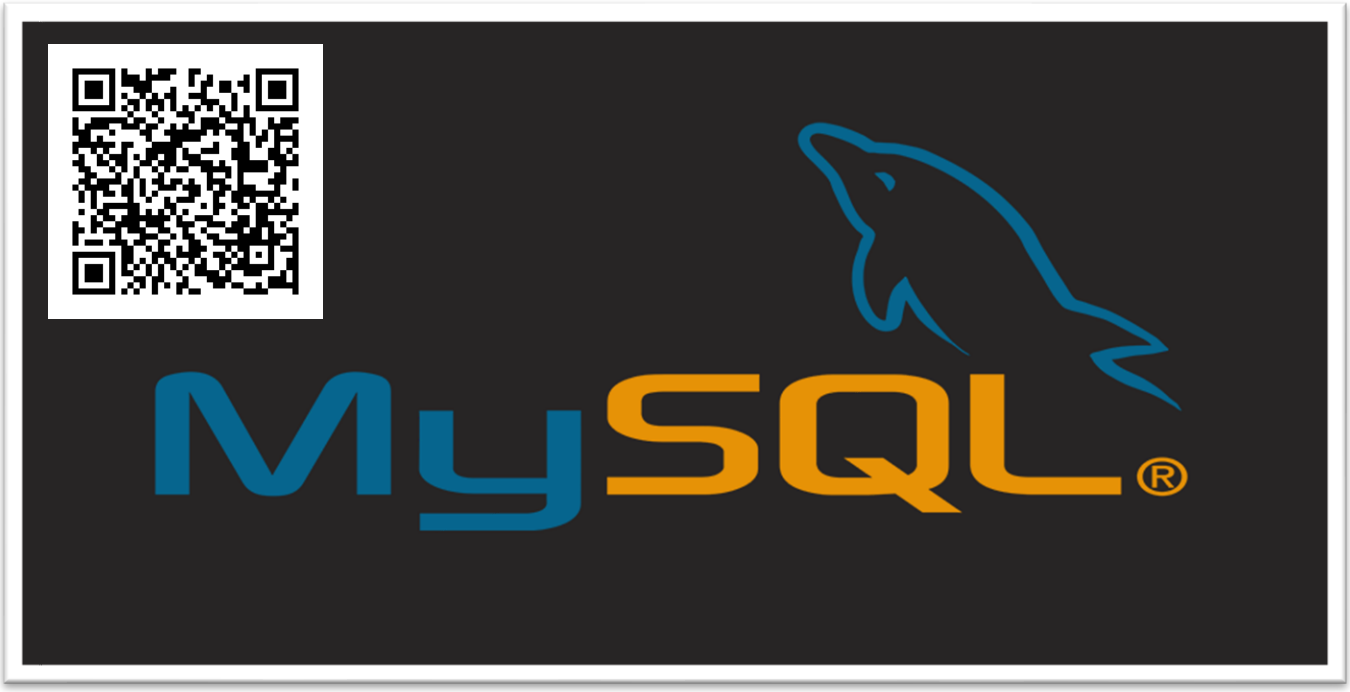
Banco de Dados

Sumário

INTRODUÇÃO



A tecnologia utilizada em métodos de armazenamento de informações vem crescendo e provocando um impacto incremental no uso de computadores, em todas as áreas em que são aplicados.

Um sistema de banco de dados adiciona uma nova dimensão à estrutura de gerenciamento de uma organização. A complexidade dessa estrutura depende do tamanho da organização, de suas funções e de sua cultura empresarial. Porquanto, os sistemas de banco de dados podem ser criados e gerenciados em diferentes níveis de complexidade e com adesão variável a padrões precisos.

Por exemplo, compare um sistema local de locação de filmes com um sistema nacional de reclamações de seguros. O sistema de locação de filmes pode ser gerenciado por duas pessoas, o hardware utilizado provavelmente é um único microcomputador, os procedimentos são simples e o volume de dados tende a ser baixo; Já o sistema nacional de reclamações de seguros possui pelo menos um administrador de sistemas, vários DBA´s em tempo integral e muitos projetistas e programadores; O hardware inclui diversos servidores em vários locais por todo o país, e é provável que os procedimentos sejam numerosos, complexos e rigorosos e que o volume de dados tenda a ser alto.

TEMA 1: INTRODUÇÃO A BANCO DE DADOS

Figura 1: O banco de dados / Fonte: Freepix

Nos dias atuais, podemos mencionar alguns itens relevantes para atingir a eficiência e a eficácia dos sistemas desenvolvidos, com a finalidade de atender seus usuários nos mais variados domínios de aplicação: automação de corporações, sistemas de apoio a decisões, controle de reserva de recursos, controle e planejamento de produção, alocação e estoque de recursos entre outros. Alguns aspectos são:

- Projetos Lógico e Funcional do Banco de Dados: São capazes de atuar de forma antecipada no volume de informações armazenadas a curto, médio e longo prazo. Os projetos devem ter uma grande capacidade de adaptação para os três casos mencionados;
- Necessitam ter generalidade e alto grau de abstração de dados, possibilitando confiabilidade e eficiência no armazenamento dos dados e permitindo a utilização de diferentes tipos de gerenciadores de dados através de linguagens de consultas padronizadas;
- Projeto de uma interface ágil e com uma "rampa ascendente" para propiciar aprendizado suave ao usuário, no intuito de minimizar o esforço cognitivo;
- Implementação de um projeto de interface compatível com múltiplas plataformas (UNIX, Windows Server, Windows Workgroup, etc.);
- Independência de Implementação da Interface em relação aos SGBDs que darão condições às operações de armazenamento de informações (ORACLE, SYSBASE, INFORMIX, PADRÃO XBASE, etc.).
- Conversão e mapeamento da diferença semântica entre os paradigmas utilizados no desenvolvimento de interfaces (Imperativo (ou procedural), Orientado a Objeto, Orientado a evento), servidores de dados (Relacional) e programação dos aplicativos (Imperativo, Orientado a Objetos).

Conceitos Gerais

Um Banco de Dados pode ser definido como um conjunto de dados organizados e relacionados. A definição de dados compreende como o uso de “fatos conhecidos” que podem ser armazenados e que possuem um significado importante.

A definição do termo Banco de Dados é atende às seguintes propriedades:

- é uma coletânea lógica coesa de dados com um significado intrínseco; dados desordenados não podem ser referenciados como um banco de dados;
- é projetado, construído e populado com dados com objetivos específicos; possui um conjunto pré-definido de usuários e aplicações;
- simula aspectos do mundo real, sendo que quaisquer alterações no cenário a que ele representa são automaticamente refletidas no banco de dados.

A informação em um banco está organizada em forma de registros. Cada registro contém toda a informação sobre uma pessoa ou um elemento do banco. Por exemplo, cada registro no diretório telefônico contém um nome, endereço e número telefônico de uma pessoa.

Cada registro contém campos. Um campo é utilizado para armazenar o endereço e outro campo para armazenar o número telefônico da pessoa. Cada registro contém cada um destes campos e cada registro pode ter informação nestes campos.

O nome de um campo geralmente identifica a informação armazenada no campo. Por exemplo, os campos podem se chamar Nome, Endereço ou Número telefônico. Cada campo tem um tipo que identifica a classe de informação que pode ser armazenada: números, datas, caracteres alfanuméricos e outros. Como cada campo contém um tipo específico de dados, você pode realizar cálculos e outras operações com a informação guardada neles. Por exemplo, pode somar os números dos campos. Pode comparar a data de um campo com a de outro. Pode mostrar o nome de uma pessoa (armazenado em um campo) depois de seu sobrenome (armazenado em outro campo) para construir a primeira linha de uma etiqueta de correio.

O conjunto de registros que utilizam os mesmos campos forma uma tabela. Cada banco de dados pode ter muitas tabelas. A imagem a seguir mostra como se relacionam estes conceitos.

Um banco de dados pode ser desenvolvido por aplicações específicas para esta tarefa ou por um Sistema Gerenciador de Banco de Dados (SGBD). Um SGBD permite aos usuários criarem e manipularem bancos de dados de propósito geral. O conjunto formado por um banco de dados mais as aplicações que o manipulam é chamado de Sistema de Banco de Dados.

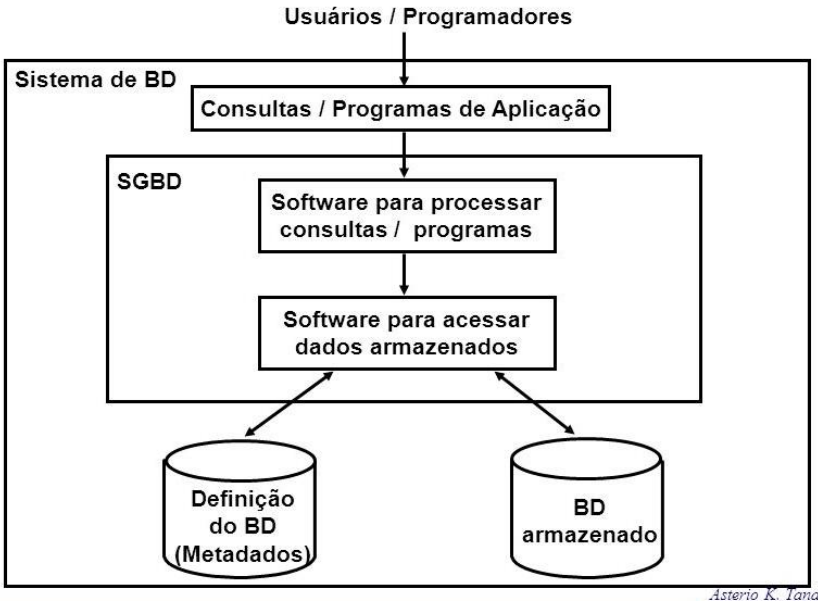


Figura 2: Um ambiente de Sistema de Banco de Dados / Fonte: Tanaka, A.K.

Arquiteturas

Nas arquiteturas primárias o uso dos mainframes para executar o processamento principal e de todas as funções do sistema, incluindo os programas aplicativos, programas de interface com o usuário, assim como as funcionalidades dos SGBDs.

O motivo pelo qual grande parte dos usuários fazia acesso aos sistemas via terminais que não possuíam poder de processamento, apenas a capacidade de visualização.

Os processamentos eram feitos remotamente, apenas as informações a serem visualizadas e os controles eram enviados do mainframe para os terminais de visualização, conectados a ele por redes de comunicação. Como os preços do hardware foram caindo, ou diminuindo, muitos usuários alteraram seus terminais por microcomputadores (PC) e workstations.

No início os SGBDs usavam esses computadores da mesma maneira que usavam os terminais, ou seja, o SGBD era centralizado e toda sua funcionalidade, execução de programas aplicativos e processamento da interface do usuário eram executados em apenas uma máquina. De forma gradual, os SGBDs iniciaram a explorar a disponibilidade do poder de processamento no lado do usuário, o que levou à arquitetura cliente-servidor.

A arquitetura cliente-servidor fora criada para separar ambientes de computação onde um grande número de PCs, estações de trabalho, servidores de arquivos, impressoras, servidores de banco de dados e outros equipamentos são conectados juntos por uma rede. Definir servidores especializados, tais como servidor de arquivos, que mantém os arquivos de máquinas clientes, ou servidores de impressão que podem estar conectados a várias impressoras; assim, quando se desejar imprimir algo, todas as requisições de impressão são enviadas a este servidor.

Os PCs clientes possibilitaram para o usuário as interfaces adequadas para usar esses servidores, bem como poder de processamento para executar aplicações locais. Esta arquitetura se tornou muito popular por algumas razões. Primeiro, a facilidade de implementação dada a clara separação das funcionalidades e dos servidores. Segundo, um servidor é inteligentemente utilizado porque as tarefas mais simples são delegadas às máquinas clientes mais baratas. Terceiro, o usuário pode executar uma interface gráfica que lhe é familiar, ao invés de usar a interface do servidor. Desta maneira, a arquitetura cliente-servidor foi incorporada aos SGBDs comerciais.

Diferentes técnicas foram propostas para se implementar essa arquitetura, sendo que a mais adotada pelos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) comerciais é a inclusão da funcionalidade de um SGBD centralizado no lado do servidor.

As consultas e a funcionalidade transacional permanecem no servidor, sendo que este é chamado de servidor de consulta ou servidor de transação. É assim que um servidor SQL é fornecido aos clientes. Cada cliente tem que formular suas consultas SQL, prover a interface do usuário e as funções de interface usando uma linguagem de programação. O cliente pode também se referir a um dicionário de dados que inclui informações sobre a distribuição dos dados em vários servidores SQL, bem como os módulos para a decomposição de uma consulta global em um número de consultas locais que podem ser executadas em vários sítios. Comumente o servidor SQL também é chamado de backend machine e o cliente de frontend machine. Como SQL provê uma linguagem padrão para o SGBDRs, esta criou o ponto de divisão lógica entre o cliente e o servidor.

Um sistema de banco de dados adequado a um ambiente de sistema de arquivos, fornece um modelo no qual podem ser aplicados procedimentos e padrões rígidos. Como consequência, o papel do componente humano muda da ênfase em programação (no sistema de arquivos) para focar nos aspectos mais amplos de gerenciamento dos recursos de dados da organização e na administração do próprio software do banco de dados complexo.

O sistema de banco de dados torna possível atingir usos muito mais sofisticados dos recursos de dados contanto que seja projetado para aproveitar esse poder disponível. Os tipos de estruturas de dados criados no banco de dados e a extensão dos relacionamentos entre elas desempenham um papel poderoso na determinação da eficiência do sistema.

Atualmente, existem várias tendências para arquitetura de Banco de Dados, nas mais diversas direções.

Vantagens e desvantagens do uso de um SGBD

Controle de Redundância

Quando se trata do sistema de armazenamento e processamento de arquivos, sem utilizar um SGBD, os dados são responsabilidade dos usuários que os utilizam. Neste caso, as redundâncias são inevitáveis e surgem as seguintes dificuldades:

- quando é necessária a atualização de um arquivo específico de usuário ou grupo, todos os dados devem ser atualizados para manter a sua integridade no ambiente, como um todo;
- dados redundantes levam a um armazenamento excessivo de informações, tomando espaço que poderia estar sendo utilizado com novas informações.

Compartilhamento de Dados

Um SGBD multiusuário deve consentir que muitos usuários acessem o banco de dados ao mesmo tempo. O que é fundamental essencial para múltiplos acessos das aplicações integradas ao banco de dados. Deve ainda, ter o controle de concorrência para garantir que as atualizações estejam corretas. Deve, também, oferecer recursos para a multiplicidade de visões.

Restrição a Acesso não Autorizado

O SGBD precisa conter a segurança no acesso às contas de usuários, com as devidas restrições de acordo com os perfis de cada conta, sendo aplicado tanto para o acesso, como às aplicações que são gerenciadas por ele.

Representação de Relacionamentos Complexos entre Dados

Um banco de dados tem em si um conjunto de dados relacionados de muitas formas. O SGBD deve gerenciar todos os relacionamentos entre os dados, assim como, guardar, recuperar e atualizar de forma dinâmica e eficaz.

Tolerância a Falhas

Um SGBD precisa gerenciar e oferecer formas de recuperação a falhas, sejam físicas ou lógicas.

Há situações que não se deve utilizar um SGBD?

Sim, quando o uso de um SGBD representar um alto custo em comparação aos sistemas de processamento tradicional de arquivos:

- seja um alto custo na compra do software ou infraestrutura;
- generalidade que um SGBD fornece na definição e processamento de dados;
- sobrecarga no fornecimento do controle da segurança, concorrência, recuperação e integração de funções.

Problemas adicionais podem surgir caso os projetistas de banco de dados ou os administradores de banco de dados não elaborem os projetos corretamente ou se as aplicações não são implementadas de forma apropriada.

Se o Administrador do banco de dados (DBA – Database Administrator) não administrar o banco de dados de forma adequada, tanto a segurança quanto a integridade dos sistemas podem ser comprometidas. Algumas questões justificam a utilização uma abordagem processamento tradicional de arquivos:

- quando um banco de dados e suas aplicações são simples, bem definidas e não se espera mudanças no projeto;
- não há a necessidade de processamento em tempo real das aplicações, podendo ter prejuízo caso sejam sobrecarregadas por um SGBD;
- quando não houver múltiplos acessos ao banco de dados, ao mesmo tempo.

Atividades

1. O que são banco de dados?

2. Explique o conceito de generalidade e alto grau de abstração de banco de dados.

3. Comente por que é importante a eficiência dos sistemas atualmente?

4. Como é possível atingir usos muito mais sofisticados dos recursos de dados?

5. Como eram as primeiras arquiteturas de banco de dados?

6. O que é um SGBDR?

7. O que é um SGBD ou Sistema Gerenciador de Bancos de Dados e qual é a sua

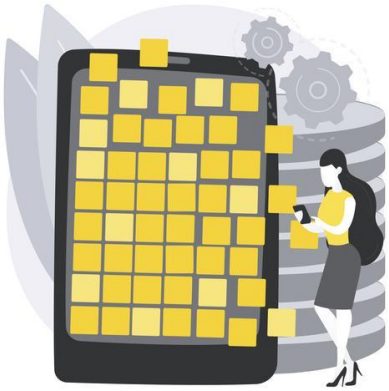
melhoria em relação ao armazenamento de dados em arquivos?

8. Quais os benefícios de usar um sistema de banco de dados para uma empresa?

9. Se o Administrador do banco de dados (DBA – Database Administrator) não administrar o banco de dados de forma adequada, o que será comprometido?

10. Algumas empresas ainda usam abordagem processamento tradicional de arquivos, quais motivos disso ainda acontecer?

TEMA 2: COMPONENTES DO SGBD



Um SGBD executa várias funções importantes que garantem a integridade e a consistência dos dados no banco. A maioria dessas funções é transparente para o usuário final e pode ser conseguida apenas pelo uso de um SGBD. Incluem gerenciamento de dicionário e armazenamento de dados, transformação e apresentação de dados, gerenciamento de segurança, controle de acesso multiusuário, gerenciamento de backup e recuperação, gerenciamento de integridade de dados, linguagens de acesso ao banco de dados e interfaces de programação de aplicações, além de comunicação do banco de dados.

Gerenciamento do dicionário de dados

O SGBD armazena as definições de elementos de dados e seus relacionamentos (metadados) em um dicionário de dados. Por sua vez, todos os programas que acessam os dados no banco trabalham por meio do SGBD. Este utiliza o dicionário de dados para procurar os relacionamentos e de componentes de estruturas de dados necessárias, livrando o usuário de ter de codificar esses relacionamentos complexos em cada programa. Além disso, quaisquer mudanças feitas na estrutura do banco de dados são automaticamente registradas no dicionário de dados, o que isenta o usuário da necessidade de modificar todos os programas que acessa a estrutura alterada. Em outras palavras o SGBD fornece abstração de dados e remove a dependência estrutural e de dados do sistema.

Gerenciamento de armazenamento de dados

O SGBD cria e gerencia estruturas complexas necessárias para o armazenamento de dados, livrando o usuário da difícil tarefa de definir e programar as características dos dados físicos. Um SGBD moderno fornece armazenamento não apenas para os dados, mas também para as definições de telas, formulários de entrada de dados relacionados, as definições de relatórios, as regras de validação de dados, o código procedural, as estruturas para lidar com formatos de vídeo e imagem e assim por diante.O gerenciamento de armazenamento de dados também é importante para o tuning de banco de dados.

O tuning remete as atividades que tornam o desempenho do banco de dados mais eficiente em termos de armazenamento de dados, o SGBD, na verdade, armazena o banco em vários arquivos de dados físicos. Esses arquivos de dados podem ser mantidos em diferentes meios de armazenagem. Portanto o SGBD não precisa esperar que uma solicitação de disco termine para iniciar a próxima. Em outras palavras, o SGBD pode atender às solicitações concorrentes ao banco de dados.

Transformação e apresentação de dados

O SGBD transforma os dados inseridos em conformidade com a estrutura de dados necessárias. O SGBD isenta o usuário da desagradável tarefa de distinguir entre os formatos de dados lógicos e físicos. Ou seja, o SGBD formata os dados recuperados fisicamente para conformá-los às expectativas lógicas do usuário.

Gerenciamento de segurança.

O SGBD cria um sistema de segurança que garante a segurança de usuário e a privacidade dos dados. As regras de segurança determinam quais usuários podem acessar o banco de dados, quais itens de dados cada usuário pode acessar e quais operações de dados (leitura, adição, exclusão ou modificação) o usuário pode executar. Isso é especialmente importante em sistemas de dados multiusuários.

Controle de acesso.

Para fornecer integridade e consistência de dados, o SGBD utiliza algoritmos sofisticados, garantindo que vários usuários possam acessar o banco de dados ao mesmo tempo em comprometer sua integridade.

Gerenciamento de backup e recuperação.

O SGBD fornece backup e recuperação de dados para garantir a segurança e integridade dos dados. Os sistemas atuais de SGBD oferecem utilitários especiais que permitem ao DBA executar backups especiais e de rotina, procedimentos de restauração.

O gerenciamento de recuperação trata da recuperação do banco de dados após uma falha, como um erro de setor no disco ou uma queda de energia.

Gerenciamento de integridade de dados.

O SGBD promove e aplica regras de integridade, minimizando, assim, a redundância de dados e maximizando sua consistência. Os relacionamentos de dados armazenados no dicionário de dados são utilizados para garantir a integridade. Tal garantia é especialmente importante em sistemas de banco de dados orientado às transações.

Linguagem de acesso a banco de dados e interface de programação de aplicações.

O SGBD fornece acesso aos dados por meio de uma linguagem de consulta. A linguagem de consulta é uma linguagem não procedural, ou seja, permite que o usuário que o usuário especifique o que deve ser feito, sem ter de especificar como deve fazer.

A linguagem estruturada de consulta (SQL) é uma linguagem de consulta vigente e o padrão de acesso de dados suportado pela maioria dos fornecedores de SGBD.

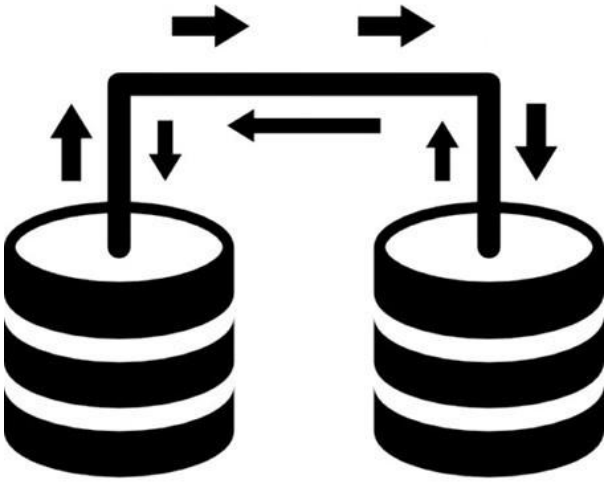
Interface de comunicação do banco de dados.

Os SGBD´s da geração atual aceitam solicitações do usuário final por meio de vários ambientes de rede diferentes. Por exemplo, o SGBD pode fornecer acesso ao banco de dados pela internet por meio do uso de navegadores web. Nesse ambiente, as comunicações podem ser realizadas de diversas maneiras.

DESAFIO

Apresente as funcionalidades de um SGBD de forma prática.

TEMA 3: MODELAGEM DE BANCOS DE DADOS



Um modelo de dados é uma abstração de um ambiente de dados real e complexo. Os projetistas de banco de dados utilizam os modelos de dados para se comunicar com programadores e usuários de aplicações. Os componentes básicos de modelagem de dados são as entidades, os atributos, os relacionamentos e as restrições. As regras de negócio são utilizadas para identificar e definir os componentes básicos de modelagens em um ambiente específico real.

Projeto de Banco de Dados

O projeto de banco de dados foca em como a estrutura do banco de dados será utilizada para armazenar e gerenciar dados do usuário final.

A modelagem de dados deve ser específica para um determinado problema de domínio. Esse problema de domínio é uma área claramente definida no ambiente real, com escopo e fronteiras bem definidos, que deve ser tratada de forma sistemática, e quanto mais fiel a modelagem for ao ambiente do problema em domínio, maior é a chance de o projeto ter um bom resultado, permitindo assim a criação de um banco de dados mais aderente à realidade, possibilitando de forma mais eficiente o desenvolvimento da aplicação.

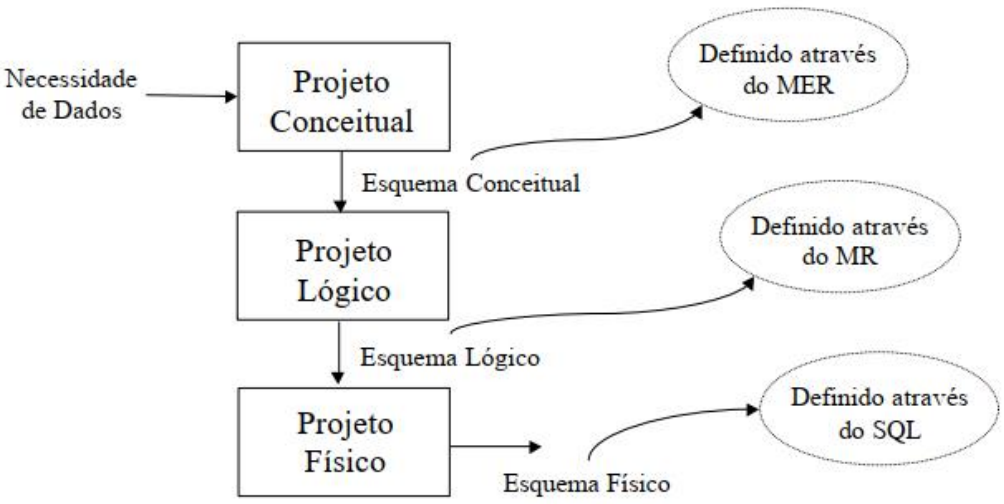


Figura 3: Projeto Banco de Dados/ Fonte: Plastino (2002)

O objetivo da modelagem de dados é garantir que todos os objetos de dados existentes em determinado contexto e requeridos pela aplicação estejam representados com precisão dentro do Banco de Dados.

Pode-se definir modelagem de dados como sendo um conjunto de conceitos que devem ser usados para descrever a estrutura de uma base de dados. Um modelo de dados é uma representação relativamente simples, normalmente gráfica, de estruturas de dados reais mais complexas.

Em termos gerais, modelo é uma abstração de um objeto ou evento real de maior complexidade do ambiente real.

No ambiente de banco de dados, um modelo representa estruturas de dados e suas características, relações, restrições, transformações e outros elementos que tenham a finalidade de dar suporte ao problema específico de domínio.

MODELO CONCEITUAL

O Modelo Conceitual é uma abstração da realidade, em que os aspectos do mundo real são descritos de forma natural, bem como seus atributos e relacionamentos. Esse modelo é utilizado para entendimento, transmissão, validação de conceitos e mapeamento do ambiente, possibilitando um melhor diálogo entre desenvolvedores e usuários.

O Modelo Conceitual não está relacionado diretamente com o modelo de banco de dados, forma de acesso ou armazenamento dos dados. Ele está focado em uma representação gráfica da uma realidade existente em um contexto de negócio, conforme está ilustrado na figura. Essa modelagem é feita utilizando o modelo entidade-relacionamento. Tem como características:

- Visão Geral do negócio;
- Facilita o entendimento entre usuários e desenvolvedores;
- Possui somente as entidades e atributos principais;
- Pode conter relacionamentos muitos para muitos.

O modelo conceitual representa uma visão global do banco de dados inteiro conforme visto pela organização como um todo. Ou seja, o modelo integra todas as visões externas (entidades, relacionamentos, restrições e processos) em uma única visão global de todos os dados da empresa. Também conhecido como esquema conceitual, constitui a base para a identificação e descrição de alto nível dos principais objetos de dados (evitando quaisquer detalhes específicos do modelo de banco de dados).

O modelo conceitual mais utilizado é o ER. Lembre-se que o modelo ER é ilustrado com a ajuda do DER que, na prática, constitui a planta básica do banco. Este é utilizado para representar graficamente o esquema conceitual.

O modelo conceitual produz algumas vantagens importantes. Em primeiro lugar, fornece uma visão de cima (nível macro) compreendida de modo relativamente fácil sobre o ambiente de dados.

Em segundo lugar, o modelo conceitual é independente em relação tanto a software como a hardware.

A independência de software significa que o modelo não é dependente de software SGBD utilizado para implantá-lo. A independência de hardware significa que o modelo não depende do hardware utilizado em sua implantação. Portanto, alterações de hardware ou software do SGBD não terão efeito sobre o projeto de banco de dados no nível conceitual. Em geral, o termo projeto lógico é utilizado parase referir as tarefas de criação de modelo de dados conceitual que possa ser implantado em qualquer SGBD.

MODELO LÓGICO

Leva em conta limites impostos por algum tipo de tecnologia de banco de dados. (banco de dados hierárquico, banco de dados relacional, entre outros. Suas características são:

- Deriva do modelo conceitual e via a representação do negócio

- Possui entidades associativas em lugar de relacionamentos muitos para muitos
- Define as chaves primárias das entidades
- Normalização até a 3ª forma normal
- Adequação ao padrão de nomenclatura
- Entidades e atributos documentados

MODELO INTERNO

Uma vez selecionado o SGBD específico, o modelo interno mapeia o modelo conceitual para o SGBD. O modelo interno é a representação do banco de dados conforme “visto” pelo SGBD. Em outras palavras, o modelo interno exige que o projetista relacione as características e restrições do modelo conceitual com as do modelo selecionado para implementação. O esquema interno constitui a representação específica de um modelo interno, utilizando estruturas de banco de dados suportadas pelo banco escolhido.

Portanto, o esquema interno deve mapear o modelo conceitual para as estruturas do modelo relacional, de modo similar, como selecionamos um banco de dados relacional, o esquema interno é expresso utilizando SQL, linguagem padrão para este banco.

Como o modelo interno depende do software específico do banco de dados, diz-se que ele é dependente de software. Portanto, uma alteração no software de SGBD exige que o modelo interno seja alterado para adequar-se às características e exigências de implementação do modelo de banco de dados. Quando é possível alterar o modelo interno sem afetar o modelo conceitual, tem-se independência lógica. No entanto, o modelo interno ainda é independente de hardware, pois não é afetado pela escolha do computador em que o software é instalado. Portanto, uma alteração nos dispositivos de armazenamento ou mesmo nos sistemas operacionais não afetará o modelo interno.

MODELO FÍSICO

Leva em consideração limites impostos pelo SGBD (Sistema Gerenciador de Banco de dados) e pelos requisitos não funcionais dos programas que acessam os dados. Características:

- Elaborado a partir do modelo lógico
- Pode variar segundo o SGBD
- Pode ter tabelas físicas (log)
- Pode ter colunas físicas (replicação)

O modelo físico opera nos níveis mais baixos de abstração, descrevendo o modo como os dados são salvos em meios de armazenamento como discos e fitas. O modelo físico exige a definição tanto dos dispositivos de armazenamento físico como dos métodos de acesso (físico) necessários para se chegar aos dados nesses dispositivos de armazenamento, o que torna dependente tanto de software quanto de hardware. As estruturas de armazenamento utilizados são dependentes do software SGBD e sistema operacional, e dos tipos de dispositivos de armazenamento com que o computador pode trabalhar. A precisão necessária na definição do modelo físico exige que o projetista que trabalha nesse nível tenha conhecimento detalhado do hardware e do software utilizado para implementar o projeto de banco de dados.

Modelos de dados anteriores exigiam que os projetistas levassem em conta com os detalhes das necessidades de armazenamento de dados do modelo físico. No entanto, o modelo relacional atualmente dominante é direcionado amplamente para o nível lógico, não para o físico, portanto, não exige os detalhes desse segundo nível como seus antecessores.

Embora o modelo relacional não demande que o projetista se preocupe com as características de armazenamento físico dos dados, a implementação de um modelo relacional pode exigir sintonização refinada em nível físico para melhorar o desempenho. Essa sintonização refinada é especialmente

importante quando os bancos de dados muito grandes são instalados em um ambiente mainframe. Mesmo assim, essa sintonização não exige conhecimento das características de armazenamento físico.

O modelo físico é dependente do SGBD, dos métodos de acesso aos arquivos e dos tipos de dispositivos de armazenamento suportados pelo sistema operacional. Quando é possível alterar o modelo físico sem afetar o modelo interno, tem-se independência física. Portanto uma alteração nos dispositivos ou métodos de armazenamento ou mesmo sistema operacional não afetará o modelo interno.

MODELO	GRAU DE ABSTRAÇÃO	FOCO	INDEPENDÊNCIA DE:
Externo	Alto	Visões dos usuários finais	Hardware e software
Conceitual		Visão global dos dados (independente do modelo de banco de dados	Hardware e software
Interno		Modelo específico de banco de dados	Hardware
Físico	Baixo	Métodos de armazenamento e acesso	Nem hardware, nem software.

MODELO DE DADOS RELACIONAL.

O Modelo de Dados Relacional foi introduzido por Codd (1970). Entre os modelos de dados de implementação, o modelo relacional é o mais simples, com estrutura de dados uniforme, e o mais formal. O modelo de dados relacional representa os dados da base de dados como uma coleção de relações. Informalmente, cada relação pode ser entendida como uma tabela ou um simples arquivo de registros.

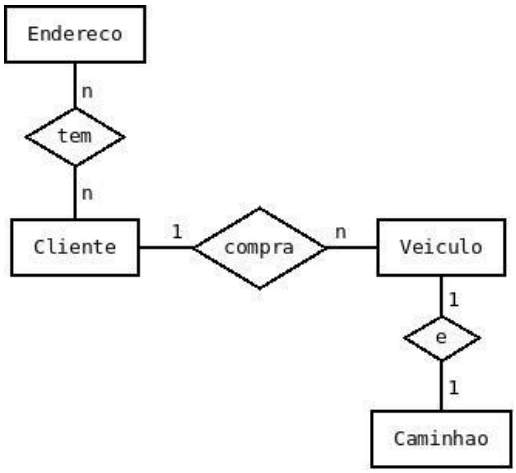


Figura 4: Representação do MER como um Diagrama Entidade-Relacionamento/ Fonte:Medium.

DDL - SQL

O padrão SQL surgiu para acessarmos e pesquisarmos dados armazenados e estruturados de uma forma específica. É uma linguagem de consultas estruturadas, Structured Query Language, ou simplesmente SQL. Para conseguirmos utilizar essa linguagem, precisamos instalar um SGBD que será o nosso servidor de banco de dados, como por exemplo, o PostgreSQL, MySQL, Oracle ou SQLServer.

Conceitos Do Modelo Entidade-Relacionamento

O objeto básico que o MER representa é a entidade. Os blocos básicos de construção de todos os modelos de dados são as entidades, os atributos, os relacionamentos e as restrições.

Entidade

Uma entidade é algo (uma pessoa, um local, um objeto, um evento) sobre o qual sejam coletados e armazenados dados. Ela representa um tipo particular de objeto do mundo real. Por isso, as entidades são distinguíveis, ou seja, cada ocorrência de entidade é única e distinta. As entidades podem ser objetos físicos, como clientes e produtos, mas também abstrações como rotas de voo ou apresentações musicais. Pode ser definida como qualquer coisa do mundo real, abstrata ou concreta, onde se deseja guardar informações. Exemplos de entidades: Cliente, Veículo, Produto, Venda.

Atributo

Um atributo é uma característica de uma entidade. É tudo o que se pode relacionar como propriedade da entidade (coluna, campo). Chama-se Domínio o conjunto de valores possíveis do atributo. Por exemplo uma entidade chamada cliente seria descrita por atributos como sobrenome, nome, telefone, endereço e limite de crédito de clientes.

Alguns atributos são obrigatórios outros são opcionais. Algumas definições:

- Atributo obrigatório – é aquele que para uma entidade ou relacionamento deve possuir um valor.
- Atributo opcional - É aquele que para uma instância da entidade ou relacionamento pode possuir um valor ou não.

Classificação:

- Atributo Identificador - capaz de identificar exclusivamente cada ocorrência de uma entidade. Também conhecido como chave Primária ou Primary Key (PK). O símbolo # é usado para representar a chave primária em algumas notações)
  Ex.: Código do Cliente, Código do Produto
- Chave Candidata - Atributo ou grupamento de atributos que têm a propriedade de identificar unicamente uma ocorrência da entidade. Pode vir a ser uma chave Primária. A chave candidata que não é chave primária também se chama chave Alternativa.

Características de uma Chave Primária:

- NÃO PODE haver duas ocorrências de uma mesma entidade com o mesmo conteúdo na Chave Primária
- A chave primária não pode ser composta por atributo opcional, ou seja, atributo que aceite nulo.
- Os atributos identificadores devem ser o conjunto mínimo que pode identificar cada instância de uma entidade.
- Não devem ser usadas chaves externas ou atributos sobre os quais você não tem controle.
- Não deve conter informação volátil.

Ao criar modelos geralmente temos diversas entidades cada uma com diversos atributos que podem se relacionar entre si. Vamos definir como podem ser estes relacionamentos.

Relacionamento

Um relacionamento pode ser entendido como uma associação entre instâncias de Entidades de acordo com as regras de negócio. Normalmente ocorre entre instâncias de duas ou mais Entidades, podendo ocorrer entre instâncias da mesma Entidade (auto relacionamento).

Quando o relacionamento é necessário? Quando existem várias possibilidades de relacionamento entre o par das entidades e se deseja representar apenas um, se ocorrer mais de um relacionamento entre o par de entidades, a fim de evitar ambiguidade, quando houver auto relacionamento, para definir o número de ocorrências de uma entidade usamos o conceito de Cardinalidade.

Os modelos de dados utilizam três tipos de relacionamentos:

- **Relacionamento um para muitos (1:M ou 1..*).**

Por exemplo um pintor faz várias pinturas, mas cada uma é criada por apenas um artista. Assim o pintor (uma entidade) relaciona-se com as pinturas (várias entidades).
Portanto, os projetistas de banco de dados identificam o relacionamento PINTOR pinta PINTURA como 1:M

- **Relacionamento de muitos para muitos (M: N ou *..*).**

Um funcionário pode aprender várias habilidades profissionais e cada habilidade profissional pode ser aprendida por vários funcionários. Os projetistas de banco de dados identificam o relacionamento FUNCIONÁRIO aprende HABILIDADE como M: N

- **Relacionamento um para um (1:1 ou 1..1)**

A estrutura de gerenciamento de uma empresa de varejo pode exigir que cada uma de suas lojas seja gerenciada por um único funcionário. Por sua vez, cada gerente de loja, que é um funcionário, gerencia uma loja apenas. Portanto o relacionamento FUNCIONÁRIO gerencia LOJA é identificado como 1:1.

A discussão precedente identificou cada relacionamento em duas direções, ou seja, os relacionamentos são bidirecionais:

- Um CLIENTE pode gerar várias FATURAS.
- Cada uma das várias FATURAS é gerada apenas por um CLIENTE.

Uma restrição é uma limitação imposta aos dados. As restrições são importantes, pois ajudam a assegurar a integridade dos dados. Elas normalmente são expressas na forma de regras.

Regras de negócio.

Quando os projetistas de banco de dados cuidam da seleção ou determinação das entidades, atributos e relacionamentos utilizados para construir um modelo de dados, podem começar obtendo uma compreensão completa de quais tipos de dados existem em uma organização, como são utilizados e em que período são utilizados. Mas esses dados e informações não produzem por si mesmo, a compreensão necessária do negócio como um todo. Do ponto de vista do banco de dados, o conjunto de dados adquire significado apenas quando representa adequadamente as regras de negócio definidas.

Uma regra de negócio é uma descrição breve, precisa e sem ambiguidades de uma política, procedimento ou princípio em uma determinada organização.

As regras de negócio decorrentes de uma descrição detalhada das operações de uma organização ajudam a criar e aplicar ações no interior de seu ambiente organizacional. Devem ser fornecidas por escrito e atualizadas para incluir qualquer alteração desse ambiente operacional.

Regras de negócio escritas adequadamente são utilizadas para definir entidades, atributos, relacionamentos e restrições.

O processo de identificar e documentação de regras de negócio é essencial para o projeto de banco de dados, pois:

- Auxiliam a estandardizar a visualização dos dados de uma empresa;
- Podem constituir uma ferramenta de comunicação entre os usuários e os projetistas;
- Permitem que o projetista compreenda a natureza, o papel e o escopo dos dados;
- Permitem que o projetista compreenda os processos comerciais;
- Permitem que o projetista desenvolva regras e restrições adequadas de participações em relacionamentos e crie um modelo de dados preciso.

DIAGRAMA ENTIDADE-RELACIONAMENTO (DER)

O MER é um modelo conceitual, já o Diagrama Entidade Relacionamento (Diagrama ER - DER) é sua forma de representar graficamente, sua ferramenta de representação. O diagrama permite a visualização das informações, pois o modelo pode ficar muito abstrato sem estar representado graficamente. Para criar a representação gráfica, existem algumas regras, conhecidas como notações do DER.

Sumário da Notação do Diagrama Entidade-Relacionamento (DER)

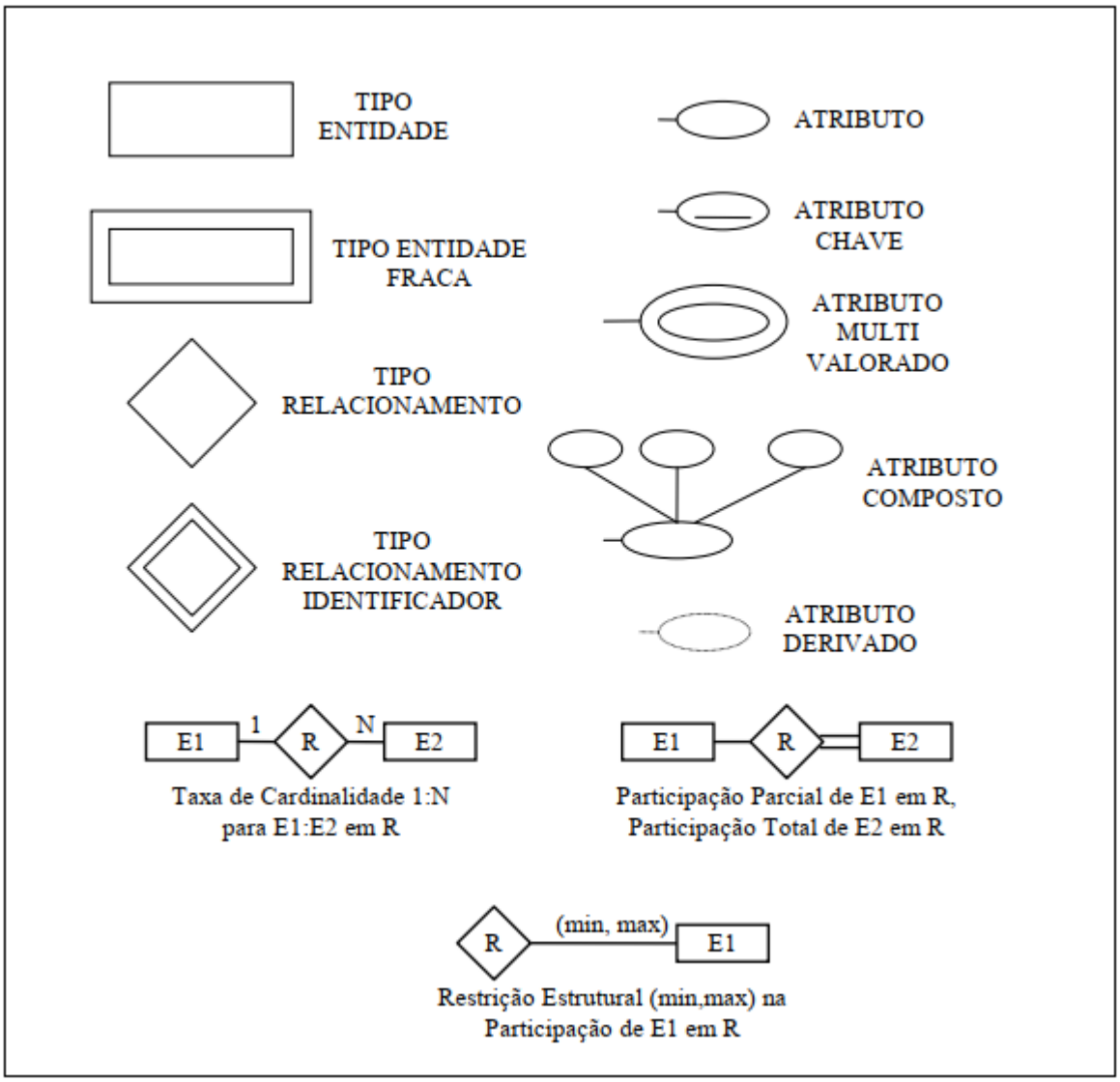


Figura 5: Notações do DER / Fonte: Unimar.

Exemplo de implementação de um sistema, representado em um DER

O DER ilustra o esquema da base de dados COMPANHIA. Os tipos de entidades tais como EMPREGADO, DEPARTAMENTO e PROJETO são mostrados em retângulos.

Tipos de relacionamentos tais como TRABALHA-PARA, GERENCIA, CONTROLA e TRABALHA-EM são mostrados em losangos interligados a tipos de entidades participantes.

Atributos são mostrados em elipses conectadas a tipos de entidades ou relacionamentos. Os componentes de um atributo composto são também representados em elipses, porém conectadas ao atributo ao qual eles pertencem (atributo Nome de EMPREGADO).

Atributos multivalorados são denotados em elipses com linhas duplas (atributo Localização de DEPARTAMENTO). Os atributos-chaves são sublinhados. Atributos derivados em elipses com linhas tracejadas (atributo NumeroDeEmpregados de DEPARTAMENTO).

Os tipos de entidades-fracas são distinguidos por retângulos com linhas duplas e os relacionamentos de identificação por losangos com linhas duplas (tipo de entidade-fraca DEPENDENTE e tipo de relacionamento de identificação DEPENDENTE-DE).

A chave-parcial de um tipo de entidade-fraca é sublinhada com linha tracejada.

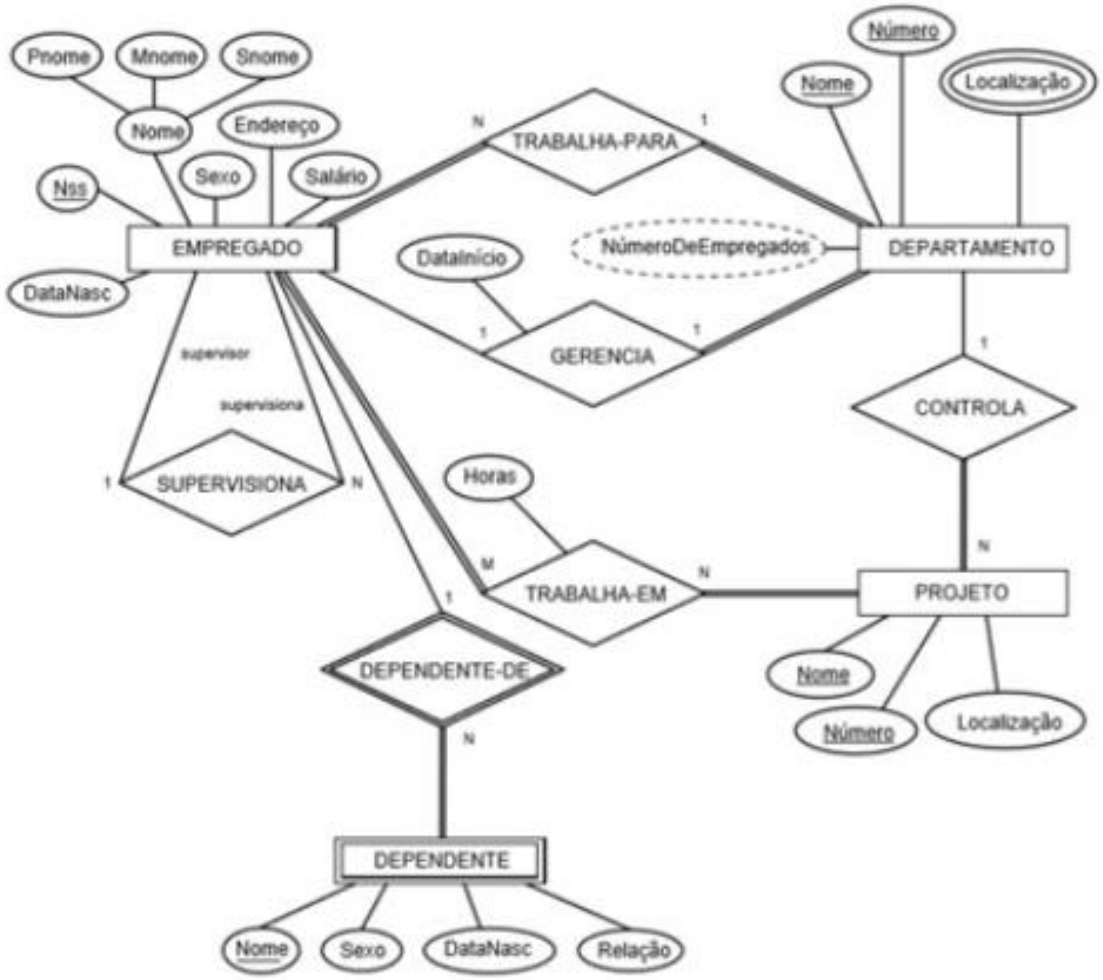


Figura 6: Exemplo de DER / Fonte: ESSA.

Na Figura são mostradas as razões de cardinalidade para cada tipo de relacionamento binário.

A razão de cardinalidade de DEPARTAMENTO:EMPREGADO em GERÊNCIA é 1:1, para DEPARTAMENTO:EMPREGADO em TRABALHA-PARA é 1:N e M:N para TRABALHAEM.

As restrições de participação parcial são especificadas por linhas simples. As linhas paralelas denotam participação total (dependência existencial). Na Figura foram mostrados os nomes de papéis para o tipo de relacionamento SUPERVISIONA porque o tipo de entidade EMPREGADO ocupa dois papéis neste relacionamento.

Atividades

1. Qual o conceito de entidade?

2. Como são representadas as entidades no modelo entidade relacionamento?

3. Crie três entidades abstratas e três entidades concretas?

4. O que são relacionamentos?

5. Explique as restrições de dados?

6. O que é uma instância de entidade?

7. Quais são os tipos de relacionamentos disponíveis?

8. Explique os relacionamentos bidirecionais?

9. Explique as regras de negócio?

10. Os atributos são responsáveis por qual função no MER?

TEMA 4: REGRAS DE NORMALIZAÇÃO

Formas Normais

A normalização de tabelas tem como objetivo resolver problemas de atualização de bases de dados, minimizando redundâncias. É um processo em que são eliminados esquemas de relações (tabelas) não satisfatórios, decompondo-os, através da separação de seus atributos em esquemas de relações menos complexas, mas que satisfaçam as propriedades desejadas.

Um banco de dados dentro dos padrões de normalização reduz o trabalho de manutenção e ajuda a evitar o desperdício do espaço de armazenamento.

O processo de normalização conduz um esquema de relação através de vários testes para garantir o relacionamento analisado está na 1ª, 2ª e 3ª **Formas Normais**.

Cada etapa ou teste corresponde uma determinada forma normal, que representa um progressivo refinamento na estrutura das tabelas. Assim, uma tabela que se encontra na Terceira Forma Normal é considerada mais normalizada (mais "enxuta", pode-se dizer) que uma tabela que se encontra apenas na Segunda Forma Normal.

A normalização possui caráter organizativo e pode ocorrer durante a concepção do modelo conceitual, durante a derivação do modelo lógico para o relacional, ou após a derivação do modelo lógico.

As principais características de uma base de dados normalizada são:

- Geração de aplicações mais estáveis.
- Aumento do número de tabelas utilizadas.
- Diminuição dos tamanhos médios das tabelas.

Benefícios da normalização:

- Estabilidade do modelo lógico: capacidade de um modelo manter-se inalterado face às mudanças que venham a ser percebida ou introduzidas no ambiente que tenha sido modelado.
- Flexibilidade: capacidade de adaptação a demandas diferenciadas, a expansão e redução, a omissão ou presença etc.
- Integridade: diz respeito à qualidade do dado. Um dado mapeado em mais de um local de modo diferente, com valores instanciados de modo diferentes, pode ser indício de que não há integridade entre eles.
- Economia: no espaço de armazenamento em relação ao custo de manipulação de dados (que representa todo e qualquer esforço, tempo, ou valor agregado ao fato de manipularmos volumes de dados maiores do que os efetivamente necessários); custo causado pelo atraso do

fornecimento da informação desejada.

- Fidelidade ao ambiente observado: ajuda a definir elementos que foram despercebidos durante o processo de modelagem.

Primeira Forma Normal – 1FN

Uma relação se encontra na primeira forma normal se todos os domínios de atributos possuem apenas valores atômicos (simples e indivisíveis), e que os valores de cada atributo na tupla seja um valor simples, ou seja, a 1FN não permite que haja em um atributo um conjunto de valores ou uma tupla de valores, em outras palavras não são aceitos *relações* dentro de *relações*. Assim sendo todos os atributos compostos e multivalorados devem ser divididos em atributos atômicos.

Passos para obtenção da 1FN:

- Remova o(s) atributo(s) que viola(m) a primeira forma normal e coloque-o em uma relação em separado (Autores_Livros), juntamente com a chave primária da relação original (IdLivro). A chave primária da nova relação é composta da chave primária da relação original mais uma chave candidata dos atributos da nova relação.
- Poderão ser utilizados tantos atributos quantos forem necessários à formação da chave da nova relação, mesmo que tenhamos que chegar a concatenação de todos os atributos de uma linha. Na prática, o grande número de colunas formando a chave pode causar transtornos, por isso, nesse caso, pode-se criar uma coluna-artificial do tipo contador.

Exemplo de 1FN: Livros

Tabela de Livros: Estrutura original

IdLivro	Título	Assunto	Autor1	Autor2	Autor3
21237	Os Sertões	Ficção	E. Cunha		
33455	Eletricidade básica	Física	A. Silva	B. Santos	
12312	Atlas do Brasil	Geografia	IBGE		

Estrutura normalizada na 1FN:

Tabela de Livros

IdLivro	Título	Assunto
21237	Os Sertões	Ficção
33455	Eletricidade básica	Física
12312	Atlas do Brasil	Geografia

Tabela Autores_Livros

IdLivro	Autor
21237	E. Cunha
33455	A. Silva
33455	B. Santos
12312	IBGE

Exemplo de 1FN: Clientes

Tabela de Clientes: Estrutura original

Codigo	Nome	Telefone	Tipo_tel	Rua	No	Cidade
00001	Maria	3441 8566	Residencial	Contorno	2316	Belo Horizonte
00001	Maria	3215 8751	Serviço	Contorno	2316	Belo Horizonte
00001	Maria	9158 3239	Celular	Contorno	2316	Belo Horizonte
00002	Antônio	8874 5698	Celular	Afonso Pena	5693	Belo Horizonte

Estrutura normalizada na 1FN:

Tabela: Clientes

Codigo	Nome	Rua	No	Cidade
00001	Maria	Contorno	2316	Belo Horizonte
00002	Antônio	Afonso Pena	5693	Belo Horizonte

Tabela: Telefone_Clientes

Codigo	Telefone	Tipo_tel
00001	3441 8566	Residencial
00001	3215 8751	Serviço
00001	9158 3239	Celular
00002	8874 5698	Celular

Segunda Forma Normal – 2FN

Uma relação encontra-se na segunda forma normal quando estiver na primeira forma normal e todos os atributos que não participam da chave primária são dependentes desta.

Assim devemos verificar se todos os atributos são dependentes da chave primária e retirar-se da relação todos os atributos de um grupo não dependente que dará origem a uma nova relação, que conterá esse atributo como não chave. Desta maneira, na segunda forma normal evita inconsistências devido a duplicidade.

Tabela de Empregado_Projeto: Estrutura original

Num_emp	Num_proj	Horas	Nome_emp	Nome_proj	Local_proj
00001	001	8	Maria	Versão Evolutiva 3.22	João Monlevade
00002	001	18	José	Versão Evolutiva 3.22	João Monlevade
00003	002	12	Samara	Versão Corretiva 3.21	Belo Horizonte

Estrutura normalizada - 2FN

Tabela: projetos

Num_proj	Nome_proj	Local_proj
001	Versão Evolutiva 3.22	João Monlevade
002	Versão Corretiva 3.21	Belo Horizonte

Tabela: Empregado_projeto

Num_emp	Num_proj	Horas
00001	001	8
00002	001	18
00003	002	12

Tabela: Empregado

Num_emp	Nome_emp
00001	Maria
00002	José
00003	Samara

Terceira Forma Normal – 3FN

Para estar na terceira forma normal a tabela não pode ter atributos não-chave se referindo a outros atributos não-chave. Assim devemos verificar se existe um atributo que não depende diretamente da chave, retirá-lo criando uma relação que conterá esse grupo de atributos, e defina com a chave, os atributos dos quais esse grupo depende diretamente. O processo de normalização deve ser aplicado em uma relação por vez, pois durante o processo de normalização vamos obtendo quebras, e, por conseguinte, novas relações. No momento em que o sistema estiver satisfatório, do ponto de vista do analista, este processo iterativo é interrompido.

Exemplo de 3FN: Empregados trabalhando em departamentos

Tabela de Empregado_Depto: Estrutura original

Num_emp	Nome	Data_nasc	Num_Depto	Nome_Depto	Emp_Ger_Depto
00001	Maria	06/03/1977	001	Homologação	018
00002	José	27/05/1973	002	Homologação	018
00003	Samara	24/08/1984	003	Desenvolvimento	005

Tabela: Empregado_Depto

Num_emp	Nome	Data_nasc	Num_Depto
00001	Maria	06/03/1977	001
00002	José	27/05/1973	002
00003	Samara	24/08/1984	003

Tabela: Departamento

Num_Depto	Nome_Depto	Emp_Ger_Depto
001	Homologação	018
002	Homologação	018
003	Desenvolvimento	005

Resumo da normalização

O quadro abaixo descreve como realizar teste para identificação da desnormalização da relação e procedimento para normalização.

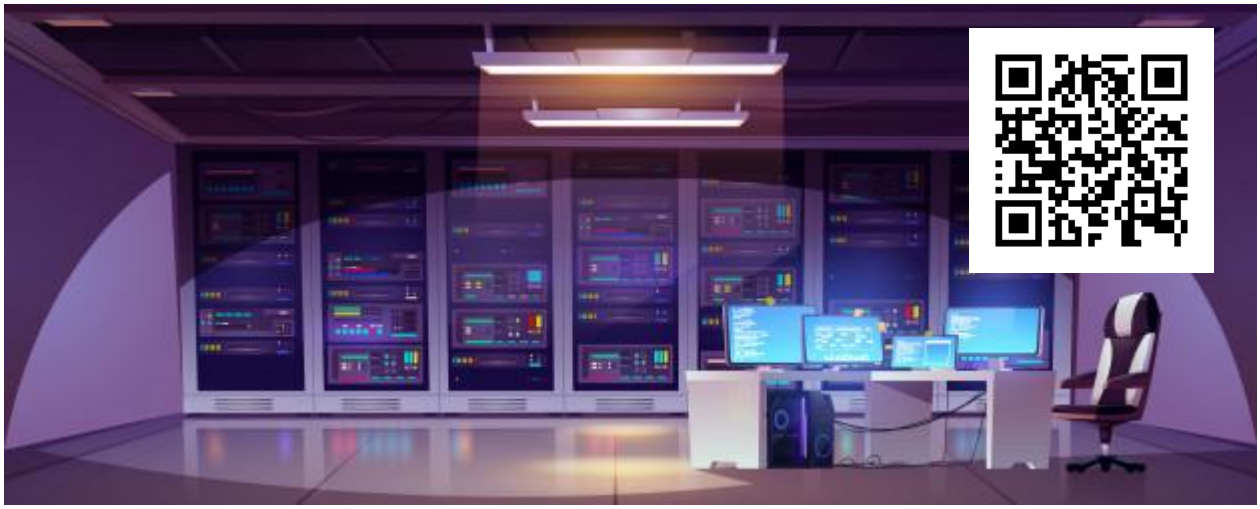
Formal Normal	Teste	Solução (normalização)
1FN	A relação não deve ter qualquer atributo não-atômico nem relações agrupadas.	Forme novas relações para cada atributo não-atômico ou relação aninhada.
2FN	Para relações nas quais a chave primária contém múltiplos atributos, nenhum atributo não-chave deve ser funcionalmente dependente de uma parte da chave primária.	Decomponha e monte uma relação para cada chave parcial com seu(s) atributo(s) dependente(s). Certifique-se de manter uma relação com a chave primária original e quaisquer atributos que sejam completamente dependentes dela em termos funcionais.
3FN	A relação não deve ter um atributo não-chave funcionalmente determinado por um outro atributo não -chave (ou por um conjunto de atributos não-chave). Ou seja, não deve haver dependência transitiva de um atributo não chave na chave primária.	Decomponha e monte uma relação que inclua o(s) atributo(s) não-chave que funcionalmente determine(m) outros atributos não-chave.

Questões

1. O que é normalização?

2. Para que é usada a técnica de normalização?
3. Há alguma vantagem para um banco de dados normalizado?
4. Quais as características do banco de dados normalizado?
5. O que é a 1ª forma normal?
6. O que é a 2ª forma normal?
7. O que é a 2ª forma normal?
8. Quando termina o processo de normalização?
9. Qual ação deve ser tomada quando há relações dentro de relações?
10. O que fazer quando há dependência transitiva de um atributo não chave na chave primária?

TEMA 5: INTRODUÇÃO À LINGUAGEM SQL – CRIAÇÃO DE BANCO DE DADOS E TABELAS



A "Linguagem Estruturada de Consultas", conhecida como SQL, é utilizada para interação com o SGBD e executar várias tarefas como criar bancos de dados, tabelas, campos e valores, assim como inserir e alterar registros, criar objetos no banco de dados, gerenciar usuário, consultar informações, controlar operações. Todas as transações efetuadas no banco de dados podem ser requisitadas ao SGBD utilizando a linguagem SQL.

A programação SQL pode ser usada para analisar ou executar tarefas em tabelas, principalmente através dos seguintes comandos:

- inserir (*'insert'*),
- pesquisar (*'search'*),
- atualizar (*'update'*) e
- excluir (*'delete'*).

O SQL pode, ainda, fazer consultas e análises mais avançadas, como escrever *queries* (comandos de consulta) com múltiplas informações.

Tipos de dados básicos

Em banco de dados relacionais, cada tabela pode conter diversas colunas, as quais armazenarão os dados. Para cada coluna, existirá um tipo de dado associado. Os tipos de dados são definidos durante a criação da tabela. Os principais tipos de dados simples definidos:

Tipos de Dados	Descrição
CHARACTER	Caractere de tamanho fixo – usualmente conhecido como CHAR
CHARACTER VARYING	Caractere de tamanho variante – usualmente conhecido como VARCHAR
CHARACTER LARGE OBJECT	Caractere longo – usualmente conhecido como CLOB
BINARY LARGE OBJECT	String binária para objetos longos – usualmente conhecido como BLOB
NUMERIC	Numérico exato
DECIMAL	Numérico exato
SMALLINT	Numérico exato
INTERGER	Numérico exato
BIGINT	Numérico exato
FLOAT	Numérico aproximado
REAL	Numérico aproximado
DOUBLE PRECISION	Numérico aproximado
BOOLEAN	Booleano
DATE	Data com informações de dia, mês e ano
TIME	Hora com informações de hora, minuto e segundo
TIMESTAMP	Determina um momento, com informações de ano, mês, dia, hora, minuto e segundo

Figura 7: Tipos de dados/ Fonte: Costa(2006)

O comando create database

A instrução create database, como o próprio nome sugere, serve para a criação da base de dados na qual as tabelas serão criadas, com uma sintaxe simples.

CREATE DATABASE - Cria um novo banco de dados MySQL.
Para criar um banco de dados, você deve ser um superusuário ou ter o privilégio especial CREATEDB. Por padrão, o novo banco de dados será criado baseado no modelo de banco de dados de sistema template1. Um modelo diferente pode ser especificado escrevendo o nome do TEMPLATE. Em particular, ao escrever o TEMPLATE0 template0, você pode criar um banco de dados virgem contendo apenas os objetos padrão predefinidos pela sua versão do MySQL. Isso é útil se você desejar evitar a cópia de qualquer objeto local de instalação que possa ter sido incluído no template1.

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ LC_COLLATE [=] lc_collate ]
           [ LC_CTYPE [=] lc_ctype ]
           [ TABLESPACE [=] tablespace ]
           [ CONNECTION LIMIT [=] connlimit ] ]
```
Parâmetros

- name: O nome de um banco de dados para criar.
- user_name: O nome da função do usuário que possuirá o novo banco de dados, ou DEFAULT para usar o padrão (ou seja, o usuário que está executando o comando). Para criar um banco de dados de propriedade de outra função, você deve ser um membro direto ou indireto dessa função

ou ser um superusuário.

- template: O nome do template a partir do qual criar o novo banco de dados, ou DEFAULT para usar o template padrão (template1).
- encoding:Codificação do conjunto de caracteres para usar no novo banco de dados. Especifique uma constante de cadeia (por exemplo, 'SQL_ASCII'), ou um número de codificação de inteiro, ou DEFAULT para usar a codificação padrão (ou seja, a codificação do banco de dados de modelo).
- lc_collate: Ordem de agrupamento (LC_COLLATE) para usar no novo banco de dados. Isso afeta a ordem de classificação aplicada às strings, por exemplo, em consultas com ORDER BY, bem como a ordem usada em índices em colunas de texto. O padrão é usar a ordem de agrupamento do banco de dados de modelos.
- lc_ctype: Classificação de caracteres (LC_CTYPE) para usar no novo banco de dados. Isso afeta a categorização de caracteres, por ex. inferior, superior e dígito. O padrão é usar a classificação de caracteres do banco de dados de modelos.
- tablespace: O nome do espaço de tabela que será associado ao novo banco de dados, ou DEFAULT para usar o espaço de tabela do banco de dados de modelo. Esse espaço de tabela será o espaço de tabela padrão usado para objetos criados neste banco de dados.
- connlimit: Quantas conexões simultâneas podem ser feitas neste banco de dados. -1 (o padrão) significa sem limite.

Parâmetros podem ser desenvolvidos na ordem que for necessária, não apenas na ordem mostrada acima.

O comando create table

Após a criação da base de dados, é necessário criar as tabelas. Para isso, o comando create table é utilizado e permite criar e definir a estrutura de uma tabela. Nele pode-se também, definir as colunas (campos), suas respectivas restrições, além de suas chaves primárias e estrangeiras.

O comando CREATE TABLE define uma nova tabela, inicialmente vazia (sem nenhum registro), no esquema de banco de dados atual. A tabela criada pertence ao usuário que executa o comando (usuário logado).

É possível especificar um esquema diferente do esquema atual para a criação da tabela, bastando para isso identificar o esquema junto ao nome desejado para a tabela (separado por um ponto.), usando uma sintaxe do tipo *nome_esquema.nome_tabela*.

A sintaxe para a criação de uma tabela é:

Sintaxe básica:

```
CREATE TABLE [IF NOT EXISTS] nome_tabela (
  nome_coluna tipo_dados [COLLATE colação] constraint,
  nome_coluna tipo_dados constraint,
  nome_coluna tipo_dados constraint,
  ...,
[FOREIGN KEY chave_estrangeira REFERENCES coluna]
[ON DELETE ação] [ON UPDATE ação]
)
```

Se não for usada a opção COLLATE, a colação padrão será utilizada. Os itens entre colchetes [ ] são opcionais na criação de uma tabela. A cláusula IF NOT EXISTS verifica se a tabela com o nome especificado já existe antes de tentar criá-la.

As constraints (testes) possíveis para colunas são:

- NOT NULL

- NULL (padrão)
- CHECK
- DEFAULT
- UNIQUE
- PRIMARY KEY
- REFERENCES

E as constraints específicas para tabelas são:

- CHECK
- UNIQUE
- PRIMARY KEY
- EXCLUDE (extensão específica do MySQL)
- FOREIGN KEY

Existem na verdade duas formas de definir constraints. É possível definir contraints de coluna e constraints de tabela.

Uma constraint de coluna é definida como parte da definição da coluna em si. Já uma constraint de tabela não é vinculada a nenhuma coluna em particular, e pode ser aplicada a mais de uma coluna ao mesmo tempo.

A cláusula check serve para implementar restrições de domínio. Por exemplo, durante a criação de uma tabela projeto, inserimos uma restrição que garante que a data de início do projeto seja menor que a data prevista de término. A cláusula check também poderia ser usada para comparar um atributo com um valor absoluto e não apenas para comparar um atributo com outro atributo.

Exemplo de criação de tabelas no MySQL:

Exemplo 1

Vamos criar uma tabela de nome tabela_autores, que irá conter os campos COD_Autor, Nome_Autor, Sobrenome_Autor e Data_Nasc:

```
CREATE TABLE tabela_autores (
codigo_Autor int(4) UNIQUE AUTO_INCREMENT,
nome_Autor varchar(30) NOT NULL,
sobrenome_Autor varchar(50),
data_Nasc date,
PRIMARY KEY (codigo_Autor)
);
```

Exemplo 2

Agora iremos criar a tabela de editoras, com os campos COD_Editora e Nome_Editora:

```
CREATE TABLE tabela_editoras(
cod_Editora int,
Nome_Editora varchar(35) UNIQUE NOT NULL,
PRIMARY KEY (cod_Editora)
);
```

Exemplo 3

Vamos criar também uma tabela para armazenar os gêneros literários:

```
CREATE TABLE tabela_generos (
```

```
COD_Genero integer,
Nome_Genero varchar(40) UNIQUE NOT NULL,
PRIMARY KEY (COD_Genero)
);
```

Exemplo 4

Finalmente, vamos criar a tabela de livros, incluindo os relacionamentos com as demais tabelas por meio do uso de chaves estrangeiras:

```
CREATE TABLE tabela_livros (
      COD_Livro int UNIQUE NOT NULL,
Nome_Livro varchar(50) NOT NULL,
Autor int NOT NULL,
Editora int NOT NULL,
      Data_Pub date,
Genero int NOT NULL,
Preco_Livro float,
FOREIGN KEY (Autor) REFERENCES tabela_autores (codigo_Autor) ON DELETE
CASCADE,
      PRIMARY KEY (COD_Livro)
);
```

A palavra SERIAL na criação da coluna COD_Livro, usa o auto-incremento nesta coluna. Isso significa que os CODs dos livros serão números gerados automaticamente pelo MySQL, de modo que não precisaremos informar esse dado ao cadastrar os livros, o que faremos no próximo artigo. Além disso, não precisamos informar o tipo de dados desta coluna, visto que o auto-incremento é sempre realizado com números inteiros (integer).

Após criar as tabelas podemos descrevê-las individualmente com o comando \d, como segue: \d nome_tabela

```
\d tabela_livros
\d tabela_autores
\d tabela_editoras
\d tabela_generos
```

Outros exemplos de Create Table:

```
CREATE TABLE UNIQUE
CREATE TABLE Pessoa ( ID int NOT NULL,
Sobrenome varchar(255) NOT NULL, Nome varchar(255),
Age int, UNIQUE (ID)
);
CREATE TABLE PRIMARY KEY
CREATE TABLE Pessoa (
ID int NOT NULL,
Sobrenome varchar(255) NOT NULL, Nome varchar(255),
Age int,
PRIMARY KEY (ID)
);
CREATE TABLE FOREIGN KEY
CREATE TABLE Pedidos (
```

```
PedidoID int NOT NULL, PedidoNumr int NOT NULL, PessoaID int,
PRIMARY KEY (PedidoID),
FOREIGN KEY (PessoaID) REFERENCES Pessoa(PessoaID)
);

CREATE TABLE DEFAULT
CREATE TABLE Pessoa ( ID int NOT NULL,
Sobrenome varchar(255) NOT NULL, Nome varchar(255),
Idade int,
Cidade varchar(255) DEFAULT 'Sandnes'
);

CREATE TABLE check
CREATE TABLE Pessoa ( ID int NOT NULL,
Sobrenome varchar(255) NOT NULL, Nome varchar(255),
Idade int,
CHECK (Idade>=18)
);
```

INSERINDO DADOS NAS TABELAS CRIADAS NO MYSQL:
O comando INSERT permite incluir novos dados dentro de uma tabela.

Sintaxe:

```
INSERT INTO nome_tabela
VALUES (valores);
```

ou I

```
INSERT INTO nome_tabela (campos)
VALUES (valores);
```

Exemplo

```
INSERT INTO tabela_autores VALUES (001, 'Carlos, 'Drumond de Andrade',
'1902-10-31');
ou
INSERT INTO tabela_autores (COD_Autor, Nome_Autor, Sobrenome_Autor,
Data_Nasc) VALUES (001, 'Carlos, 'Drumond de Andrade', '1902-10-31');
```

Questões

1. O que significa SQL?
2. O que é a Linguagem SQL?
3. Qual o comando de criação de banco de dados?
4. O que é Template na criação de um banco de dados??
5. O que faz a instrução tablespace?
6. Qual o comando de criação das tabels?
7. É possível especificar um esquema diferente do esquema atual para a criação
8. A sintaxe para a criação de uma tabela é:
9. O que é uma constraint de coluna?
10. O que faz a cláusula check?

TEMA 6: MODELAGEM EM DML

As discussões anteriores sobre o modelo relacional contemplaram apenas os aspectos estruturais. Agora, a atenção será voltada para a álgebra relacional, que é uma coleção de operações utilizadas para manipular relações. Essas operações são usadas para selecionar tuplas de uma determinada relação ou para combinar tuplas relacionadas a diversas relações com o propósito de especificar uma consulta, uma requisição de recuperação sobre a base de dados.

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

SELECT

O operador SELECT é unário; isto é, ele é aplicado somente a uma relação. Assim, o SELECT não pode ser usado para selecionar tuplas de mais de uma relação. Observe também que a operação de seleção é aplicada individualmente para cada tupla. Assim, as condições de seleção não podem ser aplicadas a mais que uma tupla. O grau da relação resultante é a mesma que a relação original.

O número de tuplas da relação resultante é sempre menor ou igual ao número de tuplas da relação original. O comando SELECT permite recuperar os dados de um objeto do banco de dados, como uma tabela, view e, em alguns casos, uma stored procedure (alguns bancos de dados permitem a criação de procedimentos que retornam valor).

A sintaxe mais básica do comando é:

```
SELECT <lista_de_campos>
FROM <nome_da_tabela></nome_da_tabela></lista_de_campos>;
```

Exemplo:

```
SELECT CODIGO, NOME FROM CLIENTES;

SELECT * FROM CLIENTES;
```

É possível selecionar o conteúdo de uma tabela parcial colocando restrições para as linhas a serem incluídas no resultado. Isto é feito com a utilização da cláusula WHERE para adicionar restrições condicionais ao comando SELECT. A sintaxe a seguir permite especificar quais linhas serão selecionadas:

```
SELECT <lista_de_campos>
FROM <nome_da_tabela></nome_da_tabela></lista_de_campos>
WHERE  <lista de condições>;
```

O comando SELECT recupera todas as linhas que atendam às condições especificadas – também conhecidas como critérios condicionais – na cláusula where. A lista de condições em where é uma representação por uma ou mais expressões condicionais, separadas por operadores lógicos.

A cláusula where é opcional. Se nenhuma linha atender aos critérios especificados nesta cláusula, o usuário verá uma tela em branco ou uma mensagem dizendo que nenhuma linha foi retornada.

Operadores SQL

Os operadores aritméticos desempenham operações matemáticas em duas expressões de um ou mais dos tipos de dados da categoria de tipo de dados numéricos.

Operador	Significado
+ (Somar)	Adição
- (Subtrair)	Subtração
* (Multiplicar)	Multiplicação
/ (dividir)	Divisão

% (módulo) Retorna o resto inteiro de uma divisão. Por exemplo, 12% 5 = 2 porque o resto de 12 dividido por 5 é 2.
= (Operador de atribuição)
O sinal de igual (=) é o único operador de atribuição Transact-SQL.

O operador de atribuição também pode ser usado para estabelecer a relação entre um título de coluna e a expressão que define os valores para a coluna.

```
SELECT FirstColumnHeading = 'xyz',
SecondColumnHeading = ProductID
FROM Production.Product;
```

Operadores compostos

Os operadores compostos executam alguma operação e definem um valor original para o resultado da operação.

Operador	Link para mais informações	Ação
+=	+= (Atribuição de adição) (Transact-SQL)	Adiciona alguma quantidade ao valor original e define o valor original como resultado.
-=	-= (Subtract Assignment) (Transact-SQL)	Subtrai alguma quantidade do valor original e define o valor original como resultado.
*=	*= (Atribuição de multiplicação) (Transact-SQL)	Multiplica por uma quantidade e define o valor original para o resultado.
/=	(Divide Assignment) (Transact-SQL)	Divide por uma quantidade e define o valor original para o resultado.
%=	Atribuição de módulo (Transact-SQL)	Divide por uma quantidade e define o valor original para o módulo.
&=	&= (Atribuição de AND bit a bit) (Transact-SQL)	Executa um AND bit a bit e define o valor original como o resultado.
^=	^= (Atribuição de OR exclusivo bit a bit) (Transact-SQL)	Executa um OR bit a bit exclusivo e define o valor original como o resultado.
|=	|= (Atribuição de OR bit a bit) (Transact-SQL)	Executa um OR bit a bit e define o valor original como o resultado.

Operadores lógicos

Operador	Significado
ALL	TRUE se tudo em um conjunto de comparações for TRUE.
AND	TRUE se as duas expressões boolianas forem TRUE.
ANY	TRUE se qualquer conjunto de comparações for TRUE.
BETWEEN	TRUE se o operando estiver dentro de um intervalo.
EXISTS	TRUE se uma subconsulta tiver qualquer linha.
IN	TRUE se o operando for igual a um de uma lista de expressões.
LIKE	TRUE se o operando corresponder a um padrão.
NOT	Inverte o valor de qualquer outro operador booliano.
OR	TRUE se qualquer expressão booliana for TRUE.
SOME	TRUE se algum conjunto de comparações for TRUE.

UPDATE

O comando UPDATE é responsável por alterar um ou mais registros de uma tabela, dependendo de suas condições e é claro respeitando as restrições da tabela, sintaxe:

```
UPDATE  <nome_tabela>
SET <nome_coluna> = <novo_valor>
      [, <nome_coluna1> = <novo_valor1>]
[WHERE <condição>]
```

DELETE

O comando DELETE nos permite remover um ou mais registros de uma tabela, sintaxe:

```
DELETE [FROM] <nome_tabela>
[WHERE <condição>]
```

Excluindo dados de uma tabela

O comando delete é utilizado para excluir linhas de uma tabela. No exemplo vamos apagar os pedidos que tenham seu número acima de 1000.

Exemplo

```
DELETE FROM Pedidos
WHERE PedidoNumr>=1000
```

Questões

1. Qual o significado de DML?
2. Quais são os comandos DML?
3. O que é um operador de atribuição?
4. Quais são as cláusulas obrigatórias no comando select?
5. Para que servem os operadores compostos?
6. O que faz o comando Update?
7. Dê um exemplo.
8. O que faz o comando INSERT?
9. Apresente um exemplo de como é possível usar um operador numa busca.
10. O que faz o comando DELETE?

TEMA 7: CONDIÇÕES DE FILTRAGEM EM BUSCAS SQL

WHERE

A cláusula WHERE permite aplicar uma condição para que seja realizado o comando SQL. O objetivo dele é fazer a filtragem dos dados determinados pelos comandos SELECT, UPDATE e DELETE.A condição é construída através de uma comparação entre dois valores, utilizando os operadores relacionais.

Operadores Relacionais

Para aplicar a condição (ou filtro) podem ser utilizados os operadores de comparação: =, >, <, >=, <= e <> (diferente). A tabela mostra os operadores de comparação que são usados:

Símbolo	Operação
=	Igualdade
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
<>	Diferente

Figura 8: Exemplo de DER / Fonte: ESSA.

Exemplos: para saber se um determinado valor está abaixo de 100, podemos utilizar a comparação: valor < 100. Podemos testar se o código é igual

GROUP BY

Os dados resultantes de uma seleção podem ser agrupados de acordo com um critério específico. Este procedimento é realizado usando a cláusula group by.

Exemplo

```
SELECT COUNT(PedidoNumr) FROM Pedidos GROUP BY PedidoNumr
```

Junções (join)

Quando precisamos realizar consultas que envolvam mais de uma tabela, uma das soluções seria a utilização de junções. As junções permitem que acessemos mais de uma tabela utilizando apenas um SELECT.

Junção interna (inner Join)

A junção interna entre tabelas é a modalidade de junção que faz com que somente participem da relação resultante as linhas das tabelas de origem que atenderem à cláusula de junção.

```
SELECT * FROM tabela_Livros
INNER JOIN tabela_autores
ON tabela_Livros.ID_Autor = tabela_autores.ID_Autor;
```

Junções externas (outer join)

Na junção externa, os registros que participam do resultado da junção não obedecem obrigatoriamente à condição de junção, ou seja, a não inexistência de valores correspondentes não limita a participação de linhas no resultado de uma consulta.

Sintaxe:

```
SELECT coluna
FROM tabela_esq
RIGHT (OUTER) JOIN tabela_dir
ON tabela_esq.coluna=tabela_dir.coluna
WHERE tabela_esq.coluna IS NULL;
```

Exemplo:

```
SELECT * FROM tabela_Livros
RIGHT JOIN tabela_editoras
ON tabela_Livros.ID_editora = tabela_editoras.ID_editora
WHERE tabela_Livro.ID_editora IS NULL;
```

COMANDO: SELECT...INTO

Esse comando é usado para armazenar o resultado de uma consulta em uma variável.

```
SELECT Nome, Sobrenome INTO Relacao_Autores
FROM tabela_Autores
where ID = ID_Autor
```

O resultado da consulta deve ter sempre como retorno somente uma linha, caso o resultado tenha mais de uma linha, deve ter o mesmo número de variáveis para receber esses valores.

Questões

1. Crie dois comandos utilizando a cláusula between.
2. Qual o significado da cláusula where?
3. Crie dois comandos utilizando a cláusula where.
4. Crie um comando delete usando a cláusula where.
5. Crie um comando usando a cláusula set.
6. Apresente um exemplo usando group by.
7. Apresente um exemplo usando inner join.
8. Apresente um exemplo usando Select into.
9. Como armazenar o conteúdo de uma busca em uma variável?
10. Como se pode fazer consultas em mais de uma tabela?

TEMA 8: MODELAGEM DDL

Em tese, uma linguagem de banco de dados permite a criação de banco de dados e estruturas de tabelas para executar tarefas básicas de gerenciamento de dados (adicionar, excluir e modificar) e consultas complexas, transformando dados brutos em informações úteis. Além disso, essa linguagem deve executar essas funções básicas exigindo o menor esforço possível do usuário, com uma estrutura e sintaxe de comandos fáceis de aprender.

Comandos de Definição de Dados.
Os comandos de definição de dados SQL devem ser portáveis, ou seja, adequar-se há um padrão básico para que os usuários não tenham de reaprender o básico quando passarem de um SGBDR para outro. A SQL atende bem a essas exigências ideais de linguagens de banco de dados.

Suas funções de enquadram em duas amplas categorias:

- Linguagem de definição de dados (DDL): A SQL inclui comandos para criar objetos de banco de dados como tabelas, índices e views. Bem como comandos para definir direitos de acesso a esses objetos.

Create schema authorization

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
              | [DEFAULT] COLLATE [=] collation_name
```

Create table

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death
DATE);
```

O comando alter table
O comando alter table permite alterar a estrutura de uma tabela acrescentando, alterando, retirando e

alterando nomes, formatos das colunas e a integridade referencial definidas em uma determinada tabela.

- **nome-tabela** representa o nome da tabela que será atualizada.
- **nome-coluna** representa o nome da coluna que será criada.
- **tipo-do-dado** a cláusula que define o tipo e tamanho dos campos definidos para a tabela.
- **DROP nome-coluna** realiza a retirada da coluna especificada na estrutura da tabela.

- **ADD nome-coluna tipo-do-dado** realiza a inclusão da coluna especificada na estrutura da tabela. Na coluna correspondente a este campo nos registros já existentes será preenchido o valor NULL (Nulo). As definições NOT NULL e NOT NULL WITH DEFAULT são semelhantes à do comando CREATE TABLE.
- **MODIFY nome-coluna tipo-do-dado** permite a alteração na característica da coluna especificada.

Apagando uma coluna de uma tabela

Imagine que você deseja, por alguma razão, apagar a coluna que armazena o nome `Nome_Genero` dos livros da tabela `tabela_generos`.

Exemplo

```
ALTER TABLE tabela_generos
DROP Nome_Genero;
```

Modificando uma coluna de uma tabela

Se precisássemos mudar as características de uma coluna da tabela após a sua criação, usaríamos o comando modify. Como exemplo, imagine que desejamos aceitar valores nulos no atributo

`PedidoNumr` da tabela `Pedidos`.

```
CREATE TABLE Pedidos (
PedidoID int NOT NULL, PedidoNumr int NOT NULL, PessoaID int,
PRIMARY KEY (PedidoID),
FOREIGN KEY (PessoaID) REFERENCES Pessoa(PessoaID)
);
```

Exemplo

```
ALTER TABLE Pedidos
MODIFY PedidoNumr INTEGER NULL;
```

O COMANDO DROP TABLE

O comando drop table serve para destruirmos uma tabela. Se, por exemplo, precisássemos destruir a tabela `Pedidos` ,

Exemplo

`DROP TABLE` Pedidos;

```
ALTER TABLE
```

```
ALTER TABLE table_name
ADD column_name datatype;

ALTER TABLE table_name
ALTER COLUMN column_name datatype;

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

Create table as

```
CREATE TABLE new_table_name AS
    SELECT column1,                          column2,...
    FROM existing_table_name
    WHERE ....;
```

Drop table

```
DROP TABLE table_name;
```

Questões

1. Qual o comando utilizado para alterar uma coluna em uma tabela?
2. Qual o comando utilizado para criar um esquema de banco de dados?
3. Quais são os comandos disponíveis em DDL?
4. Qual o comando utilizado para remover uma tabela de um banco de dados?
5. Qual o comando utilizado para criação de uma tabela em um esquema específico de banco de dados?
6. Qual o comando utilizado para modificar uma coluna de uma tabela?
7. E para alterar um tipo de dados de uma coluna?
8. Qual o comando utilizado para validar as remoções em um banco de dados?
9. Para que serve o comando Drop table?
10. Apresente um exemplo de Drop Table.

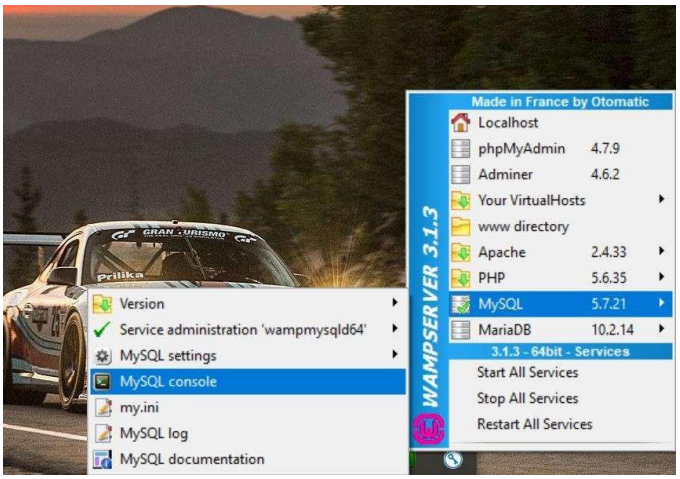
TEMA 9: RECURSOS AVANÇADOS DE DDL

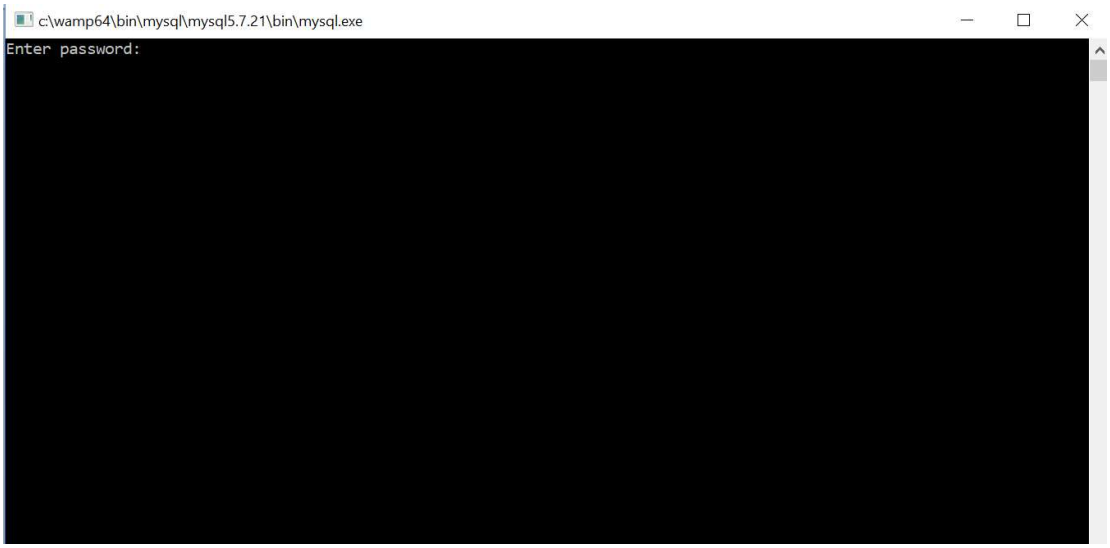
Antes de podermos utilizar um novo SGBDR, é necessário realizar duas tarefas: primeiro, criar a estrutura do banco de dados e segundo, criar as tabelas que manterão os dados do usuário final. Para realizar a primeira tarefa o SGBDR cria os arquivos físicos que conterão o banco de dados.

Recursos avançados de DDL

Quando se utiliza um SGBDR empresarial, antes de começar a criar tabelas é necessário ser autenticado por esse sistema. A autenticação é o processo por meio do qual um SGBD garante que somente usuários registrados possam acessar o banco de dados. Para ser autenticado, deve-se fazer login no SGBDR utilizando um ID de usuário e uma senha criada pelo administrador de banco de dados. Em um SGBDR empresarial todo ID de usuário está associado a um esquema de banco de dados.

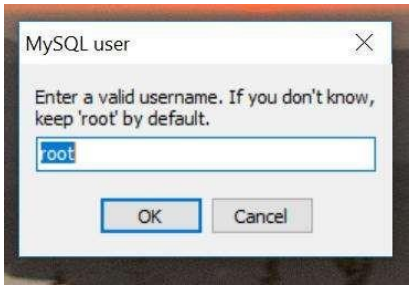
Conectando no MySQL





Esquema de banco de dados:

Quando se utiliza um SGBDR empresarial, antes de começar a criar tabelas é necessário ser autenticado por esse sistema. A autenticação é o processo por meio do qual um SGBD garante que somente usuários registrados possam acessar o banco de dados. Para ser autenticado, deve-se fazer login no SGBDR utilizando um ID de usuário e uma senha criada pelo administrador de banco de dados.



Em um SGBDR empresarial todo ID de usuário está associado a um esquema de banco de dados.No ambiente de SQL esquema é um grupo de objetos de banco de dados como tabelas e índices relacionados entre si. Geralmente o esquema pertence a um único usuário ou aplicação. Um único banco de dados pode manter vários esquemas que pertençam a vários usuários ou aplicações. Pense no esquema como um agrupamento lógico de objetos de banco de dados, como tabelas, índices e views. Eles são úteis, pois agrupam as tabelas por proprietários (ou função) e aplicam o primeiro nível de segurança permitindo que cada usuário veja apenas as tabelas que lhe pertencem.

Os padrões de SQL ANSI definem um comando para a criação de um esquema de banco de dados:

```
CREATE SCHEMA AUTHORIZATION {CRIADOR}
```

Após a criação do esquema de banco de dados, estamos prontos para definir as estruturas das tabelas no banco de dados.

Especificações

Quando for indispensável haver dados disponíveis, a especificação NOT NULL não permitirá que o usuário deixe o atributo vazio (sem nenhuma entrada de dados).

UNIQUE

Restrições são regras que o Mecanismo de Banco de Dados do SQL Server impõe a você. Por exemplo, você pode usar as restrições UNIQUE para garantir que não há valores duplicados inseridos

em colunas específicas que não participam de uma chave primária. Embora a restrição UNIQUE e a restrição PRIMARY KEY impõem exclusividade, use a restrição UNIQUE em vez da restrição PRIMARY KEY quando for impor a exclusividade de uma coluna, ou uma combinação de colunas, que não seja uma chave primária.

Diferente das restrições PRIMARY KEY, as restrições UNIQUE permitem o valor NULL. Porém, como com qualquer valor que participa de uma restrição UNIQUE, só um valor nulo é permitido por coluna. Uma restrição UNIQUE pode ser referenciada por uma restrição FOREIGN KEY.

Quando uma nova restrição UNIQUE é adicionada a uma coluna ou colunas existentes em uma tabela, o Banco de Dados, por padrão, examina os dados existentes nas colunas para certificar-se de que todos os valores são únicos. Se uma restrição UNIQUE for adicionada a uma coluna que tem valores duplicados, o mecanismo de Banco de Dados retornará um erro e não adicionará a restrição.

A especificação UNIQUE cria um índice exclusivo no respectivo atributo. Utilize-o para evitar a duplicidade de valores em uma coluna.

SYNTAX:

```
MySQL> create table tempdate (udatetime datetime, utimestamp
timestamp);
Query OK, 0 rows affected (0.32 sec)
```

Além dos formatos apresentados na tabela, a SQL dá suporte a muitos outros tipos de dados, incluindo TIME(Hora), TIMESTAMP (registro de data e hora), REAL (número real), DOUBLE (duplo), FLOAT (ponto flutuante), e intervalos como INTERVAL, DAY TO HOUR. Muitos SGBDR´s também expandiram a lista para incluir outros tipos de dados como LOGICAL (lógico), CURRENCY (moeda), AUTONUMBER (numeração automática) e sequência.

Permissões dos usuários

O MySQL é um sistema gerenciador de banco de dados de código aberto que ajuda os usuários a armazenar, organizar e posteriormente recuperar dados.
Ele possui diversas opções para garantir a usuários específicos permissões dentro de tabelas e bancos de dados.

Quando um usuário é criado, não possui permissões para fazer nada com os bancos de dados. Na verdade, mesmo se tentar fazer login (com a senha password), ele não será capaz de chegar ao shell do MySQL.

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
```

Portanto, a primeira coisa a ser feita é fornecer ao usuário o acesso às informações que eles irão precisar.

```
GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';
```

Os asteriscos do comando referem-se ao banco de dados e tabela que ele pode acessar . Este comando permite que o usuário leia, edite, execute e realize todas as tarefas em todos os bancos de dados e tabelas. Observe que, nesse exemplo, estamos concedendo ao usuáirio acesso root completo a tudo em nosso banco de dados.

Outras permissões dos usuários

- ALL PRIVILEGES — como vimos anteriormente, isso garante ao usuário do MySQL acesso completo a um banco de dados (ou, se nenhum banco de dados for selecionado, acesso global a todo o sistema)
- CREATE — permite criar tabelas ou bancos de dados
- DROP — permite deletar tabelas ou bancos de dados
- DELETE — permite excluir linhas de tabelas

- INSERT — permite inserir linhas em tabelas
- SELECT - permite usar o comando SELECT para ler os bancos de dados
- UPDATE — permite atualizar linhas de tabelas
- GRANT OPTION — permite conceder ou remover privilégios de outros usuários

Atividades

1. O que é necessário antes de criar uma tabela em um banco de dados?
2. Complete a syntax CREATE SCHEMA AUTHORIZATION
3. Quais são os direitos do usuário root no schema root?
4. Quais as permissões um usuário tem ao ser criado?
5. O ID de usuário está associado a que objeto de banco de dados?
6. Como funciona a especificação NOT NULL?
7. Como funciona a especificação UNIQUE?
8. Qual uma alternativa para a especificação UNIQUE?

TEMA 10: OUTROS RECURSOS

Funções agregadas

Muitas vezes, precisamos de informações que resultado de alguma operação aritmética ou de conjunto sobre os dados contidos nas tabelas de um banco de dados. Para isso, utilizamos as funções agregadas. A seguir apresentaremos algumas delas.

Função count( )

A função count, como o próprio nome sugere, conta a quantidade de linhas de uma tabela que satisfazem uma determinada condição.
Para contar o número de pedidos que existe na tabela pedidos, pode-se usar o count.

Exemplo

```
SELECT COUNT(PedidoNumr) FROM Pedidos
```

Função avg( )

A função avg é responsável por extrair a média aritmética dos valores de uma coluna.
Para calcular a média de pedidos da tabela pedidos, pode-se usar o avg.

Exemplo

```
SELECT AVG(PedidoNumr) FROM Pedidos
```

Função sum( )

A função sum é responsável por realizar a soma dos valores de uma coluna.
Para somar o total de pedidos abaixo de 1000 da tabela pedidos, pode-se usar o sum.

Exemplo

```
SELECT SUM(PedidoNumr) FROM Pedidos WHERE PedidoNumr<1000
```

COMANDO OU OPÇÃO	DESCRIÇÃO
Create schema authorization	Cria um esquema de banco de dados
Create table	Cria uma nova tabela no esquema de banco de dados do usuário
Not null	Assegura que uma coluna não contenha valores nulos
Unique	Assegura que uma coluna não contenha valores duplicados
Primary key	Define a chave primária de uma tabela
Foreign Key	Define a chave estrangeria de uma tabela
Default	Define o valor padrão de uma coluna(quando nenhum valor é fornecido)
Check	Valida os dados de um atributo
Create index	Cria um índice para uma tabela
Create view	Cria um subconjunto dinâmico de linhas/colunas a partir de uma ou mais tabelas
Alter table	Modifica uma definição de tabelas(adiciona, modifica ou exclui atributos ou restrições)
Create table as	Cria nova tabela com base em uma consulta no esquema de banco de dados do usuário
Drop table	Exclui uma tabela(e seus dados) de forma permanente
Drop index	Exclui um índice de forma permanente
Drop view	Exclui uma visuilização de forma permanente

CRIAÇÃO DE ÍNDICE

No banco de dados MySQL os índices podem ser criados com considerável facilidade, tanto no momento da concepção da tabela quanto em uma tabela já existente.

CREATE INDEX

```
CREATE UNIQUE INDEX index_name ON table_name (column1, column2, ...);
```

Neste exemplo é criada uma tabela chamada CLIENTES com dois campos: Codigo, do tipo inteiro e Nome do tipo texto.

```
CREATE TABLE CLIENTES(
Codigo  INT,
Nome    VARCHAR(50),
INDEX (Codigo)
);
```

O índice é criado com o uso da palavra reservada INDEX, seguida do nome da(s) coluna(s) a ser(em) indexada(s). Porém, nem sempre sabemos onde vamos precisar de um índice e muitas vezes é preciso criá-los quando a tabela já existe e inclusive quando já possui registros. Isso pode ser feito com uma instrução DDL (Data Definition Language), como veremos a seguir. Inicialmente criamos a tabela sem índice algum, em seguida adicionamos o índice à coluna “Codigo”.

```
CREATE TABLE CLIENTES(
```

```
    Codigo  INT,
    Nome    VARCHAR(50
);
```

Criando o índice separadamente

```
CREATE INDEX idx_CLIENTES_CODIGO ON CLIENTES(Codigo);
```

Nesse caso precisamos definir um nome para o índice (por questão de padronização, alguns profissionais optam por iniciar o nome do índice com um prefixo que indique que ele é um índice, como “id” ou “idx” de “index”, em inglês). Após o nome do índice adicionamos a palavra reservada “ON” que indica em que tabela e coluna o índice será criado, dados que vêm logo em seguida, como vemos na listagem.

Nem sempre o uso de índice trará um bom desempenho, pois a escolha incorreta de um índice pode causar um desempenho insatisfatório. Portanto, a tarefa do otimizador de consulta é selecionar um índice ou uma combinação de índices apenas quando isso gerar melhoria de desempenho e evitar a recuperação indexada quando isso atrapalhar o desempenho.

VIEW

Uma View é um objeto que pertence a um banco de dados, definida baseada em declarações SELECT´s, retornando uma determinada visualização de dados de uma ou mais tabelas.Esses objetos são chamados por vezes de “virtual tables“, formada a partir de outras tabelas que por sua vez são chamadas de “based tables” ou ainda outras Views.

CRIANDO UMA VIEW

Create view

```
CREATE VIEW view_name AS SELECT column1, column2, ...
FROM table_name
WHERE condition;
ou
CREATE VIEW nome_da_view AS SELECT * FROM nome_tabela;
```

Verificar se a View foi criada

```
SHOW TABLES;
```

Caso exista uma nova tabela chamada “nome_da_view” foi o nome que definimos para essa nova view; A criação da view foi executada com sucesso.

Alterando uma View

```
ALTER VIEW nome_da_view AS SELECT * FROM nome_outra_tabela;
```

Excluindo uma View

```
DROP VIEW nome_da_view;
```

E alguns casos, as Views são atualizáveis e podem ser alvos de declaração INSERT, UPDATE e DELETE, que na verdade modificam sua “based tables“.

Questões

1. O que são Funções agregadas?

2. Qual o papel da função count( )?

3. O que faz a função avg( )?
4. Qual o papel da função sum( )
5. O que é um index?
6. O que é uma view?
7. Como criar uma view?
8. E como é possível verificar se a View foi criada?
9. Qual o comando para remover uma visualização?
10. Como alterar uma view?

TEMA 11: COMANDOS DCL

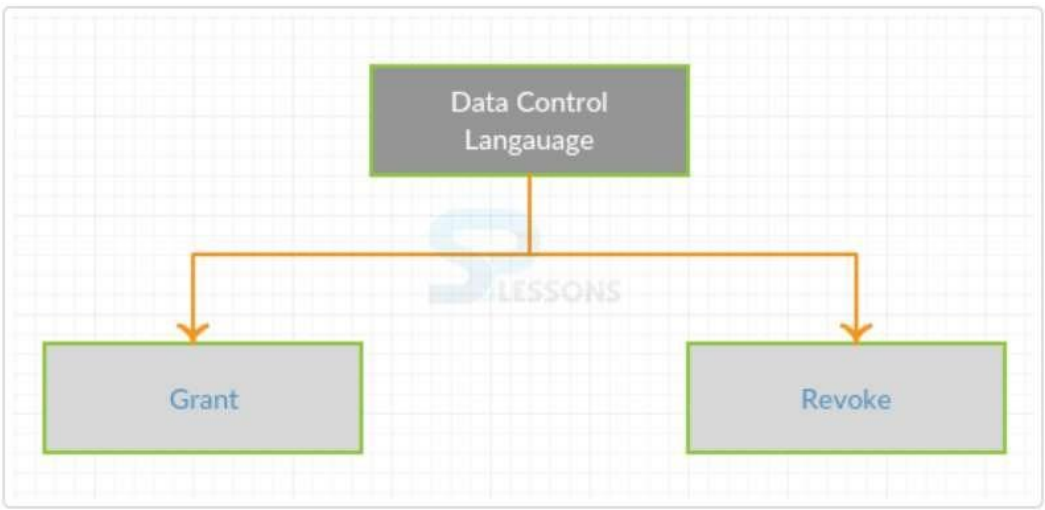
A Linguagem de consulta (SQL) estruturada como todos sabemos é a linguagem de banco de dados pelo uso de que podemos realizar determinadas operações no banco de dados existente e podemos usar essa linguagem para criar um banco de dados. O SQL usa certos comandos como Create, Drop, Insert etc. para realizar as tarefas necessárias. A Linguagem de Controle de Dados, ou do inglês Data Control Language(DCL), é uma linguagem de computador e um subconjunto de SQL, usada para controlar o acesso aos dados em um banco de dados.

Esses comandos SQL são principalmente categorizados e uma dessa categoria será discutida neste tema.

Banco de Dados consistente

Um banco de dados consistente é aquele em que são satisfeitas as restrições de integridade de banco de dados. Para assegurar a consistência do banco de dados, toda transação deve começar com o banco de dados em estado considerado consistente. Se ele não estiver nesse estado, a transação resultará em um banco de dados inconsistente e violará suas regras de integridade e de negócios. Por esse motivo todas as transações são controladas e executadas pelo SGBD para garantir a integridade do banco de dados. A maioria das transações reais é formada por duas ou mais solicitações. A solicitação de banco de dados é o equivalente a um único comando de SQL em um aplicativo ou transação. Para executar operações DML em uma tabela, as permissões são necessárias e são chamadas Privilégios. Esses privilégios podem ser controlados eficientemente usando as instruções do MySQL DCL.

O MySQL Data Control Language é semelhante ao SQL Data Control Language e estes são classificados em dois tipos:



Grant

Grant é usado para conceder permissões aos clientes. No banco de dados MySQL, ele oferece ao servidor e ao cliente uma grande quantidade de privilégios de controle. No lado do servidor do procedimento, ele incorpora a possibilidade de o servidor controlar determinados benefícios do cliente sobre o banco de dados MySQL e reduzir suas permissões de conexão do banco de dados ou conceder autorizações limitadas para uma tabela específica.

Concessões em objetos do banco de dados

Esta funcionalidade do comando GRANT concede privilégios específicos sobre um objeto do banco de dados para um ou mais papéis. Estes privilégios são adicionados aos já existentes, caso haja algum.

A palavra-chave PUBLIC indica que os privilégios devem ser concedidos para todos os papéis, inclusive aos que vierem a ser criados posteriormente. PUBLIC pode ser considerado como um grupo definido implicitamente que sempre inclui todos os papéis. Um determinado papel possui a soma dos privilégios concedidos diretamente para ele, mais os privilégios concedidos para todos os papéis que este seja membro, mais os privilégios concedidos para PUBLIC.

Se for especificado WITH GRANT OPTION quem receber o privilégio poderá, por sua vez, conceder o privilégio a terceiros. Sem a opção de concessão, quem recebe não pode conceder o privilégio. A opção de concessão não pode ser concedida para PUBLIC.

Não é necessário conceder privilégios para o dono do objeto (geralmente o usuário que o criou), porque o dono possui todos os privilégios por padrão (Entretanto, o dono pode decidir revogar alguns de seus próprios privilégios por motivo de segurança). O direito de remover um objeto, ou de alterar a sua definição de alguma forma, não é descrito por um privilégio que possa ser concedido; é inerente ao dono e não pode ser concedido ou revogado. O dono possui também, implicitamente, todas as opções de concessão para o objeto.

Dependendo do tipo do objeto, os privilégios padrão iniciais podem incluir a concessão de alguns privilégios para PUBLIC. O padrão é: não permitir o acesso público às tabelas, esquemas e espaços de tabelas; para os bancos de dados conceder o privilégio CONNECT e o privilégio de criação de tabela TEMP; para as funções conceder o privilégio EXECUTE; e para as linguagens conceder o privilégio USAGE. O dono do objeto poderá, é claro, revogar estes privilégios (para a máxima segurança o comando REVOKE deverá ser executado na mesma transação que criar o objeto; dessa forma não haverá espaço de tempo para outro usuário utilizar o objeto).

Os privilégios possíveis são:

SELECT
Permite consultar (SELECT) qualquer coluna da tabela, visão ou seqüência especificada. Também permite utilizar o comando COPY TO. Para as seqüências, este privilégio também permite o uso da função currval.

INSERT
Permite inserir (INSERT) novas linhas na tabela especificada. Também permite utilizar o comando COPY FROM.

UPDATE
Permite modificar (UPDATE) os dados de qualquer coluna da tabela especificada. Os comandos SELECT ... FOR UPDATE e SELECT ... FOR SHARE também requerem este privilégio (além do privilégio SELECT). Para as seqüências, este privilégio permite o uso das funções nextval e setval.

DELETE
Permite excluir (DELETE) linhas da tabela especificada.

REFERENCES
Para criar uma restrição de chave estrangeira é necessário possuir este privilégio, tanto na tabela que faz referência quanto na tabela que é referenciada.

TRIGGER
Permite criar gatilhos na tabela especificada (Consulte o comando CREATE TRIGGER).

CREATE
Para bancos de dados, permite a criação de novos esquemas no banco de dados.

Para esquemas, permite a criação de novos objetos no esquema. Para mudar o nome de um objeto existente é necessário ser o dono do objeto e possuir este privilégio no esquema que o contém. Para espaços de tabelas, permite a criação de tabelas e índices no espaço de tabelas, e permite a criação de bancos de dados possuindo este espaço de tabelas como seu espaço de tabelas padrão (Deve ser observado que revogar este privilégio não altera a colocação dos objetos existentes).

CONNECT
Permite ao usuário se conectar ao banco de dados especificado. Este privilégio é verificado no estabelecimento da conexão (além de serem verificadas as restrições impostas por pg_hba.conf).

TEMPORARY - TEMP
Permite a criação de tabelas temporárias ao usar o banco de dados.

EXECUTE
Permite utilizar a função especificada e qualquer operador implementado utilizando a função. Este é o único tipo de privilégio aplicável às funções (Esta sintaxe funciona para as funções de agregação também).

USAGE
Para as linguagens procedurais, permite o uso da linguagem especificada para criar funções nesta linguagem. Este é o único tipo de privilégio aplicável às linguagens procedurais.Para os esquemas, permite acessar os objetos contidos no esquema especificado (assumindo que os privilégios requeridos para os próprios objetos estejam atendidos). Essencialmente, concede a quem recebe o direito de "procurar" por objetos dentro do esquema. Sem esta permissão ainda é possível ver os nomes dos objetos, por exemplo consultando as tabelas do sistema. Além disso, após esta permissão ter sido revogada os servidores existentes poderão conter comandos que realizaram anteriormente esta procura, portanto esta não é uma forma inteiramente segura de impedir o acesso aos objetos.

ALL PRIVILEGES
Concede todos os privilégios disponíveis de uma só vez. A palavra-chave PRIVILEGES é requerida pelo SQL estrito. Os privilégios requeridos por outros comandos estão listados nas páginas de referência dos respectivos comandos.

Sintaxe

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

Exemplo
Visualizando o exemplo abaixo, o conceito de comando grant pode ser facilmente entendido.

```
mysql>grant select on sample.* to reader@localhost identified by 'secret';
Query OK,0 rows affected
mysql>exit;
Bye
D:\MySQL\bin>mysql -u reader -p
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| employee           |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
7 rows in set (0.00 sec)
mysql> select * from employee;
+--------+----------+-------+
| emp_id | emp_name |salary |
+--------+----------+-------+
|   1001 | mike     | 12000 |
|   1002 | maze     | 13000 |
|   1003 | jack     | 14000 |
+--------+----------+-------+
3 rows in set (0.04 sec)
```

Descrição: O comando revoke cancelará todas as permissões do usuário. Sintaxe:

REVOKE

O comando REVOKE revoga, de um ou mais papéis, privilégios concedidos anteriormente. A palavra-chave PUBLIC se refere ao grupo contendo todos os usuários, definido implicitamente. O significado dos tipos de privilégio deve ser visto na descrição do comando GRANT.

Deve ser observado que um determinado papel possui a soma dos privilégios concedidos diretamente para o próprio papel, mais os privilégios concedidos para os papéis dos quais o papel é membro no momento, mais os privilégios concedidos para PUBLIC.

Daí, por exemplo, revogar o privilégio SELECT de PUBLIC não significa, necessariamente, que todos os papéis perderão o privilégio SELECT para o objeto: os papéis que receberam o privilégio diretamente, ou através de outro papel, ainda terão o privilégio.

Se for especificado GRANT OPTION FOR somente a opção de concessão do privilégio é revogada, e não o próprio privilégio. Caso contrário, tanto o privilégio quanto a opção de concessão serão revogados.

Se o usuário possui um privilégio com opção de concessão, e concedeu este privilégio para outros usuários, então os privilégios que estes outros usuários possuem são chamados de privilégios dependentes. Se o privilégio ou a opção de concessão que o primeiro usuário possui for revogada, e existirem privilégios dependentes, estes privilégios dependentes também serão revogados se for especificado CASCADE, senão a ação de revogar falhará.

Esta revogação recursiva somente afeta os privilégios que foram concedidos através de uma cadeia de usuários começando pelo usuário objeto deste comando REVOKE. Portanto, os usuários afetados poderão manter o privilégio, se o privilégio também tiver sido concedido por outros usuários.

Ao revogar o privilégio de membro de um papel, GRANT OPTION passa a se chamar ADMIN OPTION, mas o comportamento é semelhante.

Deve ser observado, também, que esta forma do comando não inclui a palavra GROUP.

```
<priv_type> [<column_list>]
[priv_type [<column_list>]] … ON [object_type] priv_level FROM user
[user] …
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM user [user] …
```

CREATE USER

Usada para criar um usuário no sistema (sem privilégios)

Sintaxe:

```
CREATE USER usuário@host IDENTIFIED BY 'senha';
```

host é o nome do host a partir de onde o usuário pode se conectar ao banco de dados; geralmente usamos localhost para a máquina local. Se não for especificado um host, o MySQL acrescentará automaticamente o símbolo % como nome do host, o que significa que o usuário poderá se conectar de qualquer lugar. É possível também usar o endereço IP de um host (por exemplo, 127.0.0.1 para o host local).

Após a criação do usuário, ele não terá nenhum privilégio em nenhum banco de dados. Os privilégios podem ser atribuídos por meio da declaração **GRANT**, que estudaremos na próxima lição.

Exemplos:

1. Criando um usuário de nome “fabio” com senha “1234” no MySQL, com acesso a partir do host local:

```
CREATE USER fabio@localhost IDENTIFIED BY '1234';
```

Verificando se o usuário foi criado como especificado:

```
SELECT User, Host FROM mysql.user;
```

```
mysql> CREATE USER fabio@localhost IDENTIFIED BY '1234';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT User, Host FROM mysql.user;
+------------------+-----------+
| User             | Host      |
+------------------+-----------+
| root             | 127.0.0.1 |
| root             | ::1       |
| root             | debian    |
| debian-sys-maint | localhost |
| fabio            | localhost |
| root             | localhost |
+------------------+-----------+
6 rows in set (0.00 sec)
```

2. Criando um usuário de nome “ana” com acesso a partir de qualquer local:

```
CREATE USER ana IDENTIFIED BY "1234";
```

```
mysql> CREATE USER ana IDENTIFIED BY '1234';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT User, Host FROM mysql.user;
+------------------+-----------+
| User             | Host      |
+------------------+-----------+
| ana              | %         |
| root             | 127.0.0.1 |
| root             | ::1       |
| root             | debian    |
| debian-sys-maint | localhost |
| fabio            | localhost |
| root             | localhost |
+------------------+-----------+
7 rows in set (0.01 sec)
```

Note que na coluna Host aparece o símbolo **%** para a usuária ana, significando que ela pode acessar o SGBD de qualquer local.

3. Criando um usuário de nome marcos sem senha definida no momento:

```
CREATE USER marcos@localhost;
```

Para alterar ou configurar uma senha para esse usuário posteriormente, use o comando SET PASSWORD:

```
SET PASSWORD FOR 'marcos'@'localhost' = PASSWORD('1234');
```

Alteramos a senha do usuário marcos para 1234 com essa declaração.

Nas versões mais recentes do MySQL, a declaração SET PASSWORD foi deprecada, e não será mais utilizada nas próximas versões.

RENAME USER
Usada para renomear um usuário do MySQL. Se o usuário possuir privilégios configurados, eles são mantidos para o novo nome de usuário.

Sintaxe:

```
RENAME USER nome_atual TO novo_nome;
```

Exemplo:
1. Vamos renomear a usuária ana para monica:

```
RENAME USER ana TO monica;
```

DROP USER
Usada para excluir um usuário do MySQL. Esta declaração elimina o usuário e seus privilégios do sistema.

Sintaxe: DROP USER nome_usuário;

Exemplo:
Vamos remover a usuária monica: DROP USER monica;
Visualizando o exemplo abaixo, o conceito de comando revogar pode ser facilmente entendido.

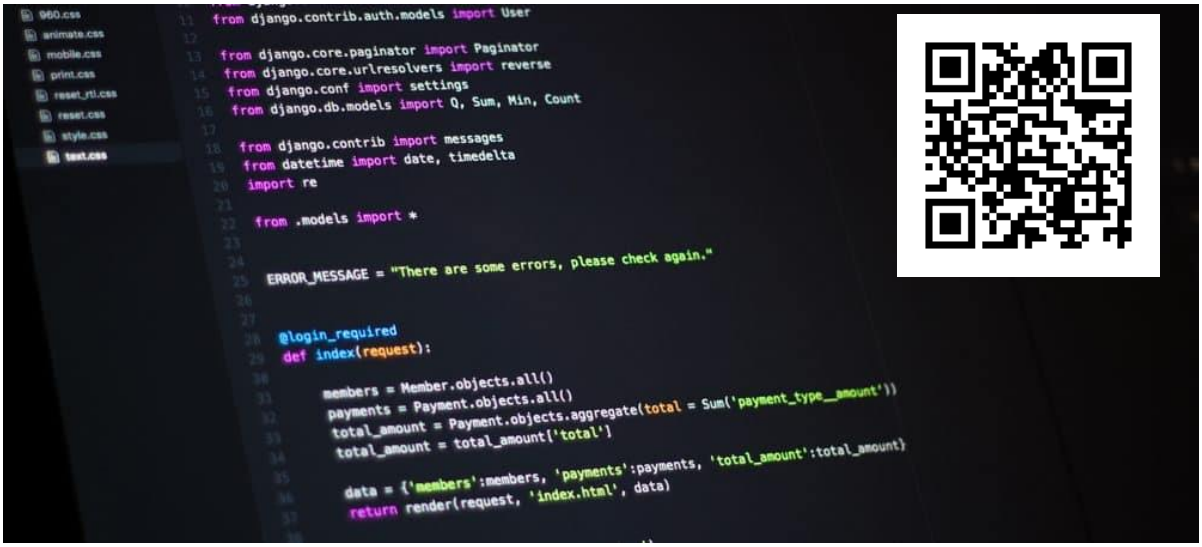
```
mysql> SHOW GRANTS FOR 'david'@'localhost';
+-----------------------------------------------------------------+
| Grants for david@localhost                                      |
+-----------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'david'@'localhost' WITH GRANT OPTION |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)mysql> REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'david'@
Query OK, 0 rows affected (0.00 sec)
mysql> show revoke for 'david'@'localhost';
ERROR 1064 (42000): there is no localhost user
```

Questões

1. O que é DCL?
2. Quais são os comandos DCL?
3. O que é o comando grant?

4. Explique a funcionalidade do comando revoke grant option.
5. Apresente a sintaxe e um exemplo do comando revoke.
6. Explique o que é a possibilidade de ALL PRIVILEGES.
7. Crie um comando DCL utilizando a funcionalidade all privileges.
8. Crie um comando Create user.
9. Crie um comando drop
10. Explique o comando rename

TEMA 12: CONECTIVIDADE EM PYTHON



Você necessita de um banco de dados para salvar informações em seus programas do Python, os principais que podem ser acessados por ele são: MySQL, SQLite, PostgreSQL e Interbase/Firebird. Falaremos sobre o MySQL. Primeiro temos que construir conexão com o MySQL. O seguinte irá se conectar ao banco de dados MySQL no programa Python

```
import pymysql

#database connection
connection =
pymysql.connect(host="localhost",user="root",passwd="",database="databaseName" )
cursor = connection.cursor()
# some other statements  with the help of cursor
connection.close()
```

Primeiro nós importamos o pyMySQL, então estabelecemos uma conexão. O pyMySQL.connect () leva quatro argumentos. O primeiro é o nome do host, ou seja, o localhost, e os três restantes são como são declarados. Usando esta conexão, criamos um cursor que será usado para diferentes consultas.

Para criar uma conexão com o banco de dados, use o nome de usuário e a senha do seu banco de dados MySQL:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
print(mydb)
```

Criando um banco de dados

Para criar um banco de dados no MySQL, use uma instrução "CREATE DATABASE":

Exemplo

Crie um banco de dados chamado "mydatabase":

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE mydatabase")
```

Verifique se existe banco de dados

Você pode verificar se existe um banco de dados listando todos os bancos de dados em seu sistema usando uma instrução "SHOW DATABASES":

Exemplo

Retorne uma lista dos bancos de dados do seu sistema:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()
mycursor.execute("SHOW DATABASES")

for x in mycursor:
  print(x)
```

Ou você pode tentar acessar o banco de dados ao fazer uma conexão:

Exemplo

Tente se conectar ao banco de dados "meu banco de dados":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
```

Python MySQL creating table

Vamos agora criar uma tabela chamada Artist com colunas - name, id e track

```
import pymysql

#database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="databaseName")
cursor = connection.cursor()
# Query for creating table
ArtistTableSql = """CREATE TABLE Artists(
ID INT(20) PRIMARY KEY AUTO_INCREMENT,
NAME  CHAR(20) NOT NULL,
TRACK CHAR(10))"""

cursor.execute(ArtistTableSql)
connection.close()
```

Python MySQL insert

Agora nosso interesse é inserir algumas entidades de linha na tabela. Primeiro você tem que escrever as consultas para inserir dados diferentes e, em seguida, executá-lo com a ajuda do cursor.

```
import pymysql

#database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="databaseName")
cursor = connection.cursor()

# queries for inserting values
insert1 = "INSERT INTO Artists(NAME, TRACK) VALUES('Towang', 'Jazz' );"
insert2 = "INSERT INTO Artists(NAME, TRACK) VALUES('Sadduz', 'Rock' );"

#executing the quires
cursor.execute(insert1)
cursor.execute(insert2)


#commiting the connection then closing it.
connection.commit()
connection.close()
```

Python MySQL select

Nós inserimos duas linhas no código acima. Agora queremos recuperá-los. Para fazer isso, dê uma olhada no seguinte exemplo:

```
import pymysql

#database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="databaseName")
cursor = connection.cursor()

# queries for retrievint all rows
retrive = "Select * from Artists;"

#executing the quires
cursor.execute(retrive)
rows = cursor.fetchall()
for row in rows:
   print(row)


#commiting the connection then closing it.
connection.commit()
connection.close()
```

```
D:\Software\Python\Python36-32\python.exe D:/T_Code/Pythongenerator/package2/mySql.py
(1, 'Towang', 'Jazz')
(2, 'Sadduz', 'Rock')

Process finished with exit code 0
```

Selecione com um filtro

Ao selecionar registros de uma tabela, você pode filtrar a seleção usando a instrução "WHERE":

Exemplo

Selecione o (s) registro (s) em que o endereço seja "Park Lane 38": resultado:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address ='Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
```

```
  print(x)
```

Caracteres curinga
Você também pode selecionar os registros que constam, incluindo ou terminam com uma determinada letra ou frase.

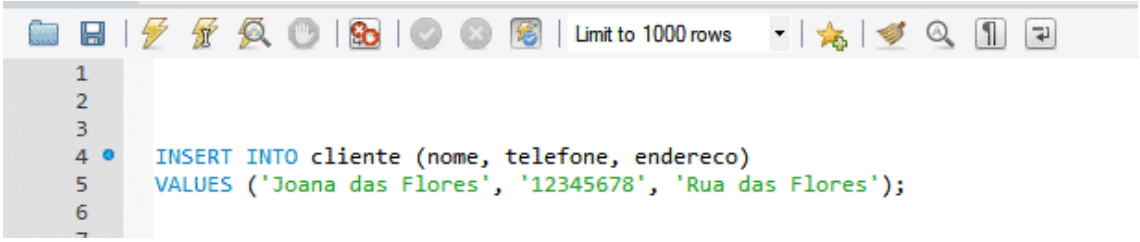
Use o % para representar caracteres curinga:

Exemplo
Selecione os registros em que o endereço contenha a palavra "caminho":

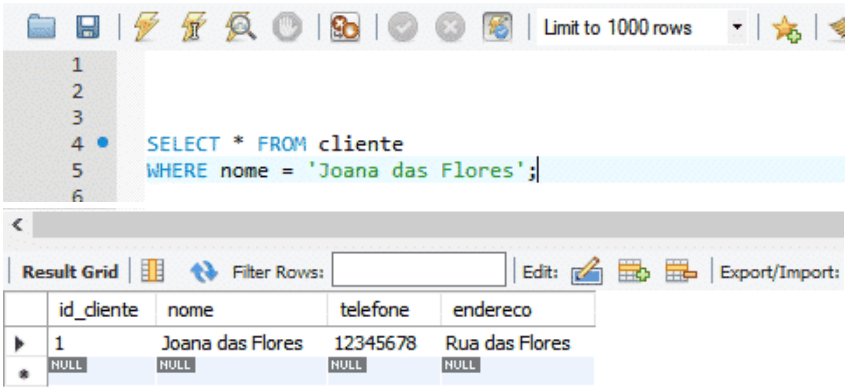
```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address LIKE '%way%'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

CRUD - (Create, Read, Update e Delete)
Pode ser traduzido como: criar, ler, atualizar e excluir, engloba os principais comandos da linguagem SQL (Structured Query Language) para a manipulação de dados, que são: INSERT (inserir), READ (ler), UPDATE (alterar) e DELETE (remover). As interfaces CRUD permitem cadastrar (create), visualizar (read), editar (update) e excluir (delete) registros de um sistema.

INSERT – CREATE
Esse comando é responsável por inserir dados na tabela. Assim, toda vez que você quiser adicionar algo novo, você precisa usar o comando INSERT, seguido do campo e do valor que você quer adicionar.



SELECT – READ
O SELECT é de longe o comando mais utilizado do CRUD. Justamente porque é ele quem traz todo o resultado da busca que você quer saber. Assim, ele traz para você exatamente o que você quer saber quando pesquisa algo no banco de dados.



UPDATE

O comando UPDATE tem por finalidade a atualização dos dados que você inseriu na tabela. Se por algum motivo você quer modificar um campo, atualizar o valor ou acrescentar novos dados, é o UPDATE que você vai utilizar.

Python MySQL Update

```
updateSql = "UPDATE  Artists SET NAME= 'Tauwang'  WHERE ID = '1' ;"
cursor.execute(updateSql  )
```

Exemplo

```
UPDATE cliente SET endereco = 'Rua da Flor' WHERE nome = 'Joana das Flores';
```

Para atualizar um campo, basta especificar qual campo você vai modificar, com o valor antigo pelo valor que você quer substituir.

DELETE

Por fim temos o comando DELETE que simplesmente exclui o dado especificado. Muito cuidado, pois, uma vez excluído não tem Ctrl + z para voltar no tempo e desfazer o erro.

Python MySQL Delete

```
deleteSql = "DELETE FROM Artists WHERE ID = '1'; "
cursor.execute(deleteSql )
```

Exemplo

```
DELETE from cliente WHERE nome = 'Joana das Flores';
```

Python MySQL – Drop Table

```
dropSql = "DROP TABLE IF EXISTS Artists;"
cursor.execute(dropSql)
```

Impedir injeção de SQL

Quando os valores da consulta são fornecidos pelo usuário, você deve escapar dos valores.

Isso evita injeções de SQL, que é uma técnica comum de hacking para destruir ou usar indevidamente seu banco de dados.

O módulo mysql.connector tem métodos para escapar dos valores da consulta:

Exemplo
Escape dos valores da consulta usando o %s método de placholder :

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Limite o resultado
Você pode limitar o número de registros retornados da consulta, usando uma instrução "LIMIT":

Exemplo
Selecione os 5 primeiros registros na tabela "clientes":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5")

myresult = mycursor.fetchall()
```

```
for x in myresult:
  print(x)
```

Comece de outra posição
Se você deseja retornar cinco registros, a partir do terceiro registro, você pode usar uma palavra-chave "OFFSET":

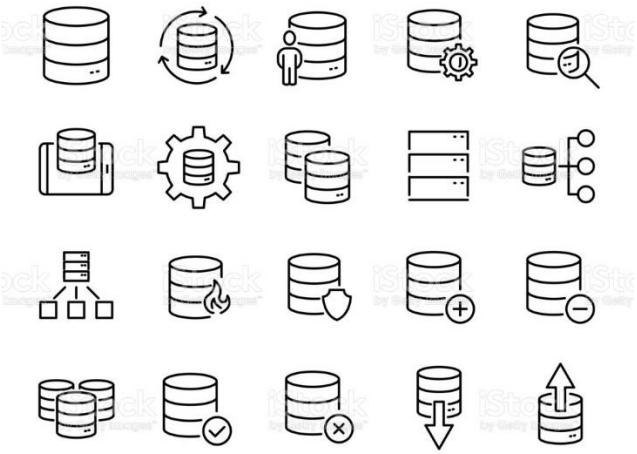
Exemplo
Comece na posição 3 e retorne 5 registros:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Questões

1. O que é o Python?
2. Por que é uma ferramenta interessante de se usar com MySQL?
3. Crie um comando para inserir uma linha usando Python e MySQL.
4. Crie um comando para selecionar os dados inseridos usando Python e MySQL.
5. Crie um comando para atualização de dados usando Python e MySQL.
6. Crie um comando para deletar um objeto de banco de dados usando Python e MySQL.
7. O que é o CRUD em Python e MySQL.
8. Apresente um exemplo de CRUD.
9. Explique como criar um banco de dados MySQL e Python.
10. Como limitar buscas?

TEMA 13 : DATA WAREHOUSE

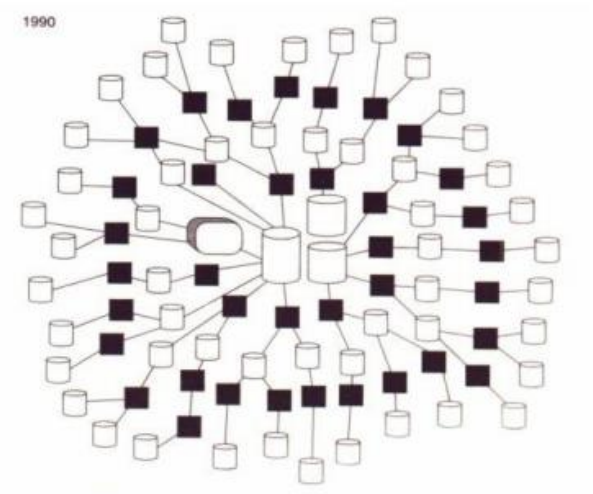


O surgimento do conceito Data Warehouse surgiu em decorrência às dificuldades que muitas organizações começaram a ter com uma grande quantidade de dados que suas aplicações estavam gerando. O volume de dados e a dificuldade de reuni-los para a análise eficiente, foram os responsáveis pela iniciativa de reunir, em um único local, os dados considerados relevantes ao processo decisório.

Evolução da tecnologia

Com o avanço da internet e crescimento das transações online com alta performance, começaram a ser necessários programas de extração de dados. Esses programas têm como princípio percorrer os arquivos do banco de dados usando condições e critérios e, ao encontrar os dados específicos, direciona-os para um arquivo de extração do banco de dados.

O processo de extração se popularizou rapidamente, em decorrência do aumento de informações e necessidade de análise de dados por todos os segmentos de organizações, fossem privadas ou governamentais, formando a chamada “arquitetura de desenvolvimento espontâneo” ou “teia de aranha”. O processo destas operações se baseava nas extrações, seguidas pelas extrações das extrações, assim como as extrações das extrações das extrações, tudo em um processo de busca por filtragem dos dados.



Esta estrutura era utilizada com uma arquitetura de desenvolvimento espontâneo, a qual não garantia credibilidade dos dados, a produtividade era limitada e a dificuldade de transformar dados puros em informações muito grande.

O AMBIENTE PROJETADO

A arquitetura de desenvolvimento espontâneo não era suficiente para atender as necessidades do futuro das empresas, fazendo-se necessário uma mudança de arquitetura, surgindo o ambiente

projetado de Data Warehouse. No cerne do ambiente projetado está a percepção de que há fundamentalmente duas espécies de dados – dados primitivos e dados derivados. A Tabela 1 mostra algumas das principais diferenças entre dados primitivos e derivados.

Data warehouse é uma coleção de dados orientada por assuntos, integrada, variante no tempo e não volátil, que tem por objetivo dar suporte aos processos de tomada de decisão. O objetivo é fornecer uma “imagem única da realidade do negócio”. De uma forma geral, sistemas de Data warehouse compreendem um conjunto de programas que extraem dados do ambiente de dados operacionais da empresa, um banco de dados que os mantém, e sistemas que fornecem estes dados aos seus usuários. É, ainda, o epicentro da infraestrutura estratégica da empresa. Ele suporta processamento informacional promovendo uma sólida plataforma de dados históricos integrados para serem analisados com visão corporativa.

O Data warehouse é um banco de dados contendo dados extraídos do ambiente de produção da empresa, que foram selecionados e depurados tendo sido otimizados para processamento de consulta e não para processamento de transações. Em geral, um Data warehouse requer a consolidação de outros recursos de dados além dos armazenados em banco de dados relacionais, incluindo informações provenientes de planilhas eletrônicas, documentos textuais, etc.

Sistemas de Data warehouse revitalizam sistemas da empresa, pois:

- Permitem que sistemas mais antigos continuem em operação;
- Consolidam dados inconsistentes dos sistemas mais antigos em conjuntos coerentes;
- Extraem benefícios de novas informações oriundas das operações correntes;
- Provêm ambiente para planejamento de novos sistemas de cunho operacional.

É importante considerar, no entanto, que um Data warehouse não contém apenas dados resumidos, podendo conter também dados primitivos. É desejável prover ao usuário a capacidade de aprofundar-se num determinado tópico, investigando níveis de agregação menores ou mesmo o data primitivo, permitindo também a geração de novas agregações ou correlações com outras variáveis.

Extração de dados

O objetivo do Data Warehouse é centralizar os dados retirados de diversas fontes e facilitar a consulta.

Os dados podem ser extraídos de:

- Planilhas;
- ERPs;
- CRMs

Com diversos formatos, entre os quais:

- Bancos de Dados (SQL)
- XLS
- TXT
- CSV

Após a extração, os dados normalmente são acomodados na área destinada aos processos de qualidade e padronização dos dados. Posteriomente podem ser direcionados ao Enterprise Data Warehouse (EDW) ou aos Data Marts diretamente. Viabiliza a busca por todas as informações relevantes em um único espaço – organizado, atualizado, criado com foco em facilitar a consulta.

TIPOS DE DATA WAREHOUSE

Integrado: Data warehouses integrados têm como principal função gerar relações consistentes entre dados de fontes variadas. Eles são capazes de padronizar informações que vêm de sistemas diferentes, permitindo que, posteriormente, elas sejam tratadas dentro dele.

Variável ao longo do tempo: Já os que se caracterizam por ser variáveis ao longo do tempo usam recursos de data mining, que tomam como referência principal um ou mais períodos.
Dessa forma, a mineração de dados não se aplica em tempo real, como acontece em bancos OLTP.

Por assunto: Por sua vez, os armazenamentos de dados organizados por assunto são aqueles que atendem aos objetivos de negócios em contextos específicos.
Como exemplo, um escritório contábil que precisa listar e cadastrar diferentes clientes e contribuintes, assim como os impostos que eles tenham que apurar e recolher.

Não volátil: Dados em data warehouses são sempre tratados para posterior processamento.
Isso significa que, antes de eles serem utilizados pelo usuário final, devem passar por processos de exclusão e consultas, nos quais são modificados. Desse modo, eles passam a ser estáticos, ou não voláteis.

Questões

1. Que é um data warehouse?
2. Para que serve?
3. Como é utilizado?
4. Porque utilizar um DW nas empresas?
5. Quais as principais características de um DW?
6. Como os dados são armazenados em um DW?
7. Explique o conceito de modelo dimensional.
8. que são ferramentas OLAP? Para que serve?
9. Qual a diferença entre OLAP e OLTP?
10. Quais os métodos de armazenamento das ferramentas OLAP?

TEMA 14: BUSINESS INTELLIGENCE

No competitivo mundo dos negócios, a sobrevivência de uma empresa depende de como rápido eles são capazes de reconhecer dinâmicas de negócios em mudança e desafios, e responder corretamente e rapidamente. As empresas também devem antecipar tendências, identificar novas oportunidades, transformar sua estratégia e reorientar os recursos para ficar à frente da competição. A chave para o sucesso é a informação.

As empresas coletam volumes significativos de dados e têm acesso a ainda mais dados de fora de seus negócios. Eles precisam da capacidade de transformar esses dados brutos informações acionáveis, capturando, consolidando, organizando, armazenando, distribuir, analisar e fornecer acesso rápido e fácil a ele. Isto é vantagem competitiva, mas também o desafio. Tudo isso é o objetivo da inteligência de negócios (BI).

Com a utilização dos dados, das informações agrupadas e do conhecimento, é possível compreender o conceito conhecido atualmente por inteligência de negócios (BI - business intelligence) que, segundo a definição, serve para aumentar a competitividade e alavancar os resultados empresariais. Com essa poderosa solução, as empresas obterão velocidade de respostas tanto no nível estratégico como no tático e operacional, conquistando com isso enormes ganhos.

O BI é o uso de diversas e variadas fontes de informação que as organizações utilizam para definir estratégias e subsidiar processos organizacionais.

Atualmente, o modelo de negócios das empresas vem se transformando e, consequentemente, aumentando a busca por ferramentas de BI para atender a demanda de gerenciamento das informações, pois elas impactam diretamente no processo de tomada de decisão.

Os conceitos de Big Data e BI são convergentes, levando alguns pesquisadores e técnicos a considerar as duas concepções como uma apenas.

Big Data é um conjunto de tecnologias voltadas para a gestão de grandes volumes de dados em diferentes formatos, tem como características possibilitar a gestão dos dados em grande velocidade de processamento, mas não se refere especificamente à sua organização, nem à rastreabilidade dos dados processados.

O conceito vem evoluindo nos últimos dez anos e a sua origem está no BI, que é o resultado do processo de mineração de dados analíticos voltados para a combinação de dados, aporte de comentários, formatação de metadados e de conteúdo a serem usados no processo decisório.

O BI é um conceito que surgiu com o propósito de realizar a integração e extração de dados de variadas fontes homogêneas e heterogêneas, e posteriormente efetuar a transformação desses dados

obtidos em informações de valor para conhecimento organizacional, valorização da experiência, e a contínua pesquisa por relações de causa e efeito de riscos em projetos, com base em hipóteses que darão suporte para o desenvolvimento de ações rápidas e estratégias competitivas.

Para o autor, o BI representa as técnicas e habilidades que permitem gerar conhecimento para empresas e gestores, a partir de dados e informações que até então não tinham muita relevância, assim, pode-se afirmar que dados e informações no BI são tidas como matérias primas para o desenvolvimento de novas percepções, soluções e padrões que auxiliam a melhores tomadas de decisões nos negócios e projetos.

Na atualidade, qualquer empresa necessita de uma integração de seus dados e informações procedente de sua cadeia de valor, e toda organização que utiliza o BI para resolver seus problemas operacionais, pode beneficiar-se com a autonomia da gestão de negócios ou projetos ao planejar e gerenciar, tanto dos riscos e ações estratégicas, como o processo de controle de mudanças.

Vantagens do BI

O BI proporciona alguns benefícios, como a expansão do conhecimento do negócio, assim como uma melhor visão da variação do mercado com a perspectiva de novas oportunidades.

Uma das principais vantagens do Business Intelligence é que ele permite uma visão nova para a organização buscar maior eficácia e maior competitividade no mercado com suas ferramentas de análise e cruzamento de informações.

O BI tem como vantagem sua tecnologia, que é capaz de superar a concorrência e, ao mesmo tempo, atender às necessidades de informação dos gestores da organização.

O BI agiliza processos, diminui o desperdício de tempo e transforma dados em informações relevantes. A implantação de uma ferramenta de BI consegue agilizar processos do negócio através de automatização, extraindo dados, fazendo cruzamentos e análises, montando e enviando relatórios com gráficos de forma automática e inteligente. O resultado das etapas que formam o BI pode estar disponível de forma online, e ser acessado na nuvem.

Existem inúmeras outras vantagens na adoção de BI, como redução de retrabalho, informações confiáveis, documentos padronizados e recebimento das informações em tempo real, entre outros.

Um dos principais benefícios do BI é a sua capacidade de fornecer informações confiáveis e precisas em tempo real para as decisões, planejamento estratégico e para a sobrevivência organizacional.

Desvantagens do BI

Nem tudo são flores na área tecnológica. Muitas ferramentas acabam tendo impactos inesperados e negativos na organização.

A principal desvantagem da aplicação do BI em pequenas e médias empresas é o alto custo da ferramenta é a mais apontada. Muitas vezes estas empresas não possuem recursos financeiros necessários para viabilizar o projeto.

Além disso, pode ser que seja necessária a adaptação de toda sua estrutura operacional para a implantação do BI, gerando despesas de hardware e software, assim como despesas com mão-de-obra especializada como contratação de programadores, analistas de BI, adaptação na estrutura de segurança. O suporte tem custo elevado e a manutenção e atualização necessárias dos sistemas.

Muitos projetos não alcançam o sucesso por falta de foco da empresa e dos envolvidos no processo, faltando maturidade da gestão para trabalhar com indicadores e múltiplas informações, que podem acabar dificultando a análise estratégica. Em alguns casos é apontada uma curva de aprendizado lenta e prejudicial para o sucesso do BI na organização.

Assim, o BI é importante ao suprir a necessidade de qualidade de informações que auxiliem o gestor no momento de decidir sobre algo. Nesse contexto, as tecnologias da informação, mais especificamente o BI assumem papel preponderante no tocante ao suporte aos tomadores de decisões, que sempre estão em busca de diferenciais competitivos para sua empresa e projetos.

Modelagem de dados

De um modo geral, um modelo é uma abstração e reflexão do mundo real. A modelagem nos dá a capacidade de visualizar o que ainda não conseguimos perceber. É o mesmo com modelagem de dados. O objetivo principal de um modelo de dados é garantir que todos os objetos de dados exigidos pela empresa são representados de maneira precisa e completa.

Da perspectiva de negócios, um modelo de dados pode ser facilmente verificado porque o modelo é construído usando notações e linguagem que são fáceis de entender e decifrar.

No entanto, do ponto de vista técnico, o modelo de dados também é detalhado o suficiente para serve como um modelo para o DBA ao criar o banco de dados físico. Por exemplo, o modelo pode ser facilmente usado para definir os elementos-chave, como as chaves primárias, chaves estrangeiras e tabelas que serão usadas no design dos dados estrutura.

Modelos de dados são sobre captura e apresentação de informações. Toda organização tem informações que normalmente estão no formato operacional (como OLTP aplicativos) ou o formulário informativo (como o data warehouse).

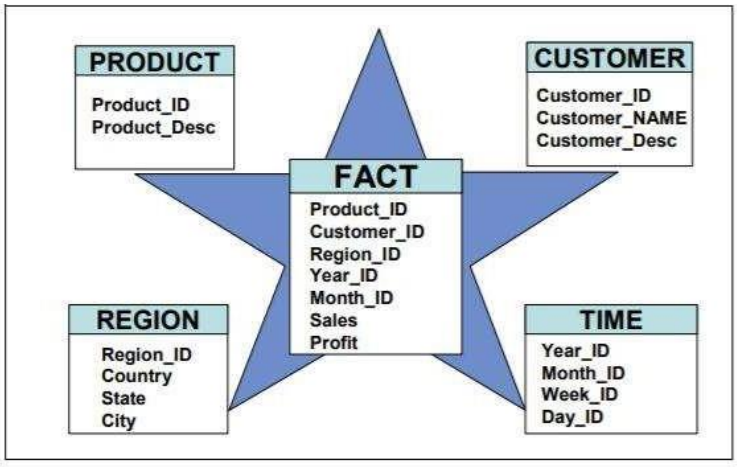
Tradicionalmente, os modeladores de dados utilizaram o diagrama E / R, desenvolvido parte do processo de modelagem de dados, como uma mídia de comunicação com o negócio analistas. O foco do modelo de E / R é capturar as relações entre várias entidades da organização ou processo para o qual projetamos o modelo.

O diagrama E / R é uma ferramenta que pode ajudar na análise de requisitos de negócios e no design da estrutura de dados resultante.

No entanto, o foco do modelo dimensional está nos negócios. Modelagem dimensional nos dá uma capacidade melhorada para visualizar as questões muito abstratas que os analistas de negócios são obrigados a responder. Utilizando modelagem dimensional os analistas podem facilmente entender e navegar na estrutura de dados e explorar completamente os dados. Na verdade, os dados são simplesmente um registro de todas as atividades de negócios, recursos e resultados da organização. O modelo de dados é um modelo bem organizado abstração desses dados. Então, é bastante natural que o modelo de dados tenha se tornado melhor método para entender e gerenciar os negócios da organização. Sem um modelo de dados, seria muito difícil organizar a estrutura o conteúdo dos dados no data warehouse.

Modelagem dimensional

Um modelo dimensional também é comumente chamado de esquema em estrela. Este tipo de modelo é muito popular no data warehousing porque pode fornecer consultas muito melhores desempenho, especialmente em consultas muito grandes, do que um modelo E / R. no entanto também tem o maior benefício de ser mais fácil de entender. Consiste, tipicamente, de uma grande tabela de fatos (conhecida como tabela de fatos), com várias outras tabelas em torno dele que contém dados descritivos, chamados dimensões. Quando é desenhado, assemelha-se à forma de uma estrela, portanto, o nome.



Características da tabela fato

A tabela de fatos contém valores numéricos do que você pode medir. Por exemplo, um valor de fato de 20 pode significar que 20 itens foram vendidos. Cada tabela de fatos contém as chaves para as tabelas de dimensões associadas. Essas são chamadas chaves estrangeiras na tabela de fatos. Tabelas de fatos geralmente contêm um pequeno número de colunas. Comparado às tabelas de dimensão, as tabelas de fatos possuem um grande número de linhas. As informações em uma tabela de fatos possuem características, como:

- É numérica e usada para gerar agregados e resumos. Os valores de dados precisam ser aditivos ou semi-aditivos para permitir resumo de muitos valores.
- Todos os fatos no Segmento 2 devem referir-se diretamente às chaves de dimensão Segmento 1 da estrutura, permite o acesso a informações adicionais das tabelas de dimensão.

Atividades

1. O que é BI?
2. Em quais situações o uso do BI nas empresas é indicado?
3. Qual o ganho que as ferramentas de BI podem trazer para as empresas?
4. Quais as principais características de um BI.
5. Qual é a diferença entre tabela Fato e tabela Dimensão?
6. Como as tabelas fato e dimensão se relacionam?
7. Explique o que é modelo estrela.
8. Explique o que é modelo snow flake ou floco de neve.
9. Qual a importância do modelo estrela para o uso do BI?
10. Quais as vantagens do BI?

REFERÊNCIAS BIBLIOGRÁFICAS

ALBERTIN, Alberto Luiz. Et al. Tecnologia da Informação. São Paulo. Atlas, 2004.

DEITEL, H. M. Java como programar. Porto Alegre, Bookman, 2001.

FERRAZ, Inhaúma Neves. Programação com arquivos. Barueri-SP. Manole, 2003.

PROGRAMAÇÃO DE COMPUTADORES, ESSA – Escola Técnica Profissionalizante, Material Didático, São Paulo - SP, 2018.

HOLZNER, Steven. Visual Basic 6 - Programação Básica, 1999.