

EAD
UNISANTA

BANCO DE DADOS

Me. Helio Augusto de Lima Rangel

**GUIA DA
DISCIPLINA**

1. POR QUE BANCO DE DADOS?

Objetivo:

Entender a evolução e aplicação dos Bancos de Dados e conhecer as principais tecnologias na área.

Introdução:

A principal motivação que fez nascer e evoluir os Banco de Dados foi a necessidade de preservar os dados utilizados nas aplicações dos computadores. Como sabemos, a memória dos computadores é volátil, ou seja, se perde quando a memória para de ser energizada. Hoje até temos tecnologia de memória que não precisa de energia para se manter. Mas esta tecnologia é relativamente recente e a evolução dos bancos de dados deixou muito para trás a ideia de apenas a persistência dos dados.

A evolução do Banco de Dados até o que é nos dias de hoje foi gradual e passou por várias etapas.

Evolução

1.1. Como eram armazenados os dados antes dos computadores?

Antes dos computadores as pessoas organizavam seus dados em ficha, livros de lançamento etc. Quem já foi em um cartório e teve oportunidade de ver as muitas estantes com enormes livros de registros, certamente ficou impressionado como era possível as pessoas se acharem no meio de tanta informação. Hoje a maioria dos cartórios digitalizou seus livros.

1.2. Como eram armazenados os dados no início da era digital?

Inicialmente não existiam os meios magnéticos para armazenar dados e para isso se utilizava fitas de papel perfurado, ou cartões, que eram “lidos” por equipamentos com pinos que se encaixavam nos furos. A fita de papel evolui para fitas magnéticas que podiam ser gravadas e regravadas, ao contrário das fitas de papel e cartões perfurados. Mesmo assim as fitas magnéticas ainda tinham a grande desvantagem de só poderem ser acessadas sequencialmente. Mais ou menos nessa época apareceram os primeiros discos

rígidos que iniciaram uma grande revolução no armazenamento de dados. Nesta época se armazenava basicamente arquivos fontes de programas e tabelas de texto. Os discos rígidos eram muito caros e com tecnologia muito sofisticada.

1.3. Algumas soluções para persistir dados antes dos atuais bancos de dados.

1.3.1. Texto espaçado:

Os dados eram gravados em linhas com algum tipo de formatação, como por exemplo número fixo de colunas para cada dado. Então se considerava que as primeiras 50 colunas continham o nome, as 20 seguintes o RG e assim por diante.

1.3.2. Texto com separadores:

As colunas são separadas por caracteres delimitadores como ponto e vírgula, por exemplo, ou os dados podem ser colocados entre aspas e separados por vírgula. Usamos até hoje várias formas de utilizar este tipo de arquivo. O Excel, por exemplo, possui um formato texto, o CSV, que utiliza exatamente este conceito.

1.3.3. Arquivos ISAN:

Os arquivos largamente utilizados nas aplicações COBOL, que por sinal ainda estão vivos e ativos espalhados por aí. São basicamente arquivos texto formatados como *Texto Espaçado*, mas que possui uma biblioteca de acesso aos dados que permite ao programador salvar, localizar, alterar e inserir dados de forma segura e intuitiva. O ISAN não trabalha com a Linguagem SQL, mas tem sua própria. Hoje em dia muitos sistemas COBOL migraram seus dados para bancos de dados relacionais modernos.

1.3.4. O Modelo Hierárquico:

É uma estrutura de armazenamento de dados bastante sofisticada que permite armazenar e recuperar informações de forma rápida, segura e relativamente simples. O Modelo hierárquico também conhecido como árvores binárias, é bastante utilizado pelos bancos de dados para auxiliar nos seus processos internos.

1.3.5. O Modelo em Redes:

Também é baseado em uma tecnologia estudada em Estrutura de dados que a lista duplamente encadeada. Também permite localização, inserção e deleção rápidas de dados. Como o Modelo Hierárquico, ele também é bastante utilizado pelos bancos de dados para auxiliar nos seus processos internos.

1.4. Os modelos de Banco de Dados mais modernos:

1.4.1. O Modelo Relacional:

Será o foco de nosso curso, largamente utilizado no mundo. Seu principal poder reside em poder garantir integridade relacional, ou em resumo, garantir que os dados não se desconectem um dos outros, ou que fiquem órfãos por exemplo. Veremos tudo isso com detalhes mais para frente.

1.4.2. O Modelo Orientado a Objetos:

Muito importante e bastante poderoso, se baseia no conceito de armazenar objetos serializados de forma que a ideia de várias tabelas para dados de uma mesma entidade, distribuída em várias tabelas é substituída por persistir o objeto completo. Isso pode parecer que teremos muitas duplicidades de dados, mas a estrutura hierárquica e o relacionamento entre as classes, consegue garantir a integridade dos dados.

1.4.3. O Modelos No Sql (Não relacionais):

Uma tendência nos bancos de dados de última geração. São bancos que acessam de forma extremamente rápida uma grande quantidade de informação, possibilitando com facilidade relacionamentos e buscas que nos Bancos de Dados Relacionais seriam impensáveis. Podemos compreender por que eles conseguem fazer isso: Exatamente por não serem relacionais, não fazem tantas verificações de integridade, não consultam tantos índices e chaves de relacionamento. As validações feitas pelos bancos de dados relacionais são cada vez mais demoradas e complexas com o aumento da concorrência de diversos usuários simultâneos.

1.4.4. *Big Data*:

É um banco de dados “fora da caixinha”, é não Relacional e especializado em volumes gigantescos de dados, trabalhando com vários tipos de dados como imagens, vídeos, textos gigantescos (livros completos, bibliotecas etc.), e-mails, dados gerados por “Internet das Coisas” e por aí vai. Tudo isso em velocidades extremamente altas conseguindo garantir consistência e confiabilidade e consequente agregar valor ao projeto.

2. REQUISITOS DE UM BANCO DE DADOS

Objetivo:

Entender e aprender a avaliar um Banco de Dados tendo em vista a sua aplicação.

Introdução:

Existem diversas opções no mercado de Banco de Dados Relacionais, o profissional de Banco de Dados precisa levar em conta diversos fatores para que possa decidir pela melhor solução. Vamos mostrar quais são os critérios que podem nos ajudar nesta hora.

O que é um Banco de Dados Relacional (SGBDR)

“É uma coleção de itens de dados com relacionamentos predefinidos entre si. Esses itens são organizados como um conjunto de tabelas com colunas e linhas. As tabelas são usadas para reter informações sobre os objetos a serem representados no banco de dados. Cada coluna da tabela retém um determinado tipo de dado e um campo armazena o valor em si de um atributo. As linhas na tabela representam uma coleção de valores relacionados de um objeto ou de uma entidade. Cada linha em uma tabela pode ser marcada com um único identificador chamado de chave principal. Já as linhas entre as várias tabelas podem ser associadas usando chaves estrangeiras. Esses dados podem ser acessados de várias maneiras diferentes sem reorganizar as próprias tabelas do banco de dados.”

<https://aws.amazon.com/pt/relational-database/>

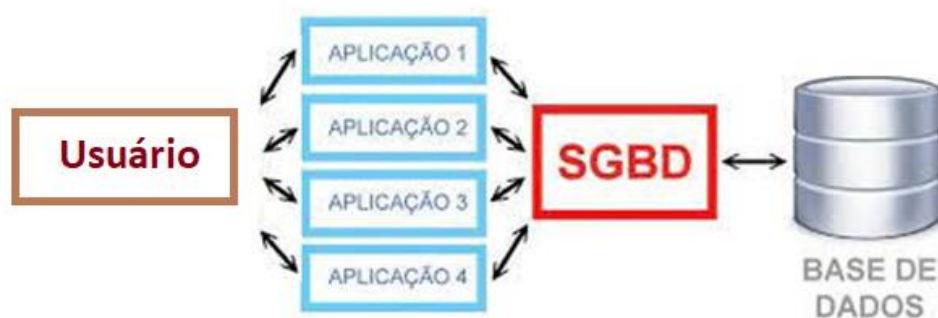


Figura1: Esquema ilustrativo de como funciona um SGBD. Note que se trata de um software que permite a interação entre uma ou mais aplicações com a Base de Dados. Observe que várias aplicações podem acessar simultaneamente a mesma base de dados

2.1. O que devemos esperar de um bom SGBDR

2.1.1. Mecanismos eficazes de segurança:

O SGBDR deve fornecer mecanismos de proteção a acesso aos dados ao banco como um todo, a esquemas específicos e alguns casos tabelas e campos. Alguns bancos permitem segurança a nível de usuário, ou seja, é possível saber exatamente que usuário se autenticou no banco e que operações ele realizou. A forma mais comum de autentificação é da aplicação. Desta forma o log de acesso e alterações, se necessário, precisa ser feito pela aplicação.

2.1.2. Bom desempenho:

O desempenho de Bancos Relacionais é bastante dependente do nível de restrições e relacionamentos que precisam realizar. Um banco mais poderoso pode muito bem ser mais lento que um outro considerado inferior. O motivo pode ser que um banco realiza mais operações de segurança de dados e recuperação em caso de falhas que outro. Mesmo assim procuramos avaliar todos estes fatores e sempre optar pelo banco que consegue fazer coisas equivalentes no menor tempo.

2.1.3. Bom custo-benefício:

O Custo de um banco de dados é mais difícil de calcular do que parece. A perda de dados importantes, uma dificuldade de encontrar suporte técnico em caso de necessidade, precisam ser avaliadas no cálculo do custo de um banco. Normalmente o custo inicial da compra da licença representa um pequeno percentual do custo com o banco de dados ao longo da vida do Sistema.

2.1.4. Controle de Redundância:

Possuir recursos que impeçam que dados duplicados sejam inseridos nas tabelas. Este é um requisito básico dos bancos relacionais;

2.1.5. Compartilhamento de Dados:

É o recurso que procura garantir que todas as pessoas, mesmo que sejam muitas, tenham acesso aos dados. Naturalmente, como já vimos, os bancos de dados relacionais

começam a perder performance à medida que o número de usuários simultâneos aumenta significativamente.

2.1.6. Representação de associações complexas:

Possuir ferramentas e recursos que permitam modelagens de alta complexidade. Normalização com a 4ª FN e 5ª FN por exemplo.

2.1.7. Garantia de restrições de Integridade:

Poder garantir integridade através de recursos de restrições de relacionamento e conteúdo de atributos; Integridade é a garantia que não temos tabelas fazendo referências a chaves em outras, e que estas chaves não existam. Vamos ver detalhadamente o conceito de integridade, mas a frente.

2.1.8. Recuperação de falhas:

Capacidade do banco de se recuperar de falhas como defeitos físicos, falhas na conexão de rede, falta de energia e/ou outros problemas que possam ocorrer durante ações como alteração e/ou inclusão de dados. A ideia de se recuperar é no mínimo manter o estado anterior ao erro sem danificar os dados pré-existentes;

2.1.9. Tratamento de transações:

Recurso muito importante que pode garantir que, durante uma sequência de atualizações, que são dependentes entre si. Exemplo: suponha um processamento em lote que cadastrar um pedido, dar baixa na mercadoria, criar a nota fiscal. Caso o Banco de Dados perceba no momento da criação da nota fiscal, por exemplo, que alguma operação é inválida, o Banco consegue desfazer as alterações de cadastrar o pedido e dar baixa no estoque. Os dados voltam a situação que estavam antes de iniciar o processo protegido pela transação.

2.1.10. Permitir o uso de stored procedures / triggers / views etc.:

Stored Procedures são funções cadastradas no banco de dados que podem ser executadas via chamada de aplicação. São importantes para agilizar e melhorar a performance de banco, mas dificultam a migração do SGBD se necessário. Triggers são

processos que rodam no banco, disparados por eventos configuráveis. Views são tabelas virtuais criadas a partir de tabelas físicas dos dados. São utilizadas para simplificar e tornar mais intuitivas algumas tabelas e relacionamentos físicos do Banco.

3. INDEPENDÊNCIA DOS DADOS

Objetivo:

Definir o conceito.

Introdução:

O conceito prega que os usuários tenham uma visão abstrata dos dados, encapsulando detalhes complexos e não relevantes no momento. Desta forma o desenvolvedor não precisa conhecer como os dados estão fisicamente armazenados para que possa trabalhar com eles.



Figura 2: Os desenvolvedores ao utilizar o SGBD utilizando uma linguagem simples e intuitiva como o SQL, por exemplo, conseguem realizar tarefas de grande complexidade sem que precisem ter necessariamente ter conhecimento da forma como o SGBD realiza o processo.

4. METADADOS

Objetivo:

Definir o conceito de Metadados.

Introdução:

Metadados ou, além dos dados, são os dados não do usuário nem do cliente, mas do próprio banco de dados. Existem tabelas internas que armazenam todas as informações do próprio Banco de Dados como todas as tabelas, seus campos, chaves, relacionamentos e todas as informações para que o SGBD saiba “quem é quem” e possa realizar as suas tarefas. O Banco de dados permite que o desenvolvedor acesse e consulte os metadados para alguma verificação ou conferência que se fizer necessário.

5. O MODELO DE DADOS REACIONAL

Objetivo:

Conhecer as características que distinguem Bancos de Dados Relacionais de outros Bancos de Dados

Introdução:

O modelo de Dados é uma das principais causa de sucesso ou insucesso dos projetos de Sistema da Informação. Um projeto cuidadoso e bem conduzido não é trivial nem barato. Envolve muitas horas de profissionais capacitados e treinados. Pensar com base na tecnologia de Bancos De Dados Relacional é o fundamento para um trabalho de sucesso. Erros na fase de modelagem, quase sempre trazem grandes prejuízos, estouro de prazos e de orçamento que nenhum projetista de *software* está disposto a vivenciar.

O que é Modelagem de dados?

É o trabalho de avaliar as regras do negócio, os requisitos levantados, os casos de uso, diagramas de classes, diagramas de objetos e outros documentos levantados na fase de análise para modelar, ou seja, determinar quais tabelas e que relacionamento elas devem ter, de forma que sejam capazes de representar corretamente o negócio que se pretende automatizar. Os modelos normalmente são claros o suficiente para que mesmo profissionais de outras áreas que não TI, consigam compreendê-los facilmente.

Exemplo:

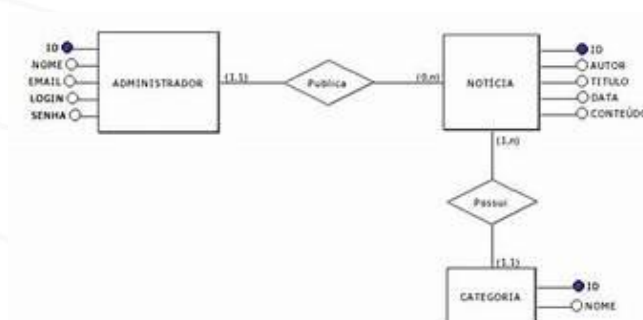


Figura 3: O Modelo de dados evidencia o relacionamento entre as entidades e seus atributos, mostrando alguns detalhes dos atributos e seus relacionamentos e cardinalidades

6. ETAPAS NO DESENVOLVIMENTO DE UM MODELO DE BANCO DE DADOS

Objetivo:

Descrever e estudar as principais etapas de desenvolvimento de um BD.

Introdução:

O desenvolvimento do Projeto de Banco de Dados envolve uma série de processos e tecnologias desenvolvidas no decorrer de vários anos em que aprendemos e aperfeiçoamos este trabalho.

A construção do modelo deve seguir quatro etapas básicas:

Especificação e Análise de requisitos (arquitetura de três níveis)

Todo o trabalho de análise do sistema incluindo todos os documentos gerados na fase de análise. (Requisitos do sistema, casos de usos, diagramas de classes, objeto, sequência etc.)

Projeto Conceitual (Alto nível de abstração MCD)

Baseado nos documentos da análise do projeto, levantamos os componentes: Entidades, atributos, relações que serão capazes de representar corretamente o mundo real do projeto.

Projeto Lógico (MER – Modelo Entidade Relacionamento)

Baseado na fase anterior, mas agora já desenhamos utilizando notações consagradas pelo mercado os componentes (Entidades, atributos, relações e cardinalidade etc.) do banco de dados representando toda a malha de relações entre as entidades levantadas;

Projeto Físico (DER – Diagrama Entidade Relacionamento)

Feito a partir do modelo Lógico, especifica como armazenar e acessar Banco de Dados. Aqui detalhamos as entidades, seus atributos chaves de todos os tipos, utilizamos um SGBD real para criar as tabelas, fazemos testes e fazemos inserção de dados para checar os relacionamentos e se o modelo lógico corresponde ao modelo físico

desenvolvido. É neste momento que necessitamos de um SGDB disponível para realizar esta atividade.

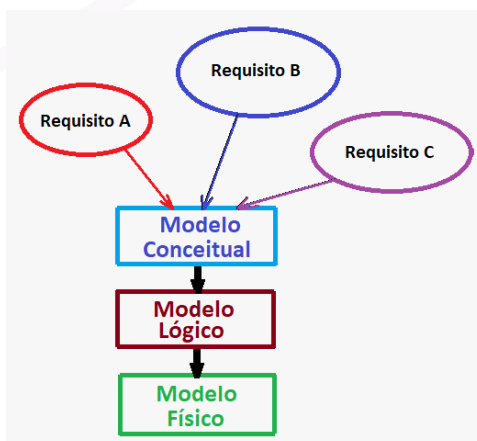


Figura 4. Diagrama representando o funcionamento da arquitetura de modelagem de três camadas; os requisitos são as entradas para conceber o modelo Conceitual, que evolui para o modelo lógico e finalmente chegamos ao modelo físico. A tentativa de queimar etapas quase sempre se mostra desastrosa. A experiência mostra, sem sombra de dúvidas, que o caminho é trabalhar seguindo os passos evolutivos já consagrados.

7. ENTIDADES OU TABELAS

Objetivo:

Conhecer o que é, e como se trabalha com tabelas.

Introdução:

O elemento básico em um banco de dados é a tabela. Nela organizamos e armazenamos os dados que precisamos.

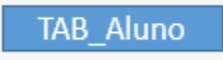
Dando nome aos “Bois”

O projetista do banco é responsável por dar nomes (batizar) as tabelas que devem representar da maneira mais clara possível a entidade que irá nomear. Existem regras de boas práticas e outras de limitação do próprio banco de dados que devemos seguir ao nomear as entidades do banco: São elas:

- Iniciar com letra (de preferência maiúscula);
- Os nomes das entidades devem ser únicos em um mesmo “esquema”;
- Usar palavras sempre no singular;
- Não podem conter espaços e devem evitar o uso de caracteres especiais (uso de acentuação também é desaconselhado, apesar da maioria dos bancos aceitarem os caracteres acentuados). O Caractere especial ‘_’ (*underscore ou underline*) é um caso de uso aceito e até recomendados;
- As empresas que fazem desenvolvimento, normalmente estabelecem padrões próprios para batizar seus identificadores. O profissional deverá se inteirar deles e procurar segui-los à risca, pois este comportamento ajuda bastante no entendimento dos diagramas e documentos por outros profissionais que venham a consultar a documentação.

Representação gráfica de uma tabela em um modelo de relacionamento (MER)

Representação no MER: Retângulo com o nome da Entidade Dentro



TAB_Aluno

Instâncias de Entidade

- São os dados que populam uma entidade
- Exemplo: Se a Entidade for TAB_Aluno, as Instâncias de Entidade poderiam ser: Maria, José, Antônio etc. Ou seja, as instâncias são os dados gravados na tabela. Uma tabela limpa, recém-criada, não possui instâncias;

8. ATRIBUTO OU CAMPO

Objetivo:

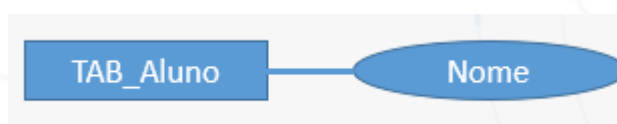
Conhecer as peculiaridades dos atributos e como eles compõem com as tabelas o principal componente de um Banco de Dados

Introdução:

O estudo dos atributos, seus tipos e formas de uso e criação representam grande parte do trabalho de modelagem. Conhecer sua aplicação e peculiaridades é fundamental para concepção de um modelo de dados eficaz. Os atributos descrevem características das entidades. Os atributos possuem tipo de dados (domínio), nome e valor específico. Exemplo: modelo, fabricante, cor, placa, ano fabricação, número e renavan etc.

Principais tipos de atributos

- Simples ou Atômico;
- É o atributo básico, caracterizado por representar uma informação única, isolada. Devemos procurar definir todos os atributos, no “final das contas” para um atributo atômico;
- Não possui Características especiais e são indivisíveis (atômicos)
- **Exemplos:** Data_Nascimento, Curso, Nome etc.;
- Representação no MER:



- Composto (Exemplo: Endereço);
- São atributos que possuem tipicamente mais de uma informação. O exemplo clássico é o endereço. O endereço normalmente é composto de várias informações como: logradouro, número, bairro, cidade, uf, complemento CEP etc. O cadastro de nome das pessoas também é considerado por muitos modelos como compostos. Por isso os projetistas costumam separar o nome do cliente em nome, sobrenome, nome do meio, apelido etc.
- Multivalorado (Exemplo: Telefone(s) de uma pessoa ou empresa);

São campos que podem ser preenchidos com mais de um valor, mas normalmente do mesmo tipo. Se diferenciam dos compostos por este motivo. Um outro exemplo poderia ser uma coluna de banco de dados que armazenasse os ganhadores de um jogo que o resultado pode ser empate.

Representação no MER:



- Determinante;

Identifica de forma única um atributo, ou seja, não pode haver dados repetidos. É indicado sublinhando-se o nome do atributo. Exemplos: CNPJ, CPF, Codigo_Fornecedor, Numero_Matricula etc. Não é necessariamente uma chave, mas pode ser usado como uma.

Representação no MER:



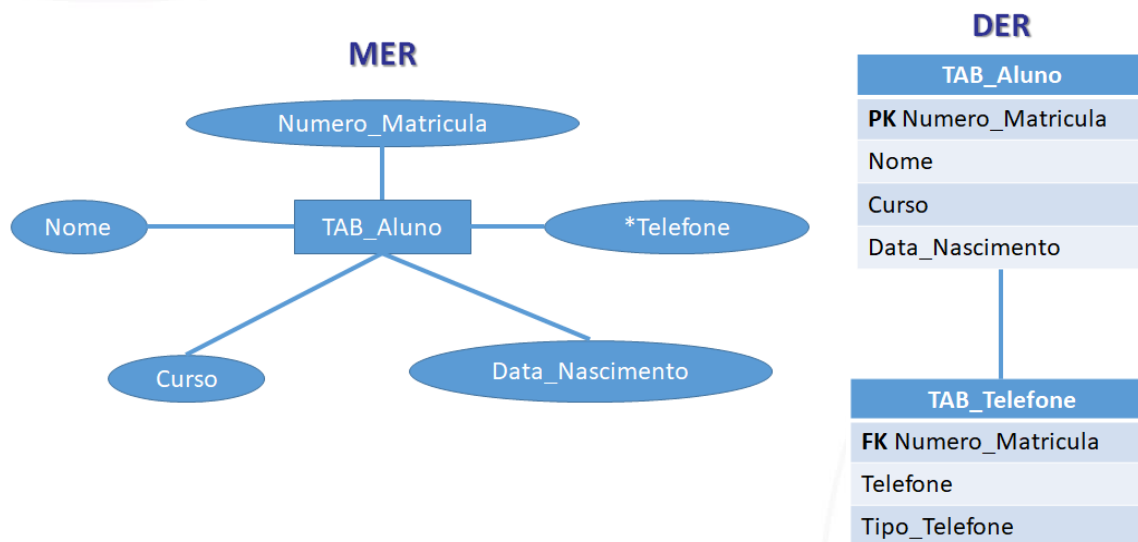
- Identificador.
- Ou Chave, é utilizado para localizar e/ou ligar registros no Banco de Dados. As chaves podem ser:
- Únicas: O valor da chave é único na entidade;
- Não únicas: São chaves que normalmente fazem parte de chaves compostas. Chaves compostas são chaves “montadas”, ou concatenadas a partir de dois ou mais atributos que sozinhos não são determinantes. Exemplo: Uma tabela Aluno_Curso pode possuir dois atributos distintos: Codigo_Aluno e Codigo_Curso, com eles podemos agrupar todos os alunos do curso, ou todos os cursos do aluno utilizando ora uma, ora outra chave. Mas a chave primária, ou seja, única da tabela, é a composição das duas chaves: codigo_curso, codigo_aluno;

Podemos representar uma entidade e seus atributos da seguinte forma:

TAB_Aluno (Numero_Matricula, Nome, Curso, Data_Nascimento, *Telefone)

- Note o sublinhado no Numero_Matricula (atributo identificador) e o asterisco ao lado do Telefone (atributo multivalorado).

Representação MER e DER de uma tabela com seus atributos:



9. RELACIONAMENTOS

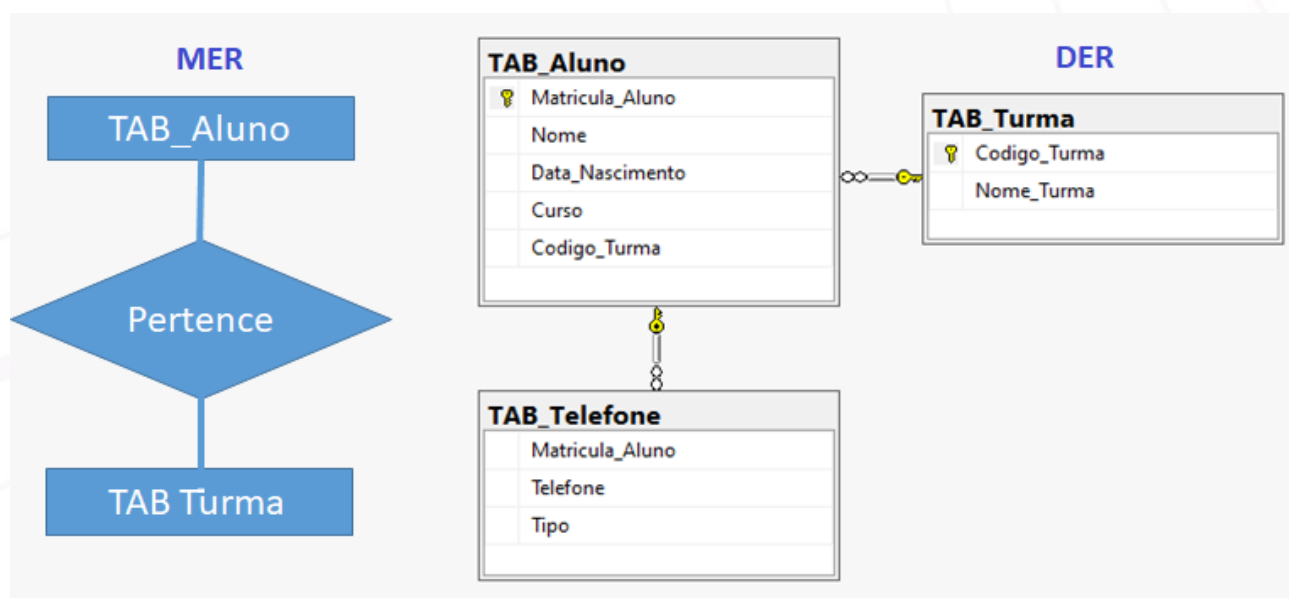
Objetivo:

Mostrar o que são e como funcionam os relacionamentos entre entidades em um banco relacional.

Introdução:

Os relacionamentos são responsáveis por ligar as entidades por meio de chaves que permitem que o SGBD recupere registros e consiga reunir informações distribuídas em outras entidades, mas que possuem um índice comum. Estas conexões são feitas por meio de Relacionamentos. Trata-se de uma estrutura que indica a associação de elementos de uma ou mais entidades;

Representamos um relacionamento em um MER por meio de um **Losango** que conecta uma ou mais entidades. No caso do DER observe, o relacionamento entre TAB_Aluno e TAB_Turma é representado por uma ligação simples. A TAB_Telefone aparece pelo desdobramento do campo multivalorado Telefone e está ligada a tabela TAB_Aluno pela chave estrangeira Matricula_Aluno.

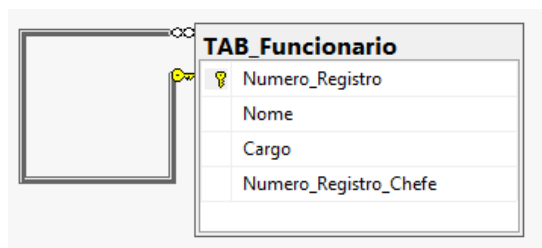


Grau de Relacionamento

O Grau de relacionamento define o número de entidades que participam do relacionamento.

Eles pode ser:

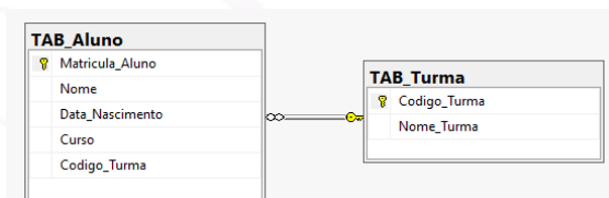
- Unário: A Entidade faz referência a ela mesma: Exemplo: Em uma tabela de funcionários, existe uma referência de outro funcionário que é o chefe;



Numero_Regis...	Nome	Cargo	Numero_Regis...
1000	Maria Do Carmo	Secretária Administrativa	NULL
2000	Jose do Patrocinio	Atendente	1000
3000	Diego	Telefonista	1000
NULL	NULL	NULL	NULL

Observe que os funcionários José do Patrocínio e Diego estão referenciando como chefe a funcionária de código 1000. Como a Maria do Carmo tem NULL na chave Numero_Registro_Chefe, isso indica que ela NÃO tem chefe (pelo menos neste banco).

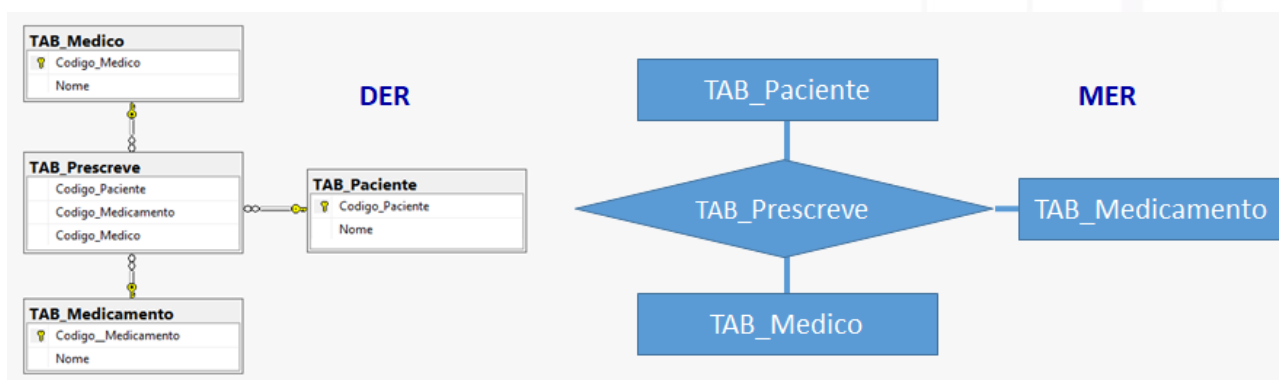
- Binários: Duas tabelas se referenciam. Um é dona da chave primária (PK) e a outra da chave estrangeira (FK).



Matricula_Aluno	Nome	Data_Nascimento	Curso	Codigo_Turma	Codigo_Turma	Nome_Turma
1000	Diego ...	1995-01-07	Fundamental 2 ...	4000	4000	Quinta Série A
2000	Carlos ...	2000-02-12	Fundamental 2 ...	4000	5000	Quinta Série B
3000		2010-12-23	Fundamental 2 ...	5500	5500	Sexta série A
4000	Manoel ...	2002-07-21	Fundamental 2 ...	5000	6000	Sexta série B
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Podemos observar as equivalências entre as chaves primária na tabela da direita, TAB_Turma, com as chaves estrangeiras na tabela da esquerda (TAB_Aluno). Podemos notar que a turma de código 6000 (Sexta série B) não tem referência na tabela de alunos. Isso quer dizer que eu posso ter turmas com zero alunos.

- Ternários ou (nÁrios – 3 ou mais entidades): São relacionamentos com outras duas ou mais tabelas. No exemplo vemos a situação de prescrição de medicamentos. Claramente temos três entidades. O Médico, o Paciente e o Medicamento. No MER observamos as três entidades e um relacionamento unindo as três. Já no DER, temos quatro tabelas. Isso porque no modelo físico, quando temos relacionamentos de mais de duas tabelas, não escapamos de precisar criar uma tabela física para viabilizar o relacionamento. Reparem que existe uma tabela TAB_Prescreve com três chaves estrangeiras. Uma para cada uma das tabelas relacionadas.



10. MODELOS DE REPRESENTAÇÃO DE ENTIDADES

Objetivo:

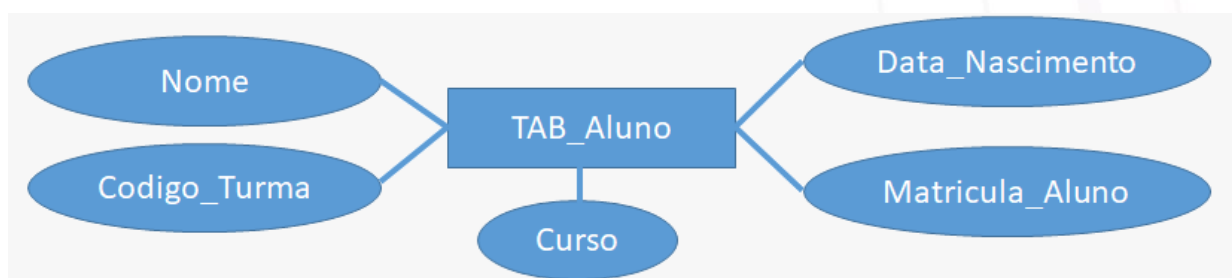
Conhecer mais aspectos da modelagem Lógica e física;

Introdução:

Nos exemplos vistos nos tópicos anteriores, estamos sendo apresentados as ferramentas de modelagem que estão a nossa disposição. Vamos conhecer mais algumas e aprender como usá-las

Modelo Lógico (MLD - Médio nível de abstração)

- Representado utilizando-se as notações de retângulo/elipse e losango
- Determina as entidades envolvidas (tabelas) e seus relacionamentos com as demais entidades do projeto; não é necessário muito conhecimento técnico para entender o modelo lógico. Independe ainda do SGBD escolhido para desenvolver a aplicação;



Modelo Físico (MFD - Baixo nível de abstração)

- Detalha todos os atributos, entidades, registros;
- Representado normalmente na forma de tabelas onde as Colunas são as propriedades dos Atributos e as linhas são os atributos das tabelas propriamente ditos;

	Nome da Coluna	Tipo de Dados	Permitir Nul...
🔑	Matricula_Aluno	int	<input type="checkbox"/>
	Nome	nchar(50)	<input checked="" type="checkbox"/>
	Data_Nascimento	date	<input checked="" type="checkbox"/>
	Curso	nchar(50)	<input checked="" type="checkbox"/>
▶	Codigão_Turma	int	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

11. CARDINALIDADE

Objetivo:

Compreender, determinar e utilizar corretamente os conceitos de Cardinalidade;

Introdução:

Nos modelos e diagramas dos bancos de dados, principalmente nos modelos lógicos, é importante para o entendimento no negócio e construção correta do modelo físico, conhecer a relação quantitativa que existe entre os relacionamentos. Vamos estudar algumas convenções que utilizamos para representá-las.

Cardinalidade refere-se ao número máximo de vezes que a instância em uma entidade pode ser relacionada a instâncias de outra entidade.

Na prática a cardinalidade estabelece como pode variar as ligações entre Entidades, determinando quantas instâncias, no mínimo e no máximo, são possíveis existirem na reação em análise.

Exemplo: A relação entre as tabelas TAB_Turma e TAB_Aluno podemos indicar um número mínimo e máximo de alunos que uma turma pode suportar, ou seja, quantas vagas temos nas turmas.

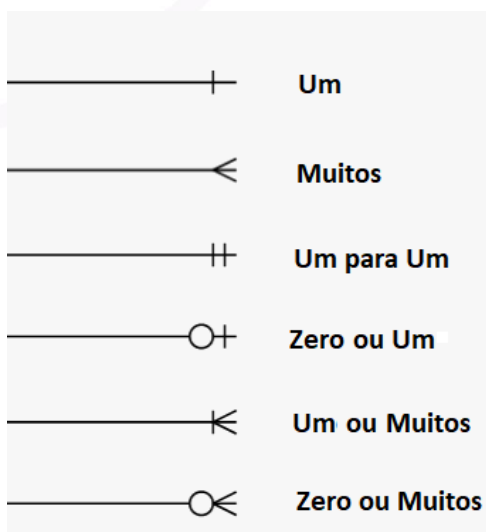
Notação de Peter Chen:



Notação Pé de Galinha



A convenção mais utilizada é conhecida como Pé de Galinha (*Crow's Foot*):

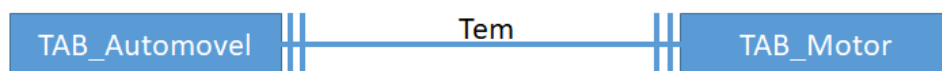


Existem muitas outras notações para representar Cardinalidade. Podemos citar: UML, OMT, IDEF, Bachman etc.

Cardinalidade 1 para 1

Exemplo:

- Pé de Galinha



- Peter Chen

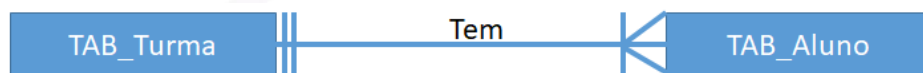


Intuitivamente sabemos que um automóvel possui um motor e um motor serve apenas um automóvel. Neste exemplo vemos duas Entidades distintas relacionadas em uma cardinalidade Um para Um.

Cardinalidade 1 para muitos

Exemplo:

- Pé de Galinha



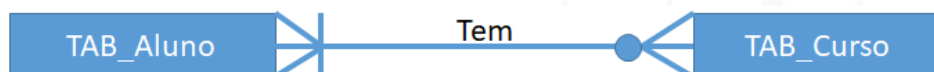
- Peter Chen



Neste exemplo percebemos que uma turma, para ser uma turma, deverá possuir pelo menos um aluno com uma quantidade máxima, que no caso, é ilimitada. Na notação Pé de Galinha não temos uma forma fácil de indicar os limites máximos e mínimos, mas repare que na notação de Peter Chen, poderíamos indicar a relação de (5,40) ou seja, no mínimo 5 e no máximo 40 alunos.

Cardinalidade muitos para muitos

- Pé de Galinha



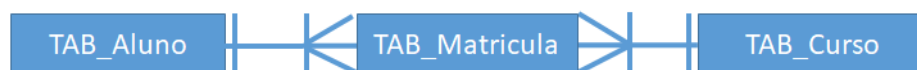
- Peter Chen



Infelizmente não é possível conseguir uma cardinalidade N pra M apenas relacionando as tabelas.

Para realizar isto na prática, precisamos criar uma tabela intermediária. Ficaria assim:

- Pé de Galinha



Neste exemplo, estamos afirmando que um aluno pode se matricular em mais de um curso, e um curso pode ter nenhum ou vários alunos matriculados... Um Curso sem nenhum aluno matriculado é um curso?

12. Restrições de Integridade

Objetivo:

Discutir o significado, as aplicações das Restrições de Integridade, saber seus tipos e examinar exemplos.

Introdução:

As restrições de Integridade são a principal ferramenta utilizada para garantir que os dados armazenados nos bancos de dados são e continuarão sendo íntegros, independente do tempo e intensidade do uso que fazemos deles.

Os tipos de integridade que veremos são:

12.1. Referencial;

Procura assegurar que valores de uma coluna em uma tabela são válidos baseados em uma outra tabela relacionada;

Exemplo 1: Se, em uma tabela TAB_Matricula por exemplo, tentamos cadastrar o aluno matrícula '54.675' da tabela TAB_Aluno, o Banco de Dados só irá permitir a inserção do novo registro na tabela TAB_Matricula se na tabela TAB_Aluno previamente já existir um aluno de matrícula '54.675'. Caso contrário, o SGBD deverá retornar um erro impedindo assim a conclusão da operação.

Esta restrição funciona também quando desejamos remover do banco uma instância (registro) que é referenciada por outro registro em outra entidade.

Exemplo 2: Considere que a tabela TAB_Matricula faz referência ao aluno matrícula '54.675' da tabela TAB_Aluno. Caso alguém tente retirar da base o registro do aluno matrícula '54.675' da tabela TAB_Aluno, o banco não irá permitir pois existe um registro de uma tabela relacionada, TAB_Matricula, que faz referência a este aluno.

12.2. de Domínio;

Valores inseridos em uma coluna devem sempre obedecer à definição dos valores que são permitidos para esta coluna – Os valores de domínio.

Exemplo 1: em uma coluna que armazena preços de mercadorias (moeda), os valores admitidos são do domínio moeda - ou seja, apenas números do tipo moeda;

Exemplo 2: Em um campo Nome de uma tabela que foi projetado para apenas 50 caracteres, não deverá permitir o cadastro de nomes de mais de 50 caracteres (incluindo espaços);

Fatores considerados nas restrições de Domínio:

- Tipo de Dados do Campo;
- Representação Interna do tipo de dado;
- Presença ou não do Dados;
- Intervalo de valores de domínio;
- Conjunto de valores discretos.

12.3. de Vazio

Este tipo de integridade informa se a coluna é obrigatória ou opcional – ou seja, se é possível não inserir um valor na coluna. Observe que uma coluna de chave primária, por definição, nunca poderá estar vazia para nenhum registro. Neste caso a restrição de vazio está implícita a qualquer chave primária.

Exemplo: Em nossa tabela TAB_Aluno, podemos estabelecer a restrição que a data de nascimento é um atributo obrigatório, ou seja, não serão aceitos registros sem a informação da data de nascimento do aluno (Vazio na data). Desta forma, o próprio SGBD retornará um erro e não permitirá a inserção de registros caso a data de nascimento não esteja sendo informada. Isso garante que, em nenhuma hipótese, teremos alunos cadastrados sem a respectiva data de nascimento.

12.4. de Chave

Os valores inseridos na coluna de chave primária (PK) devem ser sempre únicos, não admitindo-se repetições nesses valores. Desta forma, garantimos que as tuplas (registros) serão sempre distintas. Com vimos, os valores das chaves primárias também não podem ser nulos.

Exemplo 1: Em nossa tabela TAB_Aluno não podemos cadastrar mais que um aluno com matrícula '54.675'. Caso fosse permitido, ao buscar o aluno matrícula '54.675' o banco retornaria mais de um aluno, o que seria uma violação de integridade. O campo de matrícula da tabela TAB_Aluno deverá ser necessariamente único.

Exemplo 2: Podemos restringir qualquer campo como único, sem que ele seja necessariamente uma chave. Isso poderia acontecer na tabela TAB_Aluno que tenha um campo CPF_Aluno. O Campo matricula seria a chave primária mas o campo CPF_Aluno seria único. Assim podemos evitar erros de cadastro duplicado de alunos por exemplo.

12.5. Definida pelo Usuário

Diz respeito a regras de negócio específicas que são definidas pelo usuário do Banco de Dados

Exemplo: Pode-se definir que uma coluna somente aceitará um conjunto restrito de valores. Uma aplicação para esta restrição poderia ser na tabela TAB_Telefone (utilizada no exemplo de 'Atributo Multivalorado'), o atributo **tipo** poderia receber a restrição de aceitar apenas os conteúdos: 'Celular', 'Fixo', 'Comercial', 'PABX'.

13. ANOMALIAS DE ATUALIZAÇÃO

Objetivo:

Conhecer o que consideramos anomalias de atualização e entender como estão relacionadas com as restrições de integridade

Introdução:

Anomalias são problemas que podem acontecer em Bancos de Dados mal projetados e/ou não normalizados, muitas vezes ocorrendo por excesso de dados armazenados em uma mesma tabela.

As anomalias de atualização são classificadas como:

13.1. Inserção

Não deve ser possível inserir um dado a não ser que outro dado esteja disponível

Exemplo: Não deve ser permitido matricular um novo aluno sem que o aluno já esteja cadastrado previamente;

13.2. Exclusão

Ao excluirmos um registro, dados referentes em outras tabelas são excluídos.

Exemplo: Se excluirmos uma turma, os alunos da turma devem ser excluídos também;

13.3. Alteração

Ao excluirmos um registro, dados referentes em outras tabelas são excluídos

Exemplo: Se excluirmos uma turma, os alunos da turma devem ser excluídos também;

14. Normalização

Objetivo:

Conhecer as Formas Normais utilizadas na modelagem de Bancos de Dados, suas aplicações, técnicas de aplicação e sua importância na qualidade final de um projeto de banco de dados.

Introdução:

O processo de Normalização nos permite colocar nosso projeto de banco de dados “nos trilhos” utilizando técnicas que são praticamente infalíveis e que veremos a seguir.

Proposto em 1972 por Codd, aplica a um esquema de relação uma série de testes para certificar que ele satisfaça uma dada Forma Normal. (FN)

Cada uma das formas normais aprimora mais uma etapa na direção da Normalização. As formas normais devem ser aplicadas na sequência e uma serve de pré-requisito para a forma normal seguinte;

Codd propôs originalmente 3 (1ª, 2ª e 3ª) formas normais. Posteriormente a 3ªFN foi revisada e uma definição foi proposta por Boyce e Codd, finalmente ficando conhecida como Forma Normal de Boyce-Codd (FNBC) ou ainda como 4ªFN.

Objetivos da Normalização

Analisar esquemas de relação (tabelas) com base em suas dependências funcionais e chaves primárias para:

- Minimizar redundâncias;
- Minimizar anomalias de inserção, exclusão e modificação.

O procedimento de normalizar consiste em decompor as relações em esquemas de menores e mais simples relações que atendam as especificações das formas normais.

O que é Dependência Funcional

Definição: Uma chave primária em uma relação determina funcionalmente todos os outros atributos não chave na linha.

Exemplo 1: Considere que na TAB_Aluno existe um atributo chamado Media_Avaliacao. Para que exista uma média do aluno é necessário que exista um aluno, portanto, Media_Avaliacao depende deCodigo_Aluno que é a chave primária da tabela TAB_Aluno. Chamamos de atributo Media_Avaliacao de Dependente e o atributo Codigo_Aluno o atributo determinante.

Codigo_Aluno → **Media_Avaliacao**

Exemplo 2: Pensando em uma tabela de pedidos; O prazo de entrega de um pedido depende do número do pedido considerado:

Numero_Pedido → **Prazo_Entrega_Pedido**

Existem alguns tipos específicos de dependência funcional:

- Dependência Funcional Total
- Dependência Funcional Parcial
- Dependência Funcional Transitiva
- Dependência Funcional Multivalorada

As formas Normais

14.1. Primeira Forma Normal

Historicamente a primeira forma normal procura solucionar o problema dos atributos multivalorados, compostos e suas combinações;

O Domínio de um atributo deve incluir apenas valores atômicos (indivisíveis), e o valor de qualquer atributo em uma tupla (linha) deve ser único valor do domínio deste atributo;

Uma tabela está na primeira forma normal quando:

- Possui somente valores atômicos;
- Não há grupo de atributos repetidos;

- Existe uma chave primária;
- As relações não possuem atributos multivalorados ou relações aninhadas.

Exemplo:

Vamos Preencher nossa tabela com alguns dados fictícios para poder analisar a situação:

TAB_Aluno		TAB_Aluno			
Codigo_Aluno		Codigo_Aluno	541234	558766	587648
Nome		Nome	Antônio José	Maria Do Carmo	Josué Matheus
Codigo_Turma		Codigo_Turma	1000	1000	1000
Telefone		Telefone	(13) 99766-5543		(13) 99766-5741
Endereco		Endereco	(13) 99765-6677	(13) 99978-7766	(13) 99765-3469
Data_Nascimento		Data_Nascimento			(13) 98766-3856
		Endereco	Rua 4, Embaré, CEP: 11075-520	Rua Antônio 10, Carlos, Vila Belmiro, CEP: 11040-654	Rua do Lavradio Número 5, apt. 203, São Bernardo
		Data_Nascimento	10/10/1998	01/04/2001	23/09/2001

- Que campos podem ser desdobrados?
- Existe algum campo multivalorado?
- Existe algum campo não atômico que possa ser desdobrado?

A solução para 1ªFN

Esta é nossa solução para a colocação da tabela TAB_Aluno na primeira forma normal. Não existem mais campos multivalorados (telefone) nem compostos (endereco)

Como pode ser observado, a solução para o campo multivalorado foi criar uma entidade (TAB_Telefone) para armazenar os telefones e o desmembramento da coluna endereço em diversas colunas com dados de endereço atomizados.

TAB_Aluno			
Codigo_Aluno	541234	558766	587648
Nome	Antônio José	Maria Do Carmo	Josué Matheus
Codigo_Turma	1000	1000	1000
Logradouro	Rua 4	Rua Antônio Carlos 10	Rua do Lavradio 5
Complemento	Número 5		Apt. 203
Bairro	Embaré	Vila Belmiro	
CEP	11075-520	11040-654	
Cidade	Santos	Santos	São Bernardo
Data_Nascimento	10/10/1998	01/04/2001	23/09/2001

TAB_Telefone	
Codigo_Aluno	Telefone
541234	(13) 99766-5543
541234	(13) 99765-6677
558766	(13) 99978-7766
587648	(13) 99766-5741
587648	(13) 99765-3469
587648	(13) 98766-3856

TAB_Telefone						
CodigoAluno	541234	541234	558766	587648	587648	587648
Telefone	(13) 99766-5543	(13) 99765-6677	(13) 99798-7766	(13) 99766-5741	(13) 99765-3469	(13) 98766-3856

Repare que as tabelas preenchidas com dados fictícios nos ajudam a encontrar os problemas que precisam ser normalizados.

14.2. Segunda Forma Normal

- Baseada no conceito de dependência Funcional Total;
- Um sistema de Relação R está na segunda forma normal se cada atributo não chave de R for total e funcionalmente dependente de PK de R;
- Para testar a 2ª FN, verificamos as dependências funcionais cujos atributos fazem parte da chave primária;
- Caso a PK tenha um único atributo, não existe nada a fazer para adequar a tabela a 2FN.

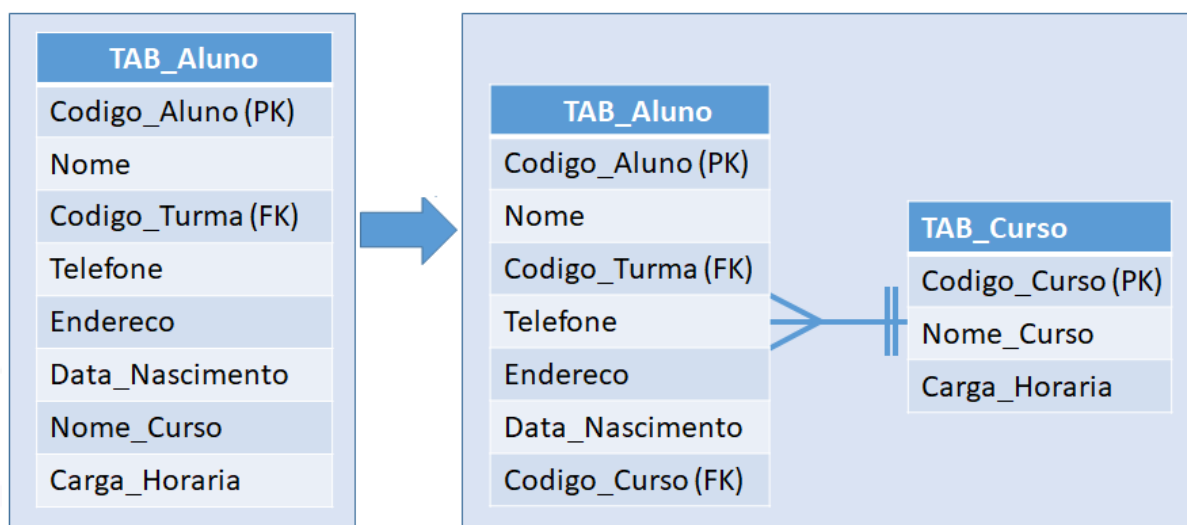
14.2.1. Uma tabela está na 2ª FN se:

- Já esteja na 1ª FN
- Todos os atributos não-chave são funcionalmente dependentes de **todas as partes** da chave primária da tabela;
- Não existem dependências parciais;
- Caso contrário, deve-se gerar uma nova tabela com os dados;
- Deve-se criar uma relação para cada chave PK ou combinação de atributos que forem determinantes em uma dependência funcional;
- Este atributo será a PK na nova tabela;
- Mova os atributos não-chave dependentes desta PK para a nova tabela;

Nota: Pense que em uma tabela não devemos “misturar assuntos” ou seja, todos os atributos de uma tabela devem ser referenciados claramente pela chave primária. Qualquer atributo que “destoe” no assunto, provavelmente deverá ser movido para uma outra tabela (provavelmente uma nova tabela)

Exemplo 1: Pense na tabela TAB_Aluno onde foram acrescentados os campos: **Nome_Curso, Carga_Horaria**. Se mais de um aluno estiver cursando o mesmo curso teremos a mesma informação: **Nome_Curso e Carga_Horaria** repetidos em todos os alunos que estiverem matriculados neste curso. Fica claro que temos aqui redundância de dados. Se for necessário mudar a carga horária ou o nome do curso, teremos que atualizar todos os registros dos alunos. O que seria algo sem sentido. Claramente **Nome_Curso e Carga_Horaria** são **outro assunto**. Parece uma boa ideia criar uma nova tabela: **TAB_Curso (Codigo_Curso (PK), Nome_Curso , Carga_Horaria)**. Para associar o Curso aos alunos. Seria também o caso de criar uma FK **Codigo_Curso** na tabela **TAB_Aluno**

OBS: Um atributo-chave é uma PK ou parte de uma PK composta.



14.3 Terceira Forma Normal

- Baseada no conceito de Dependência Transitiva;
- A relação não deve ter um atributo não chave determinado funcionalmente por outro atributo não chave (ou conjunto);
- Não deve haver dependência transitiva de um atributo não chave sobre a PK;
- Deve-se decompor e montar uma nova relação que inclua os atributos não chave que determinam funcionalmente outros atributos não chave;

14.3.1. Uma tabela está na 3ª FN se as condições forem satisfeitas:

- Estiver na 2ª FN;
- Não existirem dependências transitivas (dependência funcional entre dois ou mais atributos não chave);
- Se nenhuma coluna não chave depender de outra coluna não chave.

14.3.2. Procedimentos para 3ª FN

- Para cada atributo, ou grupo de atributos não chave que for um determinante na relação, crie uma tabela;
- Este atributo será PK na nova relação;
- Mova todos os atributos que são dependentes funcionalmente do atributo chave para a nova tabela;
- O Atributo PK da nova relação, fica também na tabela original e passará a ser uma chave estrangeira para associar a nova tabela com a tabela original.

Exemplo:

Tabelas na 2ª FN

tbl_Venda
<u>Nota_Fiscal</u>
Cod_Vendedor
Nome_Vendedor
Cod_Produto
Qtde_Vendida



Tabela Populada

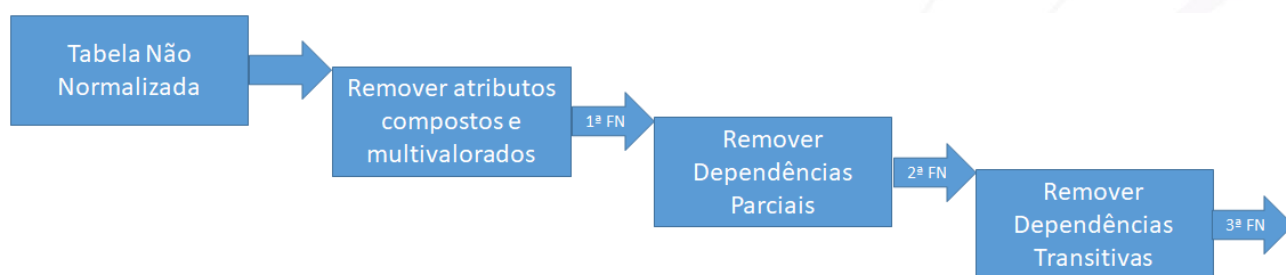
tbl_Venda				
<u>Nota_Fiscal</u>	Cod_Vendedor	Nome_Vendedor	Cod_Produto	Qtde_vendida
15326	002	Leila	132	10
15327	006	Ana	153	12
15328	002	Leila	143	11
15329	009	Fábio	132	9
15330	007	Renato	153	12

Normalizada na 3FN

tbl_Venda			
Nota_Fiscal	Cod_Vendedor	Cod_Produto	Qtde_vendida
15326	002	132	10
15327	006	153	12
15328	002	143	11
15329	009	132	9
15330	007	153	12

tbl_Vendedor	
Cod_Vendedor	Nome_Vendedor
002	Leila
006	Ana
007	Renato
009	Fábio

14.3.3. Em resumo, as etapas para chegar até a 3ª FN:



14.4 Quarta forma normal FNBC (Forma Normal de Boyce-Codd)

A definição original da 3ª FN de Codd com uma relação que:

- Tivesse duas ou mais chaves candidatas;
- Essas chaves candidatas fossem compostas;
- Elas tivessem sobreposição, ou seja, atributos em comum.

Caso essas condições não ocorram em uma tabela, basta chegar até a 3ª FN.

Dizemos que uma relação está em FNBC se e somente se os únicos determinantes são chaves candidatas.

Para normalizar uma tabela até FNBC devemos decompor a tabela com os passos a seguir:

- Encontrar uma dependência funcional não-trivial $X \rightarrow Y$ que viole a condição de FNBC. X não deve ser uma PK;
- Dividir a tabela em duas:
 - Uma com os atributos XY, ou seja, todos os atributos da dependência;
 - Outra com os atributos X juntamente com os atributos restantes da tabela original.

Exemplo 1:

Considere uma tabela: TAB_Fornece {Codigo_Fornecedor, Nome_Fornecedor, Codigo_Produto, Quantidade_Produto}

TAB_Fornece			
Codigo_Fornecedor			
Nome_Fornecedor			
Codigo_Produto			
Quantidade_Produto			

Codigo_Fornecedor	Nome_Fornecedor	Codigo_Produto	Quantidade_Produto
1000	Nestle	200	400
2000	Garoto	300	250
3000	Lacta	400	1000
2000	Garoto	200	550
1000	Nestle	400	700

Solução**Exemplo 2:****Tabela não normalizada**

TAB_Aluno_Disciplina_Professor		
Codigo_Aluno	Disciplina	Professor
500	Matemática	Arthur
501	Física	Helio
501	História	Carlos
503	Matemática	Arthur
503	Historia	Carlos
503	Física	Maria

Restrições:

- Cada estudante aprende uma disciplina lecionada por um professor;
- Cada Professor leciona apenas uma disciplina, mas uma disciplina pode ser lecionada por mais de um professor;

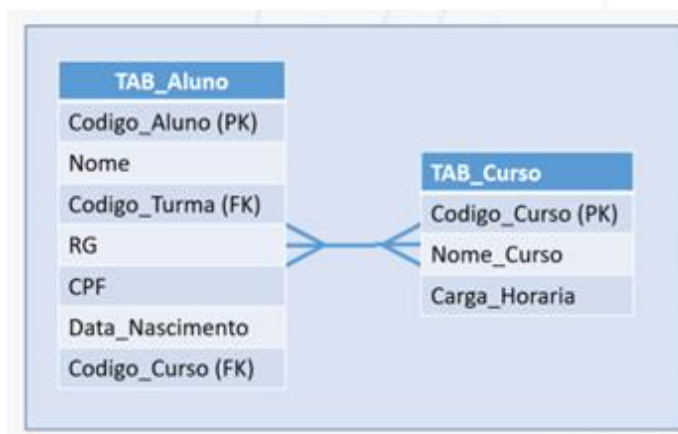
Solução

TAB_Aluno_Professor		TAB_Professor_Disciplina	
Codigo_Aluno	Professor	Professor	Disciplina
500	Arthur	Arthur	Matemática
501	Helio	Helio	Física
501	Carlos	Carlos	História
503	Arthur	Maria	Física
503	Carlos		
503	Maria		

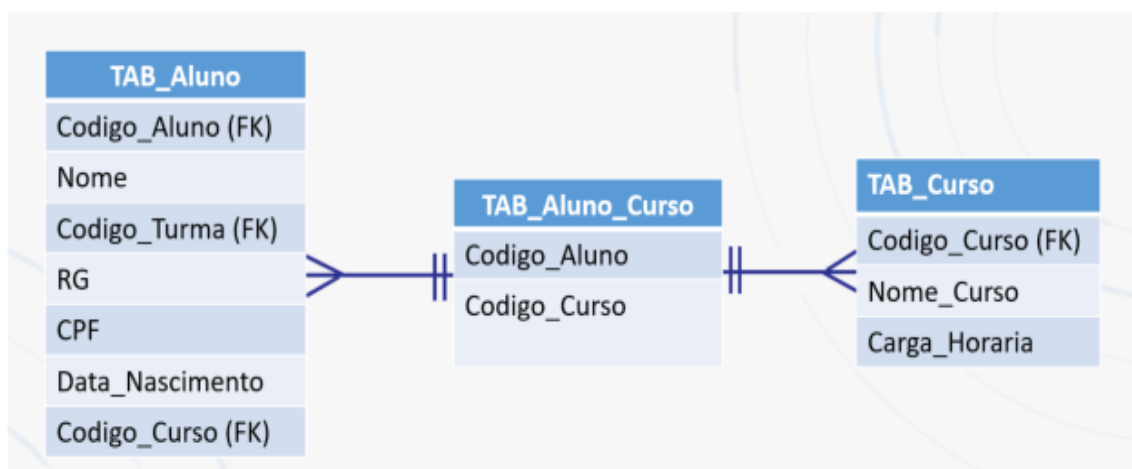
Exemplo 3:

Vamos pensar agora na seguinte situação: Um aluno pode se matricular em um ou mais cursos, e um curso pode ser cursado por um ou mais alunos. Temos aqui uma relação N x M.

Nossa relação poderia ser assim:



A solução seria criar uma terceira tabela cuja única finalidade seria ligar as duas.



15. DICIONÁRIO DE DADOS

Objetivo:

Apresentar o item de documentação da Modelagem: Dicionário de dados, mostrando exemplos elucidativos.

Introdução:

Esquema do Banco de Dados ou Repositório de Metadados, são documentos usados para armazenar/documentar informações sobre conteúdo, formato e estrutura de um banco de dados assim como os relacionamentos entre seus elementos. Os tipos de documentos são:

- Entidades
- Atributos
- Relacionamentos

É fundamental que o Dicionário de Dados seja constantemente atualizado à medida que o projeto evolui. O Dicionário de Dados vai auxiliar a minimizar os erros quando formos criar as estruturas físicas no computador.

Entidades:

Exemplo:

Tabela	Relacionamento	Nome do Relacionamento	Descrição
TAB_Aluno	TAB_Curso	Frequenta	Tabela para cadastro dos alunos
	TAB_Professor	Leciona	
TAB_Professor	TAB_Aluno	Leciona	Tabela para cadastro dos professores
TAB_Curso	TAB_Aluno	Frequenta	Tabela de Cursos



Atributos:

Exemplo:

Tabela	Nome da Coluna	Tipo de Dados	Comprimento	Restrições	Valor Padrão	Descrição
TAB_Aluno	Codigo_Aluno	Inteiro	4 bytes	PK, Not NULL	NA	Registro do aluno
	Nome	Texto	50 bytes	NOT NULL	NA	Nome completo do aluno
						Código da turma que o aluno esta matriculado. O sistema deve permitir NULL caso o aluno ainda não esteja matriculado em nenhum curso.
	Codigo_Turma	Inteiro	4 bytes	FK	NA	
	RG	Texto	10 bytes	NOT NULL	NA	RG do aluno, (sem outra referência) e sem formatação
	CPF	Texto	11 bytes	NOT NULL	NA	CPF do aluno sem formatação
	Data_Nascimento	Data	8 bytes	NOT NULL	NA	Data de nascimento do aluno
TAB_Curso	Codigo_Curso	Inteiro	4 bytes	PK, Not NULL	NA	Código do curso (Sequencial)
	Nome_Curso	Texto	50 bytes	NOT NULL	NA	Nome do curso
	Carga Horária	Real	4 bytes	NOT NULL	NA	Número de horas (uso de fração de hora em decimal. Exemplo: 1:30H deve ser registrado como: 1.5)

- Utilize o campo de Descrição para detalhar alguma coisa que for pertinente.
- A Coluna de Valor padrão foi toda preenchida com NA (Não se aplica), mas é bem útil quando precisamos definir um valor inicial default para o campo.

Exemplo: Um campo de número de dependentes. É uma boa ideia se for inicializado com 0 (zero).

Relacionamentos:

Exemplo:

Relacionamento	Tabela 1 FK	Tabela 2 - PK	Descrição
Frequenta	TAB_Aluno	TAB_Curso	Relacionamento que descre o curso que o aluno frequenta
Leciona	TAB_Aluno	TAB_Professor	Relacionamento que descreve o professor que leciona para aluno



16. A LINGUAGEM SQL NOS BANCOS RELACIONAIS

Objetivo:

Ensinar e exemplificar comandos básicos de SQL

Introdução:

A linguagem SQL é o recurso mais conhecido por administradores de dados, e programadores para a execução de comandos em bancos de dados relacionais. É por meio dela que criamos tabelas, colunas, índices, atribuímos permissões a usuários, bem como realizamos consultas, e alterações dos dados. Enfim, é utilizando a SQL que “conversamos” com o banco de dados e conseguimos o que precisamos dele.

Vamos nos concentrar nas instruções que permitem consultar, inserir, alterar e excluir registros. Muitas outras instruções existem para criar tabelas, chaves, índices, relacionar tabelas, alterar estruturas e muitas outras que não vamos tratar neste curso. O motivo é que praticamente todos os SGBS's de mercado possuem ferramentas de alto nível que geram automaticamente estes scripts para nós. De qualquer forma é fácil encontrar a documentação e exemplos caso você tenha interesse de se aprofundar. Também vamos exemplificar os comandos utilizando nas formas mais utilizadas. Existem muitas maneiras diferentes de realizar alguns dos comandos que vamos aprender;

O pré-requisito para usar SQL em um banco de dados, além de dominar a linguagem, é conhecer ou ter acesso aos metadados do banco para que o programador saiba os nomes corretos das entidades, atributos, chaves primárias e estrangeiras etc. Fica evidente, que toda a documentação que aprendemos até aqui devem estar ao alcance da equipe de desenvolvimento.

Nota: Chamamos de query o comando SQL que submetemos ao banco de dados para ser executado.

Considerações Gerais

A Linguagem SQL não é sensível ao caso, isto é, não importa se os nomes das entidades, atributos, comando etc. estão grafados em maiúsculas ou minúsculas. É uma boa prática escrever os comandos em letras maiúsculas.

Exemplo: SELECT, INSERT, UPDATE, FROM, DELETE, WHERE etc.

Em SQL o texto é escrito entre aspas simples, portanto, caso no texto que for inserido exista uma aspa simples, duplique a aspa simples para que não retorne um erro sintático na query.

Exemplo Problema:

```
UPDATE TAB_Aluno set nome='Jose D'Silva' WHERE cod_aluno='541234'.
```

Exemplo Solução:

```
UPDATE TAB_Aluno set nome='Jose D''Silva' WHERE cod_aluno='541234'
```

Cuidado quando sua query for inserir ou atualizar números. Para o banco, a vírgula decimal (em português usamos vírgula e não ponto) não é considerada separador decimal e sim separador de milhares. Isso pode confundir e trazer resultados errados em suas consultas. Portanto utilize apenas um ponto para separador decimal e retire as vírgulas do número antes de submeter (rodar) a query;

Exemplo Problema:

```
UPDATE TAB_Curso set cargaHoraria = '12,5' Where codigoCurso=1000;
```

Exemplo Solução:

```
UPDATE TAB_Curso set cargaHoraria = '12.5' Where codigoCurso=1000;
```

Trabalhar com datas também pode ser um problema. O formato da data pode estar pré-configurado para o formato data brasileiro, ou não. Uma forma de ficar independente do formato de datas do banco é utilizar o formato: 'YYYY-MM-DD'. Lembre-se de escrever as datas sempre entre aspas simples;

Exemplo:

```
SELECT nome FROM TAB_Aluno WHERE data_nascimento = '2001-01-23'
```

Naturalmente, se você estiver certo de que o banco está configurado para data BR, você pode usar a data da forma que estamos acostumados e ignorar essa dica:

Exemplo:

```
SELECT nome FROM TAB_Aluno WHERE data_nascimento = '23/01/2001'
```

Mais um alerta sobre datas: Se o tipo de data for datetime (tipo que armazena data e hora), isso quer dizer que o banco irá armazenar as datas no formato ano-mês-dia-hora-minuto-segundo, ou seja, se eu pegar a data do servidor neste exato momento (que estou digitando isso aqui) vou obter: 2022-01-15 16:22:15.

O que você espera de resultado com esta query se o campo data_nota for um campo datetime?

```
SELECT nota_Fiscal FROM TAB_Notas WHERE data_nota = '2022-01-15'
```

Os comandos SQL**16.1. O SELECT**

É a instrução que permite a consulta direta aos dados que estão armazenados no banco.

Sintaxe:

```
SELECT atributo1 [,atributo2, ... , atributoN] FROM entidade [WHERE chave=valor]
```

Exemplos:

Assim as seguintes queries são válidas como **SELECT**:

```
SELECT Codigo_Aluno FROM TAB_Aluno
```

Retorna todos os códigos dos alunos cadastrados

```
SELECT Nome_Aluno FROM TAB_Aluno WHERE Codigo_Aluno=541234;
```

Retorna o nome do aluno cujo código é '541234'

```
SELECT * FROM TAB_Aluno
```

Retorna todas as linhas e todas as colunas da tabela.

Este tipo de consulta não é recomendado porque normalmente não precisamos o retorno de toda a tabela. O Banco levará mais tempo e ocupará mais memória do que se a consulta estivesse mais bem filtrada. Por isso se recomenda que a query retorne apenas as linhas e colunas que são uteis no momento da consulta.

SELECT - Exemplos práticos:

Todas as consultas foram realizadas sobre a tabela TAB_Aluno abaixo:

TAB_Aluno

	Matricula_Aluno	Nome	Data_Nascimento	Curso	Codigo_Turma
1	1000	Diego	1995-01-07	Fundamental 2	4000
2	2000	Carlos	2000-02-12	Fundamental 2	4000
3	3000	José	2010-12-23	Fundamental 2	5500
4	4000	Manoel	2002-07-21	Fundamental 2	5000

```
select * FROM TAB_Aluno where Data_Nascimento > '2000-01-01'
```

0 %

Resultados Mensagens

	Matricula_Aluno	Nome	Data_Nascimento	Curso	Codigo_Turma
1	2000	Carlos	2000-02-12	Fundamental 2	4000
2	3000	José	2010-12-23	Fundamental 2	5500
3	4000	Manoel	2002-07-21	Fundamental 2	5000

```
select Nome FROM TAB_Aluno where Codigo_Turma = 4000
```

150 %

Resultados Mensagens

	Nome
1	Diego
2	Carlos

```
select Nome, Data_Nascimento FROM TAB_Aluno where codigo_Turma >=5000
```

%

Resultados Mensagens

	Nome	Data_Nascimento
	José	2010-12-23
	Manoel	2002-07-21

16.3.2. O UPDATE

É a instrução que permite atualizar/modificar os dados que estão armazenados no banco. É necessário definir que tabela, que atributo e que linhas devem ser modificadas.

Sintaxe:

UPDATE entidade SET cam1=val[,cam2=val, ..., camN=val] [WHERE chave=val]

Assim as seguintes queries são válidas como UPDATE:

UPDATE TAB_Aluno set nome='Diego Augusto' WHERE Matricula_Aluno='1000'

Troca o nome do aluno código de matrícula 1000

UPDATE TAB_Aluno set nome_curso='fundamental 1' WHERE Codigo_Turma=4000;

Troca os dois alunos, Diego e Carlos, de Fundamenta2 para Fundamental1

UPDATE TAB_Aluno set Codigo_Turma = '5000'

Troca TODAS os codigos_turma de todos os alunos para 5000

Ops!!! Update sem where... Deu ruim!

UPDATES – Exemplos práticos

UPDATE TAB_Aluno set nome='Diego Augusto' WHERE Matricula_Aluno='1000'

Matricula_Aluno	Nome	Data_Nascimento	Curso	Codigo_Turma
1000	Diego Augusto	1995-01-07	Fundamental 2	4000
2000	Carlos	2000-02-12	Fundamental 2	4000
3000	José	2010-12-23	Fundamental 2	5500
4000	Manoel	2002-07-21	Fundamental 2	5000

UPDATE TAB_Aluno set curso='fundamental 1' WHERE Codigo_Turma=4000;

Matricula_Aluno	Nome	Data_Nascimento	Curso	Codigo_Turma
1000	Diego Augusto	1995-01-07	fundamental 1	4000
2000	Carlos	2000-02-12	fundamental 1	4000
3000	José	2010-12-23	Fundamental 2	5500
4000	Manoel	2002-07-21	Fundamental 2	5000

Este último exemplo na realidade é uma armadilha tipo pesadelo se for realizado em uma base de dados de produção. Com um update deste (*update sem where*), podemos

```
UPDATE TAB_Aluno setCodigo_Turma = '5000'
```

trocar

uma coluna inteira de uma tabela por um mesmo valor (no caso, todos os códigos de turma ficarão iguais a 5000). As vezes pode ser isso mesmo que queremos, se estiver inicializando uma coluna por exemplo, mas o mais provável é que executamos este comando por engano, simplesmente esquecendo de definir que linhas deveriam ser afetadas. Se não tiver backup atualizado... Já era.



16.2. O INSERT

Esta instrução permite inserir novos dados no banco de dados.

Sintaxe:

```
INSERT INTO entidade (nomeCampo1[,nomeCampo2,...,nomeCampoN) VALUES  
(valor1[,valor2, ..., valor)
```

Exemplos:

```
INSERT INTO TAB_Aluno(matricula_aluno,nome, data_nascimento, curso,  
codigo_turma) VALUES
```

```
( '5000', 'Ricardo', '1991-05-01', 'Direito','8000');
```

```
INSERT INTO TAB_Turma (codigo_turma,Nome_Turma) VALUES (8000,'Direito');
```

```
INSERT INTO TAB_Aluno VALUES ('6000','Indeciso','2000-01-01','Direito',NULL)
```

Este último é uma forma alternativa de INSERT. Note que foi suprimida a lista de campos. Isso porque existe um item conteúdo para cada um dos atributos da tabela. Neste caso o SGBD coloca os campos na tabela na ordem que eles aparecem na query

TAB_Turma			TAB_Aluno				
	Codigo_Turma	Nome_Turma	Matricula_Aluno	Nome	Data_Nascimento	Curso	Codigo_Turma
1	4000	Quinta Série A	1000	Diego Augusto	1995-01-07	fundamental 1	4000
2	5000	Quinta Série B	2000	Carlos	2000-02-12	fundamental 1	4000
3	5500	Sexta série A	3000	José	2010-12-23	Fundamental 2	5500
4	6000	Sexta série B	4000	Manoel	2002-07-21	Fundamental 2	5000
5	8000	Direito	5000	Ricardo	1991-05-01	Direito	8000
			6000	Indeciso	2000-01-01	Direito	NULL

16.3. Fazendo uma consulta com mais de uma tabela

É bem simples fazer consultas de tabelas distintas ligadas por suas chaves primárias e estrangeiras. Precisamos identificar quem é quem em cada uma das tabelas, igualar os campos PK da tabela 1 = FK da tabela 2 e está feito

Exemplo:

SELECT

Matricula_Aluno, Nome, Data_Nascimento, TAB_Turma.Codigo_Turma, nome_Turma

FROM

TAB_Aluno, TAB_Turma

WHERE

TAB_Turma.Codigo_Turma = TAB_Aluno.Codigo_Turma

Matricula_Aluno	Nome	Data_Nascimento	Codigo_Turma	nome_Turma
1000	Diego Augusto	1995-01-07	4000	Quinta Série A
2000	Carlos	2000-02-12	4000	Quinta Série A
3000	José	2010-12-23	5500	Sexta série A
4000	Manoel	2002-07-21	5000	Quinta Série B
5000	Ricardo	1991-05-01	8000	Direito

Podemos filtrar melhor nossa query, mostrando apenas os alunos da quinta série

A.

SELECT

Matricula_Aluno, Nome, Data_Nascimento, TAB_Turma.Codigo_Turma, nome_Turma

FROM

TAB_Aluno, TAB_Turma

WHERE

TAB_Turma.Codigo_Turma = TAB_Aluno.Codigo_Turma AND
TAB_Turma.codigo_Turma=4000

Matricula_Aluno	Nome	Data_Nascimento	Codigo_Turma	nome_Turma
1000	Diego Augusto	1995-01-07	4000	Quinta Série A
2000	Carlos	2000-02-12	4000	Quinta Série A



Importante

Quando relacionamos tabelas que o nome do atributo na tabela 1 é idêntico ao nome do atributo da tabela 2 (Codigo_Turma por exemplo), e isso é bastante comum, colocamos o nome da tabela seguido de um ponto, seguido do nome do atributo. Isto é necessário para que o SGBD saiba qual dos atributos está sendo referenciado.

Podemos também criar apelidos para as tabelas de forma que os relacionamentos e os filtros (filtros=where) fiquem mais enxutos, menos poluídos. Vamos repetir esta última query, criando apelidos para as tabelas:

SELECT

Matricula_Aluno, Nome, Data_Nascimento, T.Codigo_Turma, nome_Turma

FROM

TAB_Aluno as A, TAB_Turma as T

WHERE

T.Codigo_Turma = A.Codigo_Turma AND T.codigo_Turma=4000

A tabela TAB_Aluno ganhou o apelido de **A** e a tabela TAB_Turma ganhou o apelido de **T**. O resultado é exatamente o mesmo.

Matricula_Aluno	Nome	Data_Nascimento	Codigo_Turma	nome_Turma
1000	Diego Augusto	1995-01-07	4000	Quinta Série A
2000	Carlos	2000-02-12	4000	Quinta Série A

Alguns bancos o 'as' entre o nome da tabela e o apelido é opcional:

Em vez de FROM TAB_Aluno **as** A, pode ser FROM TAB_Aluno A

16.4. Fazendo consulta de mais de uma tabela utilizando INNER JOIN

Nos exemplos anteriores relacionamos diretamente as chaves da tabela TAB_Aluno com a TAB_Curso igualando TAB_Aluno.Codigo_Turma com TAB_Curso.Codigo_Turma. Essa forma de relacionar funciona bem, mas se for o caso de mostrar algum aluno que

ainda não está matriculado em nenhum curso, esta *query* não vai trazer porque ela exige que exista um valor de `Codigo_Turma` na coluna de turma, e não é o caso.

Exemplo:

SELECT

`Matricula_Aluno, Nome, Data_Nascimento, T.Codigo_Turma, nome_Turma`

FROM

`TAB_Aluno as A`

INNER JOIN `TAB_Turma AS T` **ON** `A.Codigo_Turma = T.Codigo_Turma`

Retorna todos os alunos exceto o que não se está em nenhuma turma. Isso porque usamos **INNER JOIN** que equivale ao relacionamento direto utilizando os campos chave.

Vamos tentar novamente utilizando o **LEFT JOIN**

Matricula_Aluno	Nome	Data_Nascimento	Codigo_Turma	nome_Turma	curso
1000	Diego Augusto	1995-01-07	4000	Quinta Série A	fundamental 1
2000	Carlos	2000-02-12	4000	Quinta Série A	fundamental 1
3000	José	2010-12-23	5500	Sexta série A	Fundamental 2
4000	Manoel	2002-07-21	5000	Quinta Série B	Fundamental 2
5000	Ricardo	1991-05-01	8000	Direito	Direito

16.5. Fazendo consulta de mais de uma tabela utilizando LEFT JOIN

Exemplo:

SELECT

`Matricula_Aluno, Nome, Data_Nascimento, T.Codigo_Turma, nome_Turma, curso`

FROM

`TAB_Aluno as A`

LEFT JOIN `TAB_Turma as T` **ON** `A.Codigo_Turma = T.Codigo_Turma`

Agora a consulta retorna todos os alunos inclusive o aluno que não está inscrito em nenhuma turma.

Matricula_Aluno	Nome	Data_Nascimento	Codigo_Turma	nome_Turma	curso
1000	Diego Augusto	1995-01-07	4000	Quinta Série A	fundamental 1
2000	Carlos	2000-02-12	4000	Quinta Série A	fundamental 1
3000	José	2010-12-23	5500	Sexta série A	Fundamental 2
4000	Manoel	2002-07-21	5000	Quinta Série B	Fundamental 2
5000	Ricardo	1991-05-01	8000	Direito	Direito
6000	Indeciso	2000-01-01	NULL	NULL	NULL



Saiba mais

Além de INNER JOIN e LEFT JOIN existem outras formas de conectar tabelas. Pesquise e tente testar todas elas para compreender bem a diferença entre elas.

Pesquise e saiba mais sobre SQL... comandos DML, DML e DCL.

17. ADMINISTRADOR DE DADOS. O DBA

Objetivo:

Conhecer atribuições e responsabilidades dos profissionais administradores de dados.

Introdução:

O objetivo principal do DBA é garantir a existência de uma infraestrutura básica de dados cujo formato e conteúdo atenda às necessidades com relação a informações, além de garantir o acesso a bases de dados integras consistentes e seguras. O DBA deverá ter duas funções básicas: Administração de Dados e Administração de Banco de Dados, que executam atividades distintas, mas complementares e que contribuem para o cumprimento dos objetivos da área.

Responsabilidades da Administração de Dados

Compreendem basicamente a construção de modelos de dados a partir do levantamento das necessidades do negócio com relação a informações e a interação com os Administradores de Banco de Dados no sentido de implementar fisicamente as informações modeladas.

Responsabilidades:

- a. Avaliar e definir as ferramentas a serem utilizadas no suporte às atividades de administração de dados;
- b. Construir, manter e validar os modelos de dados garantindo que estes atendem aos requisitos do negócio e são viáveis de implementação física;
- c. Fazer uso efetivo da ferramenta CASE, explorando todas as suas potencialidades, interagindo com os desenvolvedores de aplicações;
- d. Definir procedimentos de segurança de acesso aos dados;
- e. Levantar as necessidades de distribuição de dados e atuar junto com os DBA's na implementação física;
- f. Redefinir os modelos de dados em função de particularidades na implementação física a fim de melhorar a performance do banco de dados;
- g. Dar suporte aos usuários finais no acesso aos dados.

Responsabilidades Administração de Banco de Dados

A Administração de Banco de Dados compreende a implementação física das bases de dados a partir dos modelos gerados pelos ADs e a administração do ambiente de bancos de dados no sentido de garantir a disponibilidade, integridade, consistência e segurança das informações; além de assegurar que os acessos aos dados sejam executados com o nível de performance requerido pelas áreas de negócio.

Responsabilidades:

- a. Produzir o detalhamento técnico da implementação das bases de dados.
- b. Implementar e manter os bancos de dados nos ambientes de teste, homologação e produção, assegurando sua disponibilidade, consistência, integridade e devido detalhamento técnico;
- c. Instalar e manter os Sistemas Gerenciadores de Banco de Dados, explorando toda a sua potencialidade;
- d. Realizar testes e monitoramento de performance nos bancos de dados, ajustando os ambientes quando necessário, bem como efetuando planejamento de capacidade;
- e. Definir e testar plano de contingência relativo aos SGDB's, e seus procedimentos de *backup* e *restore*;
- f. Definir e implementar padrões relativos à administração de banco de dados.
- g. Definir e implementar o modelo físico da base de dados a partir das informações passadas pela administração;
- h. Dar suporte usuários com relação à melhor utilização da linguagem SQL para acesso à base de dados, sugerindo mudanças e adequando a implementação física dos bancos de dados quando necessário.



Referências

HEUSER, C. A. PROJETO DE BANCO DE DADOS; PORTO ALEGRE: BOOKMAN, 2008.

COUGO, PAULO SERGIO. MODELAGEM CONCEITUAL E PROJETO DE BANCO DE DADOS; RIO DE JANEIRO: CAMPUS, 1997.

FANDERUFF, DAMARIS

ORACLE 8I, SQL*PLUS E PL/SQL: SÃO PAULO, MAKROM BOOKS, 2000

DATE, C. J., INTRODUÇÃO A SISTEMA DE BANCO DE DADOS; ELSEVIER; RJ, 2004.

NAVATHE, SHAMKANT B / ELMASRI, RAMEZ E., SISTEMAS DE BANCO DE DADOS, ADDISON WESLEY BRA, SP, 2005.

HEUSER, C. ALBERTO. PROJETO DE BANCO DE DADOS. 2009