

Banco de Dados aplicados a Jogos

08-2018

Professor: EDIVAL CARVALHO ALUNO:

OBJETIVO GERAL

Desenvolver no aluno as capacidades de modelar e construir uma base de dados que armazene informações sobre um jogo e configurar conexões de banco de dados em um projeto de jogo.

EMENTA

- Modelagem e abstração de dados.
- Atributo, Entidade e Relacionamento. Construção de modelos relacionais.
- Criação de estruturas de dados.
- Consulta e manipulação de dados: SQL.
- Armazenamento e recuperação de informações dos jogadores e do jogo em bases de dados. Controle de acesso.
- Conexão do jogo com bancos de dados.

BIOGRAFIA BÁSICA

- SILBERSCHATZ, Abraham. Sistema De Banco De Dados. Editora: Campus. 6^a EDIÇÃO. 2012.
- MANZANO, José Augusto N. G. MySQL 5.5 - Interativo - Guia Essencial de Orientação e Desenvolvimento.
- Editora: ERICA. 2011.
- MANNINO, Michael V.. Projeto, Desenvolvimento de Aplicações e Administração de Banco de Dados.
- Editora: MCGRAW-HILL - BRASIL. 2008.

Prof. Edival Carvalho

INDICE

Introdução ao Banco de Dados	6
Conceito de Banco de Dados	6
Sistemas gerenciadores de banco de dados	7
Características de Banco de Dados	10
Natureza Autodescritiva	11
Isolamento e Abstração de Dados.....	12
Suporte para múltiplas visões	12
Compartilhamento de dados e processamento de transações.....	13
<i>OLTP – On-Line Transaction Processing</i>	13
Atores	13
Vantagens de usar a abordagem de SGBD	14
Arquitetura do Sistema de BD.....	16
Arquitetura de Banco de Dados em Jogos	16
Persistência salvar / carregar:.....	17
Base de dados de Tipos:.....	17
Base de dados de Modelos:	17
O Modelo Relacional.....	18
Modelagem de Banco de Dados relacionais	22
As Reuniões de Modelagem	22
Os casos de excessão	23
O processo de modelagem de dados	23
Pré-validando cada criação de entidade	24
Validando cada evolução do modelo	26
Surrogate Key	27
Vantagens da Surrogate Key:.....	28
Desvantagens da Surrogate Key.....	29
A Dimensão tempo	30
Entidades fortes e Entidades fracas	31
Valores Nulos	32
Regras do Mundo Real x Regras do Modelo de dados	33
Atualizações das operações do mundo real para o banco de Dados	36
SQL (<i>Structured Query Language</i>).....	39
História do SQL	39
Escopo do SQL	40
SQL – DDL - Definição de Dados.....	40
Restrições de Integridade	42
CHAVE PRIMÁRIA - <i>CONSTRAINT PRIMARY KEY</i>	43
RESTRIÇÕES DE CHAVES ESTRANGEIRAS (<i>CONSTRAINT FOREIGN KEY</i>)	44
SQL – DML	45
Comando SELECT	45
Comando INSERT	46
Comando UPDATE	49
Comando DELETE	50
Normalização de Bancos de Dados Relacionais	53
FORMAIS NORMAIS	54
1FN - 1 ^a Forma Normal:	55
2FN - 2 ^a Forma Normal:	55
3FN - 3 ^a Forma Normal:	56
Normalização para a Primeira Forma Normal	57
Normalização para a Segunda Forma Normal	59
Normalização para a Terceira Forma Normal.....	60
Game Engines e Banco de Dados Jogos.....	62
Unity	62
Unity - XML	62
Unity - SQLite.....	62
Referências	63

Prof. Edival Carvalho

Prof. Edival Carvalho

Introdução ao Banco de Dados

Conceito de Banco de Dados

Bancos de dados estão presentes no cotidiano, são utilizados em diversas áreas, desde simples cadastros, a lojas online e instituições financeiras.

Muitas vezes o nome “banco de dados” é utilizado de forma genérica, sem referência ao verdadeiro significado.

Mas o que é um banco de dados?

Segundo NAVATHE e ELMASRI: “*Um banco de dados é uma coleção de dados relacionados.*”.

Segundo dicionário Michaelis, dados são definidos como:

“*Conjunto de material (= informações) disponível para análise. 2 Representação de fatos, conceitos e instruções, por meio de sinais de uma maneira formalizada, possível de ser transmitida ou processada pelo homem ou por máquinas. Os dados são sistemáticos, tem o significado implícito, ou seja, o próprio dado identifica seu tipo, origem e características.*

Um exemplo é uma agenda telefônica, seja de papel, ou no celular ou até mesmo em um arquivo do Excel. As informações de telefone, email, endereço entre outros são dados das pessoas, o que caracteriza como um banco de dados.

Apesar dessa definição genérica de banco de dados, não podemos considerar qualquer coleção de dados como um banco de dados.

Para a definição existem alguns critérios que devem ser atendidos:

- Banco de dados representam aspectos do mundo real, também chamados de minimundo ou de universo de discurso;
- Banco de dados é uma coleção lógica e coerente de dados com significado;
- O banco de dados possui um grupo de usuários definidos e uma ou mais aplicações de acordo com o interesse do grupo.
- Um banco de dados é projetado e construído a fim de atender um objetivo;

Uma coleção de dados randômicos não pode ser considerada como banco de dados;

Bancos de dados podem ser criados de forma física e controlados manualmente, mas também podem ser criados de forma computacional, virtualmente, controlados por um sistema específico.

Um banco de dados não necessita ser computadorizado, por exemplo: o controle de fichas de uma biblioteca é um banco de dados manual.

Sistemas gerenciadores de banco de dados

Os sistemas específicos utilizados na manutenção de banco de dados são chamados de SGBD (Sistemas gerenciadores de banco de dados).

Este software é utilizado para os processos de definição, construção, manipulação e compartilhamento de banco de dados entre várias aplicações e usuários.

O processo de **definição de banco de dados** é a ação de especificar os tipos de dados, as estruturas, as restrições e os relacionamentos dos dados armazenados.

O processo de **construção do banco de dados** é a ação de armazenar em mídia (arquivos) controlada pelo SGBD.

Os processos de **manipulação** são as ações de consulta, recuperação de dados, inserção de dados, atualização de dados e geração de relatórios.

O **compartilhamento** permite o acesso aos dados por múltiplos usuários e aplicações.

Outros processos podem ser implementados pelo SGBD, que são a **proteção e manutenção**.

O acesso ao banco de dados deve ser controlado, protegendo o mesmo de acessos indevidos, além de garantir a integridade e a segurança contra falhas, estes são exemplos de processos de proteção.

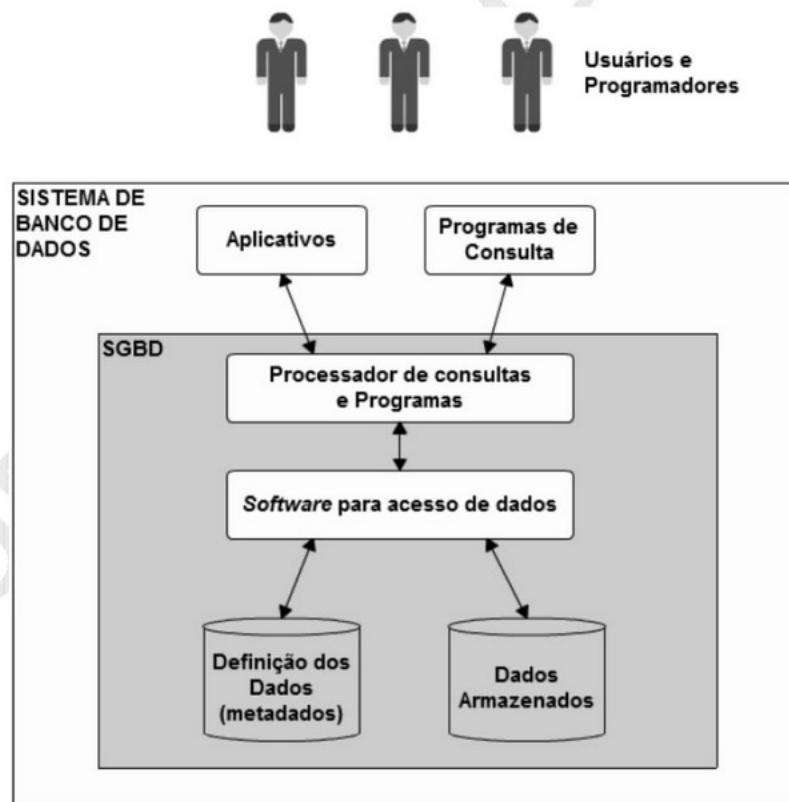


Figura 01: Componentes de um SGBD.

Um banco de dados pode durar por muitos anos, dessa forma o SGBD precisa de manutenção, para alterar os requisitos que passam por alterações ao longo do tempo.

A criação de um banco de dados computadorizado não requer que seja utilizado um SGBD típico, pois é possível criar funções próprias no programa para fazer, este caso pode ser conhecido como SGBD com uma finalidade específica. O problema é o trabalho necessário para se desenvolver estas funções específicas, já disponíveis na maioria dos softwares de banco de dados.

Exemplo 1: Um exemplo simples e que todos estão familiarizados é o banco de dados de uma faculdade.

A tabela Alunos armazena os dados de cada estudante.

IdAluno	AlNome	AlEndereço	AlTelefone
15	José Correa de Figueiras Neto	Rua do Carmo 32	21938238943
17	Maria de Figueiras Neto	Rua do Carmo 32	2198472347
22	Isabela do Nascimento	Av Chile 2001	324767324
23	Thais de Almeida Braga	Rua do Limoeiro 21	2398472323
25	Paulo Jose Figueiras	Quadra Norte 301 Bl A ap 402	723612361232
26	André Sergio Falabella Faria	Av Barata Ribeiro 821 ap 108	7821386123

Tabela 01: Tabelas Alunos.

A tabela Curso armazena os dados dos cursos da instituição de ensino.

IdCurso	CrNome	CrCargaHoraria
01	Jogos Digitais	1200
02	Engenharia de Sistemas	1200
03	Bacharelado em Matematica	1200

Tabela 02: Tabelas Cursos.

A tabela Matricula armazena os dados referentes ao fato de um aluno cursar (ou haver cursado) um determinado curso da instituição de ensino.

IdMatricula	MTStatus	IdAluno	IdCurso
0000022	A	15	01
0000016	T	15	02
0000017	A	22	02
0000018	A	23	02

0000019	A	25	01
0000020	T	26	03
0000021	A	17	01

Tabela 03: Tabela Matricula.

A tabela Inscrição_Disciplina armazena os dados das notas de cada aluno nas disciplinas cursadas.

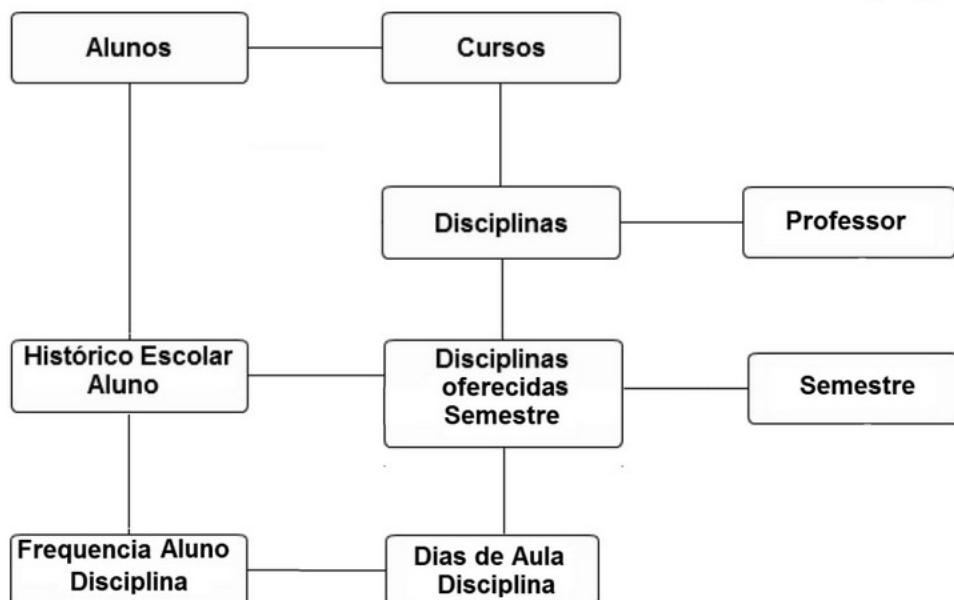


Figura 02: Diagrama do banco de dados Faculdade.

Exemplo 2 (Jogos)

Outro exemplo simples e de jogos que todos estão familiarizados é o banco de dados de um ranking de jogos.

A tabela Jogadores armazena os dados de cada jogador.

Tabela 04: Tabela Jogadores.

IDJogador	JNome	JEmail	JSenha	JDSaldo
15	Mario Pinheiro	mariopinheiro@gtnet.com	*****	R\$ 164,90
8	Luiz Feitosa Dias	luizfdias@zol.com	*****	R\$ 89,31
18	Karina Bittencourt Pinho	Karina.bittencurt@aol.com	*****	R\$ 112,87

A tabela Jogos armazena os dados de cada jogo.

Tabela 05: Tabela Jogos.

IDJogo	JGNome	JGServidor	JGTipo	JGValor	JGStatus
2	Tetris	052.786.987.001	1-Individual	R\$ 164,90	O-Online
3	Metal Slug	052.786.988.004	1-Individual	R\$ 89,31	O-Online
4	War Digital	052.786.967.006	2-Grupo	R\$ 112,87	D-Manutenção

A tabela Jogadas armazena os dados da jogada para cada jogador e jogo.

Tabela 06: Tabela Jogadas.

IdJogada	IdJogo	IdJogador	JGTempo	JGPontos
188	1	15	368	1027
189	1	9	325	1188
190	2	9	879	1700

Exemplo 2 (Jogos)

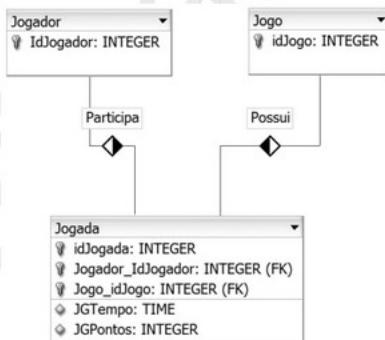


Figura 03: Diagrama do banco de dados Ranking de Jogos.

Características de Banco de Dados

Algumas características diferem a abordagem do uso de banco de dados da programação e arquivos. No processamento de arquivos, cada usuário define seus arquivos e é implementado no próprio programa. O exemplo da faculdade pode ajudar a entender melhor a diferença.

A secretaria de Notas utiliza uma lista dos alunos e um arquivo com as notas, onde pode ser gerada uma lista de histórico escolar, enquanto o departamento de Contabilidade utiliza uma outra lista de alunos para controlar as mensalidade e os pagamentos. Neste caso, ambos setores estão interessados nos dados dos alunos, mas cada um mantém as informações em arquivos separados. Os programas também manipulam os dados separados, isso porque, cada setor

precisa de dados que não estão disponíveis um para o outro, no caso as notas e as mensalidades. Esses dados redundantes de alunos resultam em uma maior utilização de espaço de armazenamento e um esforço muito maior para manter os dados atualizados. Pois mudanças que impactam um setor, por exemplo, o de Notas, podem impactar outro setor, por exemplo, o de Contabilidade.,

Quando Banco de Dados é utilizado, um único repositório é criado e mantido, sendo acessado por todos os usuários. Todos os usuários tem acesso às mesmas informações.

- As principais características são de um banco de dados são:

- Natureza autodescritiva;
- Isolamento entre os programas e dados;
- Suporte para múltiplas visões dos dados;
- Compartilhamento de dados e processamento de transações;

Natureza Autodescritiva

Umas das principais características em um Sistema de Banco de Dados é que ele não possui apenas o banco de dados, mas também uma descrição completa da estrutura desse banco de dados. Esta descrição é armazenada no catálogo do SGBD, contem informações de cada tabela, os tipos de dados e até restrições. Este catálogo é chamado de **metadados**. O catálogo é utilizando tanto pelos usuários para consultar a estrutura da base de dados quanto pelo SGBD.

O SGBD não é criado apenas para um tipo específico de aplicação, por isso existe o catálogo, para definir os tipos e a estrutura dos dados. Isso permite que possa trabalhar com diferentes bancos de dados, desde que uma base para faculdade quando para um ranking de jogo. No processamento de arquivos, a definição dos dados faz parte do próprio programa, dessa forma as aplicações ficam restritas a trabalhar apenas com aquele formato específico e aquela mesma definição de dados.

Um exemplo para essas definições são *struct* ou classes criadas em um programa C++.

No Exemplo 1 apresentado anteriormente, todas as definições de cada tipo, tamanho, e o desenho próprio das tabelas quando utilizando um SGBD ficam no catálogo, que seria utilizado para os acessos.

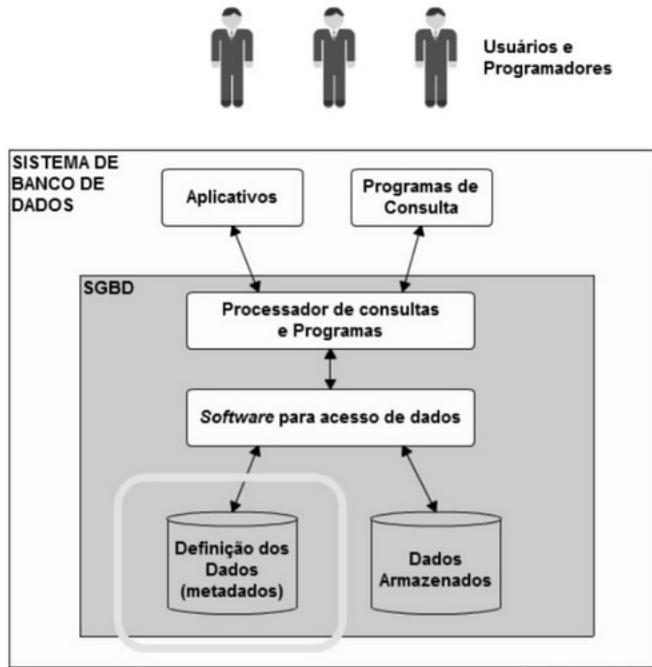


Figura 04: Natureza Autodescritiva - Metadados

Isolamento e Abstração de Dados

Enquanto em processamento de arquivos, estas definições ficam diretamente no código do programa, acessando diretamente os dados nos arquivos. Quando é utilizado processamento de arquivos, qualquer alteração na estrutura das tabelas cria a necessidade de se alterar todos os programas para se adaptarem as mudanças.

Porém quando é utilizado SGBD não exigem alterações na maioria dos casos e também dependendo de que forma os programas foram implementados. Esta propriedade é chamada independência da operação do programa.

O SGBD oferece uma representação conceitual de dados, porém não informa detalhes de como são armazenados ou as operações implementadas. Este pode ser considerado um tipo de abstração, uma vez que o modelo apresenta em forma de objetos, propriedades e relacionamentos.

Suporte para múltiplas visões

Banco de dados possui muitos usuários, e cada um possui suas próprias necessidades e exige diferentes visões para os dados.

Uma visão é um subconjunto do banco de dados que pode conter dados virtuais ou dados derivados das tabelas. As visões ajudam a criar diferentes formas de acessos aos dados com diferentes objetivos. Inclusive a proteção dos dados confidenciais.

Compartilhamento de dados e processamento de transações

SGBD multiusuário permite o acesso simultâneo por um ou mais usuários ao dados (sem que um acesso interfira nos outros acessos que estão sendo processados simultaneamente). É essencial que um banco de dados seja acessado por múltiplos usuários, especialmente quando os dados são utilizados por várias aplicações.

O SGBD possui mecanismos para controlar e garantir que alterações nos dados por vários usuários não tenham conflito.

OLTP – On-Line Transaction Processing.

Um sistema de informação que processa um volume de alterações de dados por vários usuários simultaneamente geralmente é chamado de sistema de processamento de transação *on-line*.

O SGBD possui sofisticados mecanismos para fazer o controle dessas transações, garantindo que as alterações operem de forma correta. Uma transação é um programa em execução ou um processo que possui um ou mais acessos ao banco de dados. Essas transações podem executar consultas ou atualizações nos dados. O controle dessas transações segue por meio de um nível de isolamento, definido pela execução, que garante que muitas transações executando simultaneamente.

Atores

Em pequenos bancos de dados pessoais, o próprio usuário define, desenha, constrói e manipula a base de dados. Porém em grandes organizações, muitas pessoas trabalham no projeto, no uso e na manutenção do banco de dados.

- Administradores de banco de dados
- Projetistas de banco de dados
- Usuários finais
- Engenheiros de software

O administrador de banco de dados ou (*DBA – Database administrator*) é responsável pelo recurso principal, que é o próprio banco de dados, seguindo pelo SGBD e os softwares relacionados. As principais atividades do DBA são autorizar o acesso aos dados, coordenar e monitorar o uso, adquirir novos software ou hardware conforme necessidade. Também são responsáveis por atualização de software, trabalhar em problemas, como falhas de segurança ou de performance. Geralmente é formado por uma equipe que trabalha nessas funções.

Projetistas de banco de dados - Os projetistas tem a responsabilidade de identificar os dados que serão armazenados, e definir as estruturas de armazenamento. Este trabalho acontece antes mesmo da criação do banco de dados e os dados serem carregados. Normalmente os projetistas

estão na equipe de DBAs. Projetistas também tem atividade após a criação do banco, o trabalho é criar novas visões para atender aos usuários finais.

Usuários finais - São as pessoas que executam o acesso ao banco de dados para consultas, atualização e geração de relatórios.

Engenheiros de software – São os profissionais que identificam as necessidades dos clientes e definem como o software deverá ser construído para atender estas necessidades. Para executar essa tarefa,, os engenheiros de software necessitam de um conhecimento abrangente das capacidades do SGBD para realizar as tarefas.

Os programadores – São os especialistas nas linguagens de desenvolvimento de sistemas, os quais, implementam essas especificações em aplicações.

Vantagens de usar a abordagem de SGBD

A escolha de um bom SGBD é essencial para o desenvolvimento de um bom projeto pois auxilia na administração de inconsistências e uso de grandes BD multiusuários. O SGBD auxilia com a restrição de acesso de determinados usuários/grupo de usuários em tabelas específicas – O tipo de operação de acesso também pode ser controlado: recuperação ou atualização.

O SGBD oferece estruturas de armazenamentos e técnicas de consultas

- Executa consultas (com auxílio de índices que facilitam o encontro de resultados);
- Atualiza dados de forma eficiente;
- O SGBD oferece backup e recuperação em caso de falhas de hardware e software.
- O SGBD oferece múltiplas interfaces para usuários, alguns semelhantes à formulários.
- O SGBD oferece a representação de relacionamentos complexos entre dados.

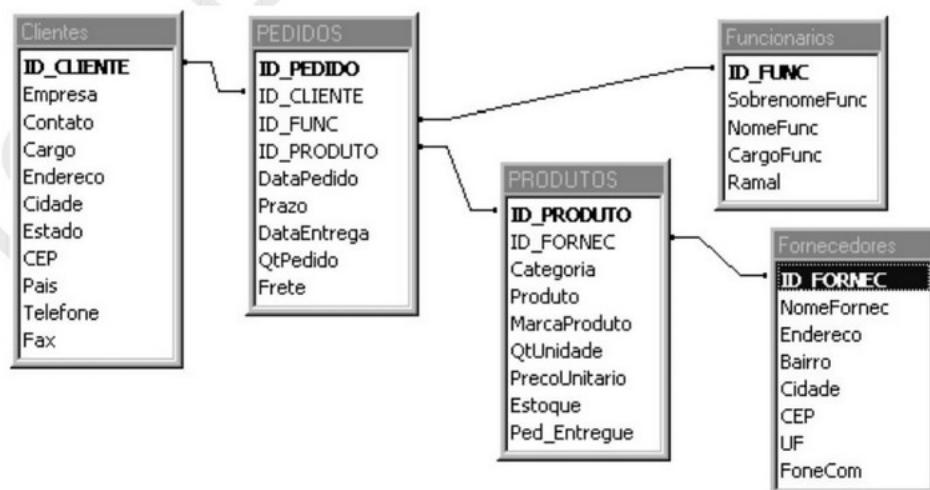


Figura 05: Representação de relacionamentos complexos entre dados.

O SGBR oferece a associação de gatilhos (ou *triggers*) que são considerados regras que são acionadas em determinado momento da atualização de um tabela, além disso, o SGBD conta com procedimentos armazenados (ou *stored procedures*).

Prof. Edival Carvalho

Arquitetura do Sistema de BD

Uma das principais características de Banco de dados é devido a sua abstração, desta forma, não existe a necessidade do detalhamento em relação ao armazenamento dos dados. Baseado nas características de um banco de dados, podemos definir sua arquitetura como de três esquemas. Nem todos os SGBD executam a separação nos três níveis, mas permite suporte a elas:

- **Nível interno:** Possui um esquema interno que descreve de que forma a estrutura do banco de dados é armazenado fisicamente na estrutura de arquivos.
- **Nível conceitual:** Este esquema não tem definição física, o foco da estrutura é descrever as entidades, relacionamentos, restrições e operações.
- **Nível externo:** Este modelo atende um grupo específico de usuários sendo possível ocultar uma parte do banco de dados, estes esquemas geralmente são exibidos em modelos de dados representativos.

Este modelo de arquitetura permite que os bancos de dados possuam **independência de dados**. Por meio da independência lógica de dados permite alterar o esquema conceitual sem ter que modificar os esquemas externos ou as aplicações. Pode-se ser criar novas restrições ou mesmo a alteração do desenho destas tabelas, porém para garantir a independência, as aplicações precisam trabalhar com visões de dados.

A independência física de dados permite alterar o esquema interno do banco sem que exista alteração para o esquema conceitual. Algumas modificações nos esquemas físicos podem garantir um melhor desempenho se, por exemplo, um novo grupo de arquivos físicos for criado ou até mesmo uma reorganização dos mesmos.

A arquitetura básica de um SGBD cliente/servidor é distribuída em dois módulos:

- **Módulo cliente:** instalado na estação de trabalho ou em um computador pessoal, este módulo oferece interfaces amigáveis.
- **Módulo servidor:** é responsável pelo armazenamento dos dados , acesso, pesquisa e outras funções.

Arquitetura de Banco de Dados em Jogos

Jogos digitais podem utilizar seus próprios bancos de dados sem necessariamente utilizar um SGBD, as bases de dados podem ser utilizadas apenas para controle interno ou também para os próprios usuários do jogo. Os SGBD podem ser utilizados caso seja necessário o controle de processamento multiusuário do jogo.

Em jogos existem alguns tipos diferentes de banco de dados como, por exemplo:

- Persistência para salvar e carregar o progresso do jogo;
- Base de dados de tipos;
- Base de dados de modelos;

Persistência salvar / carregar:

Normalmente ao jogar os usuários precisam parar e continuar o jogo, para isso é necessário salvar e posteriormente carregar esse ponto em que o jogo foi parado. Para isso é utilizado um banco de dados, ou estrutura de persistência, dependendo do tipo de jogo ou complexidade para carregar e dar continuidade ao jogo.

Base de dados de Tipos:

O objetivo desse banco de dados é guardar informações de modelos do jogo, como por exemplo: *artwork*, estatísticas básicas, *assets*, imagens, sons entre outros. Normalmente este tipo de banco de dados segue o modelo orientado a objetos, sendo, porém, implementado utilizando banco de dados relacional. Deve-se ressaltar que as informações neste tipo de base não sofrem alterações, são utilizadas na definição e acessadas pelo jogo como referência de seus mapas, objetos, personagens, etc. Em alguns jogos que existem a possibilidade de alterar mapas, criar novos cenários, esta base pode então ser atualizada ou inseridos novos dados.

Base de dados de Modelos:

O objetivo desse banco de dados é agregar os modelos utilizados no jogo. Alguns jogos com muitos usuários tem a necessidade de controlar vários modelos simultaneamente, nestes casos o banco de dados deve ser utilizado. Existem situações que essa base é em casos que o “mundo” não cabe inteiro na memória do computador que está executando o jogo, ou apesar de caber em memória existem muitos modelos como IA, física ou imagens.

Outro motivo para se utilizar o SGBD é a necessidade de sincronização quando o jogo é processado em várias máquinas (cliente e servidor para os jogos *online* ou *multiplayers*).

No caso dos jogos, o banco de dados normalmente é hierárquico, pois tem um melhor desempenho que o banco de dados relacional, pois a base de dados de modelos deve ser de acesso rápido para o jogo.

O Modelo Relacional

O modelo relacional foi concebido por *Ted Codd*, da *IBM Research*, em 1970 que baseou-se em relações matemáticas simples. Atualmente muitos SGBD que incluem o modelo de entidade-relacionamento, alguns deles são: DB2, Oracle, Sybase, SQL Server, MySQL e PostgreSQL.

O modelo relacional representa o banco de dados como uma coleção de relações e tabelas. Cada relação que existe é representada por uma tabela.

Quando os dados são representados com uma tabela, os valores de cada linha representam uma coleção relacionada de dados. Uma linha da tabela normalmente representa uma relação ou uma entidade do mundo real.

O Modelo relacional possui os seguintes conceitos:

- **Domínio:** É o conjunto de valores possíveis para preencher determinado campo (ou conjunto de campos).
- **Atributos:** um cabeçalho da coluna.
- **Tuplas:** uma linha da tabela.
- **Relações:** as ligações de referencia entre tuplas de diferentes tabelas.

Para melhor representar o modelo relacional existe um diagrama conhecido como DER (**Diagrama de Entidade-Relacionamento**). O digrama possui 3 tipos básicos de elementos:

- Entidades
- Atributos
- Relacionamentos

Entidades:

São conjuntos de coisas de interesse para uma determinada aplicação, normalmente representam alguma forma física do mundo real. Uma entidade é membro ou instancia de um tipo de entidade. As entidades devem ser identificadas de forma única para que seja possível criar as relações. São normalmente representadas por um retângulo, conforme mostrado na figura 06:

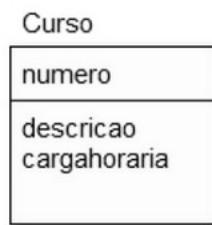


Figura 06: Entidade no modelo E-R.

Atributos:

São propriedades dos tipos de entidades ou relacionamentos. Um tipo de entidade deve ter uma **chave primária** assim como outros atributos descritivos. No diagrama os atributos adicionais podem ser omitidos e adicionados em uma lista separada.

Algumas ferramentas de criação de DER emitem os atributos por completo, tendo um gráfico formado apenas pelas entidades.

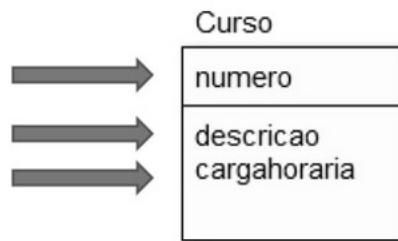


Figura 07: Entidade no modelo E-R.

Relacionamentos:

Relacionamento são associações entre tipos de entidades, normalmente o nome do relacionamento aparece sobre a linha que cria a ligação. Os nomes utilizados para relacionamentos são verbos, uma vez que as entidades são substantivos. Podem existir relacionamentos envolvendo dois tipos diferentes de entidades, um único tipo ou até mesmo múltiplos tipos.

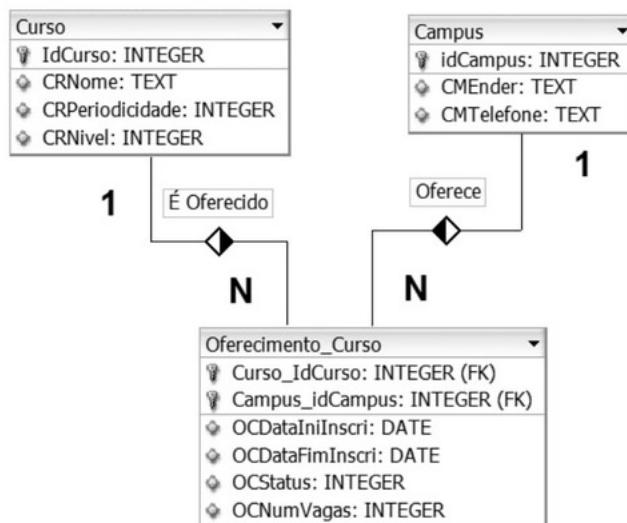


Figura 08: Exemplo de Relacionamentos no modelo E-R.

Os relacionamentos seguem um sentido natural com a linguagem, por exemplo, ao dizermos que um curso tem um oferecimento, já fica evidente a relação entre Curso e Oferecimento.

Cardinalidade de Relacionamentos

A cardinalidade restringe o número de objetos que participam do relacionamento. Com base nas relações podemos concluir que o oferecimento sempre está relacionado a um curso. Analisando a relação na direção oposta, temos que cada curso está relacionado a zero ou mais oferecimentos.

Curso

IdCurso	CrDescri	CrCargaHoraria
1	Jogos Digitais	3000
2	Técnico em Informat	800
3	Gestão de Projetos d	1200
4	Medicina	6000

Campus

IdCampus	CMEnder	CMTelone
5	Maracanã	21-2834-8362
6	Eng Paulo de Frontin	24-2922-8233
7	Nilópolis	24-89364-7832
8	Volta Redonda	24-4693-7437

Oferecimento_Curso

Curso_IdCurso	Campus_idCampus	OCDataIniInscri	OCDataFimInscri	OCStatus	OCNumVagas
1	6	02/07/2018	09/07/2018	A	40
2	6	02/07/2018	09/07/2018	A	35
2	5	09/07/2018	16/07/2018	P	35
2	8	11/07/2018	18/07/2018	P	30
3	7	03/07/2018	10/07/2018	A	40
4	8	11/07/2018	18/07/2018	P	25

Figura 09: Exemplo de Relacionamentos no modelo E-R.

Relacionamento são associações entre tipos de entidades, normalmente o nome do relacionamento aparece sobre a linha que cria a ligação. Os nomes utilizados para relacionamentos são verbos, uma vez que as entidades são substantivos.

Cardinalidade é a relação de ocorrências entre instâncias das entidades. Significa que uma ocorrência de um entidade pode estar ligada a uma ou mais ocorrências de outra entidade. A cardinalidade por ser representada pela notação 1:N, N:M, 1:1 ou N:M.

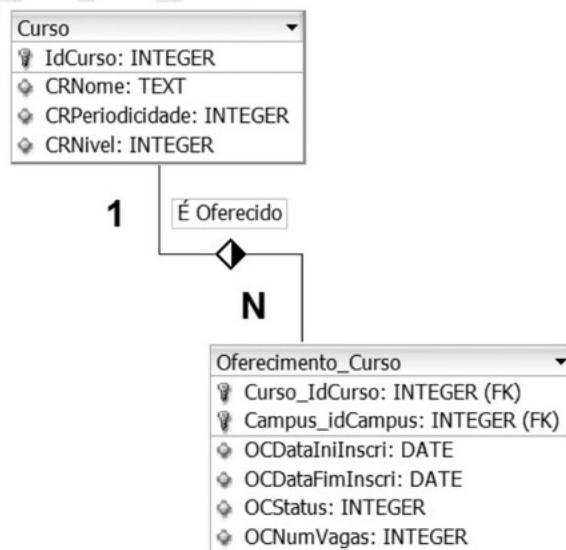


Figura 10: Cardinalidades de Relacionamentos no modelo E-R.

Classificação de Cardinalidades: podem ser classificadas pela quantidade mínima e máxima de ocorrências das duas entidades participantes do relacionamento. Uma cardinalidade mínima de um ou mais indica um relacionamento obrigatório.

Por exemplo, o relacionamento tem é obrigatório para cada entidade de oferecimento, devido a cardinalidade um. Este tipo de relacionamento obrigatório torna o tipo desta entidade depende de existência do relacionamento. Uma ocorrência da entidade oferecimento não pode existir sem estar relacionada com uma ocorrência da entidade curso. Por outro lado, uma cardinalidade zero indica um relacionamento opcional. Por exemplo, o relacionamento tem para a entidade curso é opcional, pois uma entidade curso pode existir sem se relacionar com uma entidade oferecimento.

Tabela 08: Restrições de Cardinalidade de Relacionamento.

Classificação	Restrições de Cardinalidade
Obrigatória	Cardinalidade mínima ≥ 1
Opcional	Cardinalidade mínima = 0
Funcional ou de valor único	Cardinalidade máxima= 1
1:M	Cardinalidade máxima = 1 em um sentido e cardinalidade máxima > 1 em outro sentido
N:M	Cardinalidade máxima > 1 nos dois sentidos
1:1	Cardinalidade máxima = 1 nos dois sentidos

Modelagem de Banco de Dados relacionais

Não se deve subestimar a complexidade do trabalho de modelar um banco de dados a ser utilizado por um software. Existem diversos fatores que podem afetar a qualidade do processo de modelagem, levando a construção de um modelo de dados que não prevê diversas situações do mundo real que deveriam estar refletidas no software. A seguir estão alguns fatores que impactam o processo de modelagem:

- A diferença de perfis dos participantes das reuniões de modelagem;
- Não analisar os casos de excessão;
- Não registrar por email ou ata de reunião as decisões de modelagem;

As Reuniões de Modelagem

O primeiro fator a que se deve ter atenção é a diferença de perfis dos participantes das reuniões de modelagem. Essas reuniões começam a ser feitas no início do processo de desenvolvimento do projeto. Participam dela o especialista em modelagem (normalmente um apenas) e um ou mais especialistas no processo que será atendido pelo software, por exemplo: o coordenador da área de compras da empresa quando for o projeto for um novo sistema de compras. O profissional especialista deve estar atento a esta diferença de perfis e ter em mente que os especialistas no processo possuem o conhecimento processual de negócio e ele possui o conhecimento técnico para criar um modelo de dados funcional que registre e funcione de forma fiel às informações para a organização. Esta diferença de perfis entre os participantes das reuniões torna essencial a validação por todos os participantes dos modelos de dados que estão sendo construídos, de forma mais formal, por meio de emails de confirmação ou atas de reuniões assinadas.

À medida que o projeto evolui e o processo de modelagem avança, pode surgir a necessidade da participação de outros especialistas no negócio, como por exemplo: especialista em pagamentos para informar o processo de pagamento dos fornecedores para o sistema de compras. O especialista no processo de modelagem deve estar atento para estas necessidades de informações complementares do projeto que está sendo desenvolvido e sempre solicitar a participação de outros especialistas no processo ao menor sinal de necessidade de informações complementares.

Outra fonte de informações são os documentos fornecidos pela própria instituição para o processo de modelagem. O uso destes documentos e a origem deles deve ser comunicado aos participantes do projeto logo no início dos trabalhos para evitar retrabalhos devido ao uso de documentos desatualizados ou descontinuados.

Outro ponto a ser ressaltado é a discrepância de detalhamento e importância entre o falado verbalmente e o registrado e validado (por todos os participantes) por escrito. Quando se registra e valida por escrito informações que influenciam o funcionamento do sistema que está sendo desenvolvido, os participantes tendem a demandar mais tempo e atenção no detalhamento destas informações, melhorando a qualidade do sistema que está sendo desenvolvido. Desta

forma é imprescindível a validação das informações passadas pelos especialistas no processo para o especialista em modelagem por meio de ata de reunião. Caso não tenha havido tempo para se fazer uma ata de reunião para ser assinada ou o relacionamento entre os participantes não permita essa cobrança, pode-se enviar depois um email com os principais pontos falados na reunião que influenciam a construção do sistema.

Os casos de excessão

Um outro problema muito comum no desenho de um software é quando se modela exclusivamente o caminho (ou fluxo) básico ou principal de cada processo, ou seja, modela-se apenas o funcionamento correto do sistema, esquecendo-se de modelar os fluxos de exceção e fluxos alternativos. A consequência disto é a falta de verificação e tratamento de erros pelo sistema de situações que ocorrem no mundo real como por exemplo:

- Devolução de mercadoria entregue;
- Registrar o cancelamento do pedido;
- Extravio da entrega, por roubo ou acidente.

A falta de críticas (e de informações para se poder aplicar as críticas nas transações de entrada de dados) podem gerar os seguintes comportamentos (indesejados) nos sistemas, como por exemplo: Aluno se inscreve em disciplina que não está sendo oferecida no período; Aluno recebe presença em dia de aula fora do período letivo; professor lança nota para aluno em disciplina que não leciona;

Podemos observar que o banco de dados não somente deve ser capaz de armazenar as informações geradas pelo sistema, como também prover ao sistema de informações necessárias para que as informações de entrada possam ser validadas, de forma a poder representar

O processo de modelagem de dados

A construção do modelo de dados, além de ser feito por diversas pessoas com diferentes perfis, é feita por um contínuo processo de refinamento e ajuste. Chegamos a dizer que deve ser feito à lápis no papel, pois será escrito e apagado diversas vezes, conforme o modelo for evoluindo. Esses constantes ajustes são considerados normais pois à medida que surgem novas informações durante as reuniões de modelagem, são feitas alterações no modelo de dados que está sendo desenvolvido para contemplar essas novas informações e funcionalidades. Ressaltando-se que estas modificações devem ser periodicamente validadas entre os membros do projeto.

O processo de modelagem de dados deve começar primeiramente quando já estiverem definidas as seguintes informações sobre o sistema a ser construído:

- Objetivos ,
- Identificação das principais funções do sistema a ser construído
- Escopo do sistema.

O processo de modelagem se inicia com a identificação dos principais objetos e relacionamento entre estes. É fortemente aconselhável listar todos os objetos de forma abrangente, sem muitos filtros, para que nenhuma informação ou regra deixe de ser avaliada durante o processo.

Podemos tomar como exemplo o sistema acadêmico, no qual, inicialmente possui a seguinte lista de objetos a serem analisados:

Aluno, Disciplina, Matrícula, Curso, Professor, Dia de Aula, Nota de Aluno, Frequência de Aluno, Início das aulas, Fim das Aulas, Aprovação, Reprovação, Período Letivo....

Inicialmente consideraremos as seguintes informações , construindo o primeiro esboço do modelo, no qual começamos a construir colocando as o termos mais sólidos, que temos certeza que possuem existência e informações próprias, por exemplo Aluno e Curso:

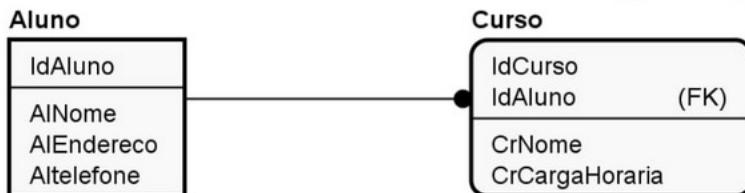


Figura 12: Relacionamento 1:N Aluno x Curso

Podemos observar que as entidades Aluno e Curso possuem existência própria e estão ligadas por um relacionamento N:M, no qual um aluno pode cursar um ou mais cursos e um curso pode ser cursado por um ou mais alunos (de preferência vários).

Pré-validando cada criação de entidade

Podemos validar cada entidade definindo os atributos de cada uma, por exemplo: a entidade Aluno teria os atributos nome, endereço e telefone e a entidade Curso teria os atributos data de início, nome e data de encerramento. Não será necessário nesta etapa esgotar todos os atributos de cada entidade colocada no modelo, mas temos que sentir confiança que cada grupo de dados (grupo de atributos) corresponde a um grupo de informações coeso (Entidade). Uma técnica bastante eficiente é criar exemplos para cada entidade. Por exemplo, um curso poderia ser formado pelos seguintes atributos: código 00052, “Excel Avançado”, “80 Horas” e um exemplo de Aluno poderia ser: código 010467, “Rafael dos Santos Almeida”, “Rua Santa Clara 201 casa, Japeri”, “024-3422-8092”.

Desta forma, já podemos marcar os seguintes termos como avaliados: Aluno e Curso.

Podemos marcar-los como analisados aplicando um traço sobre eles (no caso de utilizarmos papel e lápis) ou criar uma lista de termos já analisados. É desaconselhável removê-los da lista

pois perderíamos a informação de quais termos já foram analisados para a construção do modelo.

Voltando a lista, encontrariam o termo Matrícula. Como um aluno poderia estar inscrito em mais de um curso e um curso possui, naturalmente, mais que um único aluno (o relacionamento entre Aluno e Curso é N:M), não seria possível guardar estas informações nas entidades Aluno ou Curso.

Observamos que esta informação (Matrícula) corresponde ao relacionamento entre as entidades Aluno e Curso, possuindo informações como: código de matrícula, Data da matrícula, Status (ativa, inativa). Isso significa que matrícula é uma relação entre as entidades Aluno e Curso, sendo representada pela entidade de nome Matrícula, possuindo os atributos citados anteriormente e herdando as chaves primárias das entidades Aluno e Curso.

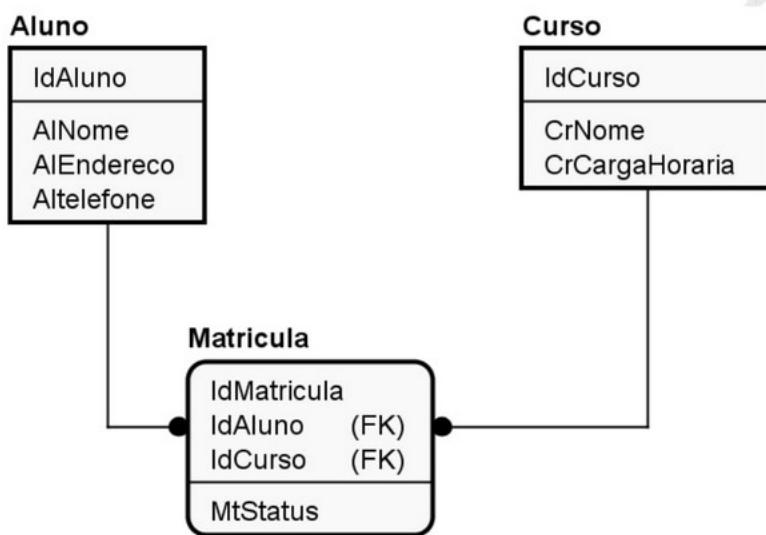


Figura 13: Relacionamento N:M Aluno x Curso

Conforme apresentado na figura anterior, o relacionamento Aluno → Matrícula é representado pela presença do atributo IdAluno, que é uma chave estrangeira (FK) na entidade Matrícula que identifica um único aluno associado a respectiva Matrícula. De forma análoga, o relacionamento Curso → Matrícula é representado pela chave estrangeira IdCurso na entidade Matrícula, a qual identifica um único Curso associado a respectiva Matrícula.

Podemos observar que cada chave estrangeira deve conter a identificação de uma única linha da entidade referenciada. Como cada entidade é identificada unicamente pela sua chave primária, as referências (chaves estrangeiras) a entidade em questão devem obrigatoriamente ser pela chave primária. Desta forma, uma regra básica da representação dos relacionamentos no modelo relacional é a seguinte:

Toda chave estrangeira presente em uma entidade somente pode ser preenchida somente com um dos valores da chave primária existente na tabela referenciada. Um exemplo prático disso é somente podemos criar Matrículas a partir de alunos cadastrados (Utilizar códigos de IdAluno que existam na tabela Aluno) e de Cursos cadastrados (utilizar códigos IdCurso que existam na tabela Curso).

É importante ressaltar que os exemplos já vistos utilizaram chaves primárias compostas de chaves estrangeiras de um único atributo (IdCurso e IdAluno). Mas devemos observar que existem também chaves primárias que são compostas de vários atributos como na entidade Inscricao_Disciplina, como mostrado na figura 13 apresentada a seguir, a tabela Frequencia_Aluno possui a chave estrangeira (do relacionamento composta pelos dois atributos IdMatricula e idDiscPeriodo, originados da chave primária da entidade Inscricao_Disciplina):

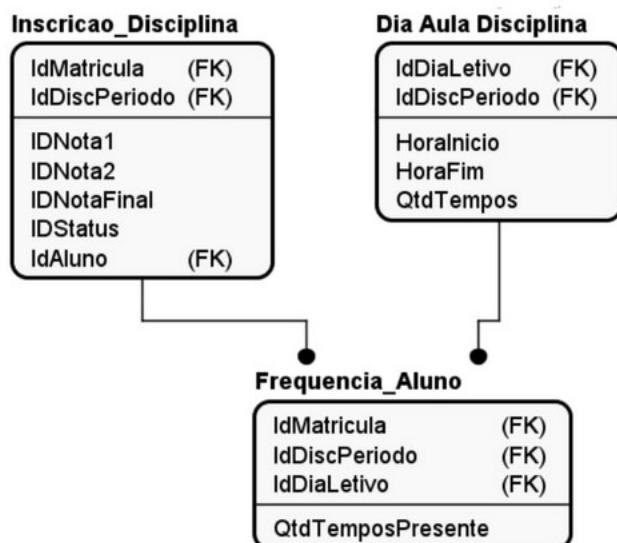


Figura 13: Chave estrangeira composta de vários atributos.

Validando cada evolução do modelo

Uma vez que a entidade Matrícula foi criada, podemos voltar a lista de termos para prosseguir. Na verdade, um passo importante para garantir a qualidade do modelo é a colocar à prova a relação Aluno → Curso. Isto significa que devemos verificar se a entidade Matrícula pode representar as situações de matrícula de alunos em cursos em uma instituição acadêmica do mundo real.

A validação progressiva do modelo é bastante eficiente na obtenção da maturidade do modelo de dados. As situações necessárias para se validar cada evolução do modelo podem ser obtidas na documentação dos casos de uso do sistema que está sendo desenvolvido, caso seja um sistema de informação ou na documentação do GDD, caso seja um jogo que esteja sendo projetado. Recomenda-se validar cada evolução do modelo durante as reuniões de modelagem, pois a validação do modelo é um trabalho de responsabilidade de todos os envolvidos no processo e não somente do profissional especialista em modelagem de dados.

No caso da entidade Matrícula, oriunda do relacionamento Aluno-Curso, podemos utilizar por exemplo as seguintes situações para validação:

1. Um aluno pode estar matriculado em mais de um curso?
2. Um curso pode possuir um ou mais alunos matriculados?
3. Um aluno pode se re-matricular em um mesmo curso?

Validando em detalhado cada situação, de preferência na reunião de modelagem, com a participação dos especialistas presentes (em modelagem e no negócio), seguiremos a ordem das situações apresentadas anteriormente:

1. Um aluno pode estar matriculado em mais de um curso?

No mundo real, um aluno normalmente está matriculado em apenas um curso. Mas estar matriculado no momento atual não significa que não tenha cursado outros cursos no passado (nem é um impedimento para se matricular em outros cursos no futuro). Observamos que o relacionamento Aluno - 1:N - Matrícula é capaz de representar esta situação do mundo real.. Observamos que, pela definição da chave primária da tabela Matrícula (IdAluno e IdCurso), o aluno pode se matricular em mais de um curso, ressaltando os cursos nos quais um aluno foi matriculado devem possuir um IdCurso diferente, pois este conjunto de linhas da tabela Matrícula terá o mesmo IdAluno. Isto deve ocorrer para garantir a não-duplicidade da chave primária da tabela Matrícula.

2. Um curso pode possuir um ou mais alunos matriculados?

Ao observarmos o relacionamento entre Curso - 1:N - Matrícula, verificamos que o modelo contempla várias matrículas em um mesmo curso, contanto que, neste conjunto de matrículas em um mesmo curso, cada matrícula possua um IdAluno distinto, já que possuirá um mesmo IdCurso. Desta forma a unicidade da chave primária de Matrícula (IdAluno e IdCurso) contempla esta situação.

3. Um aluno pode se re-matricular em um mesmo curso?

Neste caso, a unicidade da entidade Matrícula (IdAluno e IdCurso) permite a associação somente um determinado curso por cada aluno. À primeira vista parece coerente, mas se pensarmos na representação no modelo de situações do passadas e futuras do mundo real, um aluno poderá se re-matricular em um curso no qual não conseguiu completar, seja por que motivo for. Neste caso a unicidade da informação em Matrícula (IdAluno e IdCurso) não contempla duas matrículas de um mesmo aluno em um mesmo curso. Neste caso será necessário ajustar a chave primária de Matricula para que possa existir a repetição dos mesmos par de valores (IdAluno e IdCurso) em Matrícula.

Para se resolver esta limitação de representação deve-se ajustar a chave primária de Matrícula. Pode-se adicionar um campo Status à chave primária (MtStatus), mas haveria o problema de representar um aluno reprovado mais de uma vez em determinado curso (MtStatus = *Reprovado*), pois esta situação violaria a unicidade da chave primária de Matrícula.

Surrogate Key

Uma solução para o problema de se definir a unicidade de uma entidade a partir de uma informação que pode mudar (mesmo sobre uma probabilidade pequena) é a utilização de uma chave sequencial gerada (*Surrogate Key*) para definir a unicidade de uma entidade. Na figura 14,

a Surrogate Key é gerada sequencialmente, garantindo a unicidade da chave primária de Matrícula e os campos IdAluno e IdCurso são chaves estrangeiras, não fazendo mais parte da chave primária. A representação a seguir ilustra a utilização de uma *Surrogate Key*:

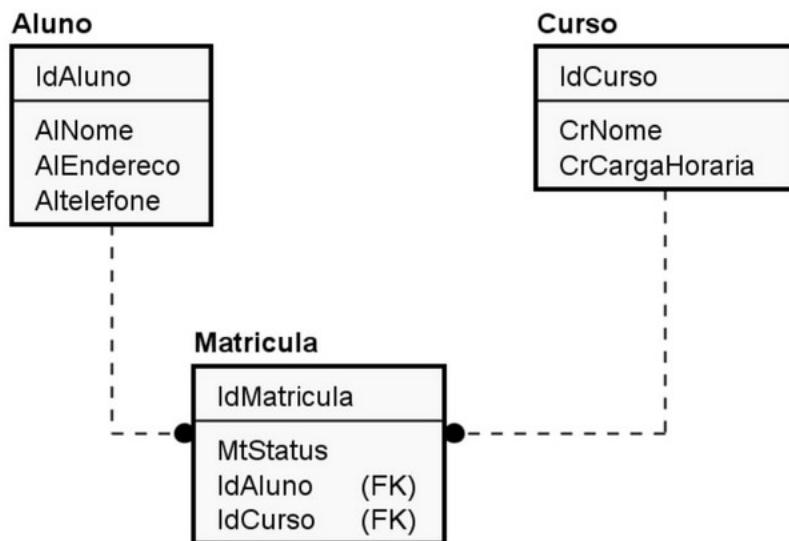


Figura 14: A Surrogate Key IdMatricula.

Vantagens da Surrogate Key:

A principal vantagem da surrogate key é a garantia de unicidade, pois como é uma chave gerada sequencialmente desvinculada com informações de objetos do mundo real Não há a possibilidade de surgir uma necessidade de mudança do valor de uma surrogate key

Com a utilização de surrogate keys as chaves primárias das tabelas herdam um menor número de atributos para compor a chave primária, o que torna as chaves primárias e estrangeiras menores, aumentando a eficiência do processamento das consultas e alterações no banco de dados.

Desta forma, configuramos a apresentamos a entidade Matrícula com a chave primária IDMatricula ao invés de utilizar o IDAluno e IDCurso combinados. Neste caso, a abordagem de surrogate key apresenta algumas vantagens, como nas situações descritas a seguir:

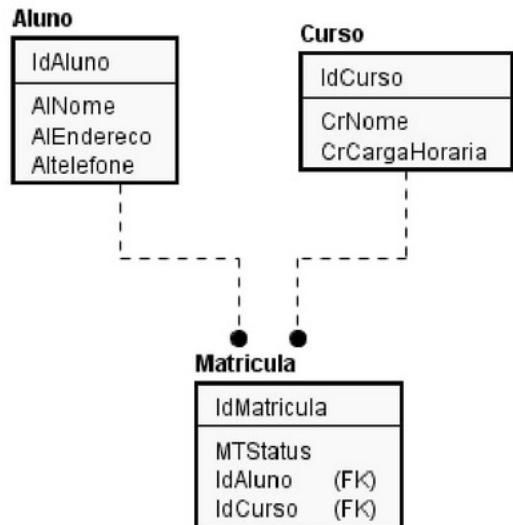


Figura 14: A Surrogate Key IdMatricula.

Observamos na definição da chave primária de matrícula, o campo IdMatricula, por ser gerado sequencialmente torna possível diversas matrículas de um mesmo aluno em um mesmo curso. Neste caso, poderia existir a situação de um aluno se matricular em um curso e desistir por algum motivo e depois de um certo tempo se matricular de novo no mesmo curso que desistiu. Neste caso existiriam duas linhas na tabela Matricula, uma com o código do aluno e o código do curso referentes a primeira matrícula (provavelmente com MTStatus = 'C' ou 'T' para indicar se é cancelado ou trancado) e outra linha com os mesmos códigos do aluno e do curso, mas com outro IDMatricula e MTStatus = 'A' para indicar que é uma matrícula ativa.

Desvantagens da Surrogate Key

A Surrogate Key substitui as chaves herdadas na composição da chave primária, sendo estas chaves herdadas definidas apenas como chaves estrangeiras na entidade em questão. Desta forma as entidades dependentes da entidade em questão receberão como chave estrangeira a surrogate key no lugar das chaves estrangeiras que poderiam referenciar outras entidades, sendo necessário consultar estas entidades para se obter mais informações. A consequência disso é necessidade de envolver mais tabelas na elaboração de uma consulta para buscar informações distribuídas pelas tabelas. A figura 15 apresentada a seguir ilustra um exemplo do aumento da quantidade de entidades envolvidas em uma consulta devido a utilização de surrogate keys.

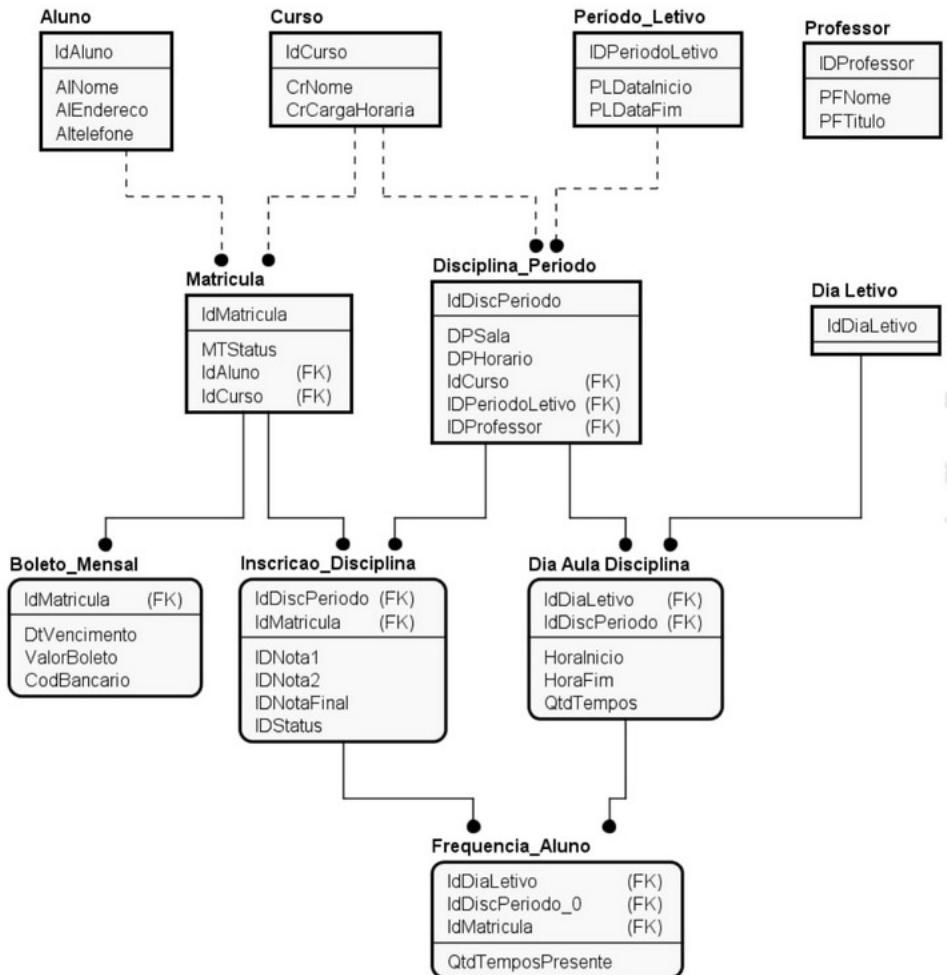


Figura 15: Propagação de Surrogate Keys no lugar de chaves identificadoras.

Para se construir uma consulta que informe o nome do aluno, a frequência deste em uma determinada disciplina oferecida em um determinado período será necessário criar uma query SQL que envolva as entidades Frequencia_Aluno, Inscricao_Disciplina, Matricula e Aluno. O processamento de uma query SQL envolvendo quatro tabelas pode exigir um esforço do sistema gerenciador de banco de dados.

Caso a chave primária da tabela Matricula seja IdAluno e IdCurso no lugar da chave sequencial IdMatricula, o código identificador do aluno (IdAluno) seria propagado pelas tabelas Matricula, Inscricao_Disciplina e Frequencia_Aluno. Uma consulta para buscar o nome do aluno, a frequência deste em uma determinada disciplina oferecida em um determinado período iria envolver as tabelas Aluno e Frequencia_Disciplina, que provavelmente exigiria menos recursos do sistema gerenciador de banco de dados para ser processada.

A Dimensão tempo

Outro ponto que às vezes é esquecido na modelagem de dados é a dimensão tempo. Normalmente, durante as reuniões de modelagem é dado um foco somente na situação atual das informações, negligenciando-se o registro do que aconteceu no passado. No exemplo a seguir, o

sistema acadêmico possui uma entidade chamada de **Período_Letivo**. A associação da entidade Período_Letivo com a entidade **Disciplina_Período** nos mostra que a entidade Disciplina_Período possui informações sobre o oferecimento de disciplinas atual (Período atual) e sobre o passado, informando as disciplinas oferecidas aos alunos nos períodos passados e suas respectivas notas, armazenadas na entidade **Inscrição_Disciplina**.

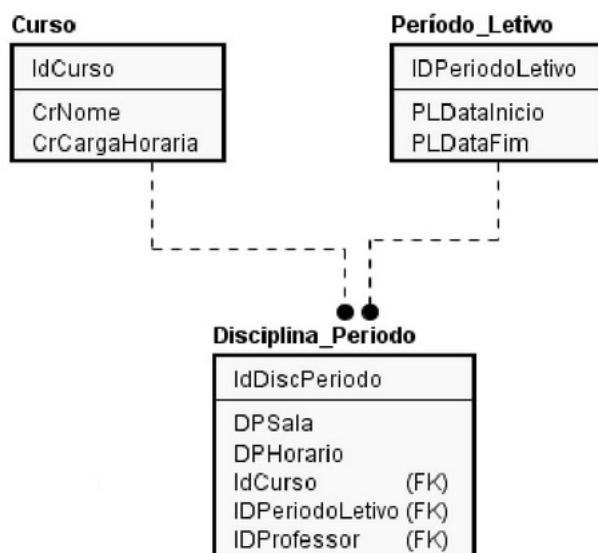


Figura 16: A dimensão tempo

Uma forma de se verificar se a dimensão tempo está devidamente representada no modelo de dados seria criar uma série de verificações sobre um determinado evento e aplicando-se a evolução do tempo sobre ele, como as perguntas apresentadas a seguir:

1. *Onde estão as notas das disciplinas que um aluno cursou no ano de 2016?*
2. *Qual foi o professor que lecionou “Sistemas operacionais” no primeiro período de 2015?*
3. *Que data foi o início do segundo período letivo de 2014?*

Entidades fortes e Entidades fracas

Podemos observar na figura 17, que, se colocarmos um sentido gráfico de cima para baixo nas relações de dependência, o modelo de dados tem as entidades fortes posicionadas na parte de cima, como por exemplo: Aluno, Curso e Período_Letivo

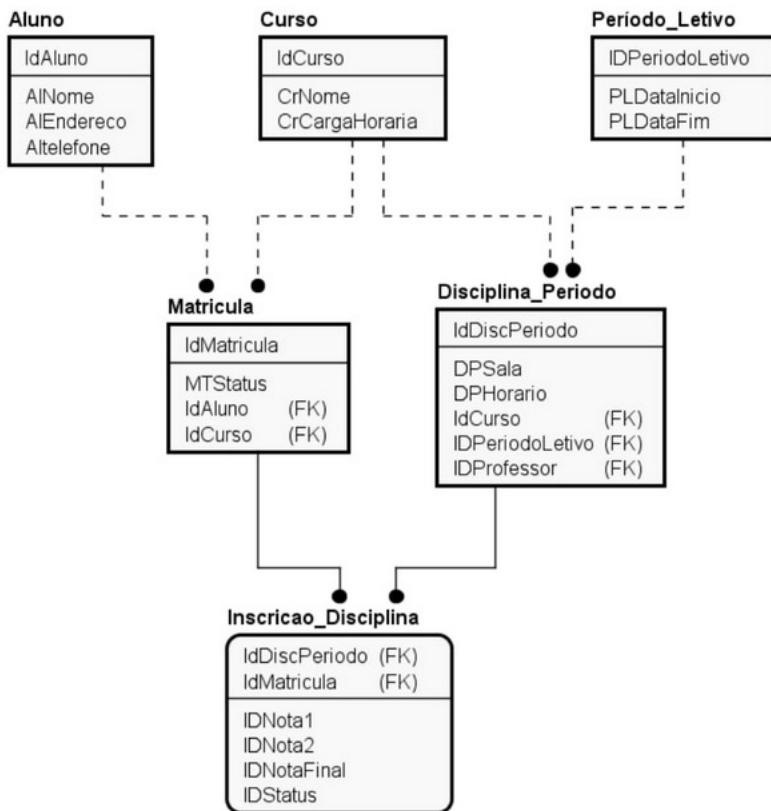


Figura 17: A hierarquia de entidades

Nos relacionamentos de linha contínua, as entidades fortes são posicionadas acima das entidades que possuem pelo menos uma relação de dependência para com as estas. Por exemplo: A entidade Inscricao_Disciplina necessita de uma Matricula e de uma Disciplina_Periodo para existir. Desta forma há a herança direta da chave primária da entidade referenciada para a entidade dependente, de forma a compor a chave primária desta. Conforme a figura 17 apresentada anteriormente, a entidade Inscricao_Disciplina herda como chaves estrangeiras o IdMatricula da entidade Matricula e IdDiscPeriodo. Estas duas chaves estrangeiras irão compor a sua chave primária.

Nos relacionamentos de linha tracejada, as chaves são propagadas como chaves estrangeiras mas não irão compor a chave primária da entidade em questão. A entidade Matricula, na figura 17 herda IdAluno de Aluno e IdCurso de Curso, mas possui como chave primária uma chave sequencial gerada (IdMatricula). Como não é possível um aluno se matricular em um curso que não existe (utilizar um valor para preencher o campo IdCurso de Matricula que não exista na tabela Curso), o banco não irá permitir o preenchimento incorreto da chave estrangeira IdCurso.

Valores Nulos

Existem situações que não se possui a informação para se preencher determinados atributos das entidades. Um dos casos presentes na figura 17 apresentada anteriormente é quando se oferece uma disciplina e ainda não foi definido qual professor irá ministrá-la. Neste caso, o IDProfessor de Disciplina_Periodo, por ser uma chave estrangeira não poderá ser preenchido com zeros ou

brancos, pois violaria as regras de integridade do banco de dados. A solução para isso é definir o campo IdProfessor de Disciplina_Periodo como NULL, ou seja, aceita nulos. O conceito de nulo é “ausência de valor”.

Cabe ressaltar que é recomendável que a maioria dos campos sejam criados como NOT NULL e nos casos que é necessário informar a “ausência de valor” o nulo seja utilizado.

Determinadas situações, mesmo sem haver informação para se preencher determinados campos não tornam o nulo necessário. Por exemplo: Um endereço pode ter o seu complemento preenchido, no caso de apartamento ou bloco, ou não, no caso de uma casa, onde a rua e o número são suficientes. Ou seja, o campo “Complemento” do endereço não necessariamente deve aceitar nulo, podendo ser preenchido com espaços ou branco, pois ao se definir um atributo em uma tabela como nulo, o software que utiliza o banco de dados deverá testar se um campo é nulo antes de tentar acessar o valor do respectivo campo, sob pena de receber um erro do sistema gerenciador de banco de dados.

No caso do professor associado ao oferecimento de disciplina em um semestre, apesar de ser necessária a figura do professor para se ministrar a disciplina, serão oferecidas disciplinas (representadas por instâncias em Disciplina_Período) antes do início das aulas para os alunos se matricularem. Neste momento, o preenchimento do campo IdProfessor não será necessário, podendo ser opcional (conter nulo). Podemos representar este relacionamento como um relacionamento que atinge a entidade Disciplina_Período pela lateral, conforme apresentado na figura 18 a seguir:

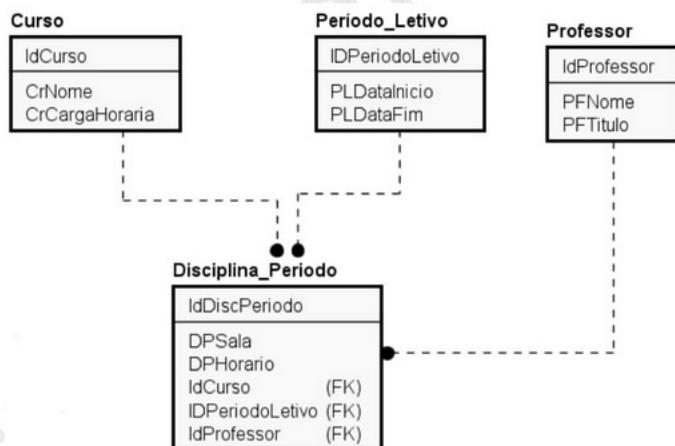


Figura 18: Relacionamento requeridos e opcionais

Regras do Mundo Real x Regras do Modelo de dados

Ao se criar um modelo de dados, deve-se sempre focar no objetivo de todo o modelo: Representar com fidelidade as informações necessárias para que o sistema funcione da forma desejada para a organização na qual foi projetado. Como já ressaltado anteriormente, o modelo deve ser validado a partir de análise de situações do mundo real.

No caso do modelo acadêmico que estamos utilizando de exemplo, observamos que a entidade Disciplina é relacionada com a entidade curso por meio de um chave estrangeira. O modelo

parece representar bem a realidade, já que existe uma ligação entre o Curso e a Disciplina. Mas, obsevemos que o relacionamento entre Curso e Disciplina possui a cardinalidade 1:N, ou seja, um Curso possui várias Disciplinas e uma determinada Disciplina pode estar ligada a somente um único curso, pois a Disciplina pode conter uma apenas uma única referência na chave estrangeira IdCurso.

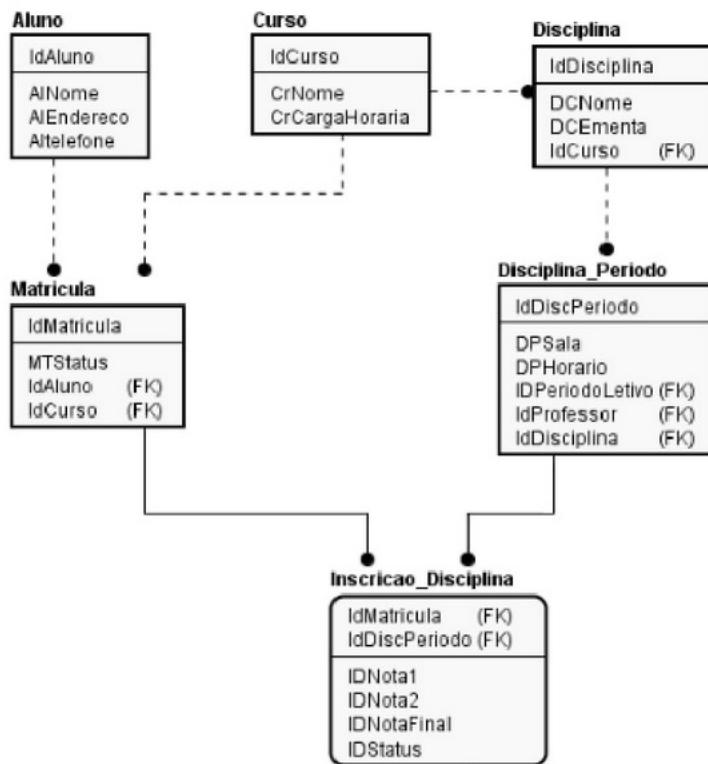


Figura 19: A tabela Curso com relacionamento 1:N com a tabela Disciplina

Esta limitação apresentada anteriormente na figura 19 pode parecer perfeitamente aceitável se for mencionada rapidamente em uma reunião de modelagem, sob a presença de participantes desatentos.

Para se avaliar se este relacionamento descrito no modelo é adequado deve-se colocar este em validade contra exemplos do mundo real. Somente desta forma se poderá verificar se o modelo é adequado.

Primeiramente iremos gerar exemplos baseados no relacionamento representado pelo modelo, para depois verificar se o conjunto gerado pode representar **todas** as situações do mundo real.

Tabela 09: Exemplos do relacionamento Curso x Disciplina

Curso	Disciplina
Jogos Digitais	Metodologia Científica
Jogos Digitais	Programação Android Studio
Jogos Digitais	Criação de Storyboard
Engenharia de Sistemas	Computação II
Engenharia de Sistemas	Computação III

Engenharia de Sistemas	Metodologia Científica
Engenharia de Produção	Metodologia Científica
Engenharia de Produção	Cálculo I
Bacharelado em Matematica	Cálculo I
Estatística	Cálculo I

Observando a tabela 09 apresentada anteriormente, nota-se que existem disciplinas que são específicas para determinados cursos como “Programação Android Studio” para o curso “Jogos Digitais” ou “Computação III” para o curso “Engenharia de Sistemas”. Mas também se observa que existem disciplinas semelhantes para cursos diferentes como “Cálculo I” para os cursos de “Bacharelado em Matemática” e “Estatística” e a disciplina “Metodologia Científica” para os cursos de “Engenharia de Sistemas” e “Engenharia de Produção”.

Desta forma podemos considerar que podem existir determinadas disciplinas que podem ser ministradas em cursos diferentes. Podemos considerar que, por exemplo, a disciplina “Cálculo I” ministrada na “Engenharia de Produção” seja diferente da disciplina “Cálculo I” ministrada no curso de “Estatística”, neste caso seria disciplinas com o mesmo nome, mas com ementas diferentes e naturalmente ministradas em salas diferentes por professores diferentes.

As considerações apresentadas anteriormente podem ser até aceitáveis sob o enfoque do armazenamento de informações. Mas quando estamos construindo um modelo que deve representar o funcionamento do mundo real, devemos verificar se no mundo real a limitação deste modelo existe, e caso exista deve-se verificar se não existe a possibilidade desta limitação ser removida do mundo real por algum motivo. Um meio objetivo de se verificar essa limitação é questionar os participantes das reuniões de modelagem (principalmente os representantes do usuários e os analistas de processos) com as seguintes perguntas:

Cada disciplina é específica de um determinado curso?

As ementas de disciplinas com o mesmo nome são diferentes para cursos diferentes?

Um aluno de um curso pode cursar uma disciplina de outro curso?

Existe a possibilidade de juntar uma turma de “Cálculo I” do curso de “Estatística” com a turma de “Calculo I” do curso de “Engenharia de Produção”?

Caso todos respondam “Sim” para as primeiras duas perguntas e Não” para as duas últimas perguntas, deve-se registrar esta decisão por meio na ata de reunião de modelagem. Caso a confecção de ata de reunião não seja um processo rotineiro das reuniões, pode-se escrever um email com esta decisão e enviar a todos, colocando-se de forma bem clara a limitação deste modelo (no caso, uma Disciplina somente estará ligada a um único Curso).

Caso os participantes da reunião cogitem a possibilidade de se juntar turmas de Disciplinas semelhantes (de Cursos diferentes), ajustando-se o modelo, conforme a figura 20. Esta flexibilidade poderia diminuir os custos, já que menos salas seria ocupadas e menos professores seriam alocados, sendo um argumento bastante considerado pelo patrocinador do projeto, quem tem o poder de dar a palavra final sobre como o sistema deverá funcionar.

Para implementar a flexibilização Curso x Disciplina, emprega-se um relacionamento N:M, no qual um Curso pode possuir várias Disciplinas e uma Disciplina pode estar associada a mais de um Curso. O relacionamento N:M, em um banco de dados relacional é representado por uma entidade dependente das duas entidades em questão.

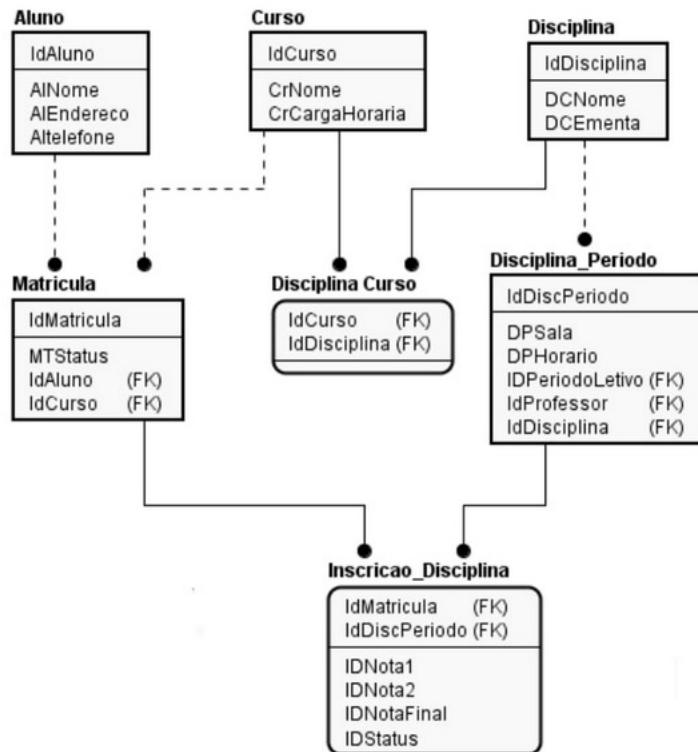


Figura 20: A tabela Curso com relacionamento 1:N com a tabela Disciplina

Atualizações das operações do mundo real para o banco de Dados

Uma característica dos sistemas gerenciadores de banco de dados é a atomicidade das operações. Mas existem algumas considerações sobre este processo. Pois nem sempre a cada atualização em uma tabela (entidade do modelo físico) o banco de dados está consistente. Existem situações nas quais o banco de dados irá refletir corretamente uma situação do mundo real após uma sequência de atualizações em uma ou mais tabelas do banco de dados.

Esta sequência de operações pode ser denominada de **Transação**. Desta forma, o banco de dados deverá prover mecanismos que a transação seja executada por completo e caso não seja possível executar a transação por completo (todos os comandos de atualização que compõem a transação), esta deverá ser desfeita por completo e os comandos que por ventura forem executados deverão ser desfeitos.

Por meio deste mecanismo, os comandos de atualização sobre um banco de dados são processados até se chegar em uma situação de consistência. Quando se atinge esta situação de consistência se executa um comando para efetivar todos os comandos executados entre a situação atual de consistência e a situação de consistência anterior antes destes comandos serem executados. Desta forma os dados armazenados no banco de dados mudam de uma

situação de consistência e outra situação de consistência, mesmo que existam diversos comandos de atualização sendo processados entre as situações de consistência.

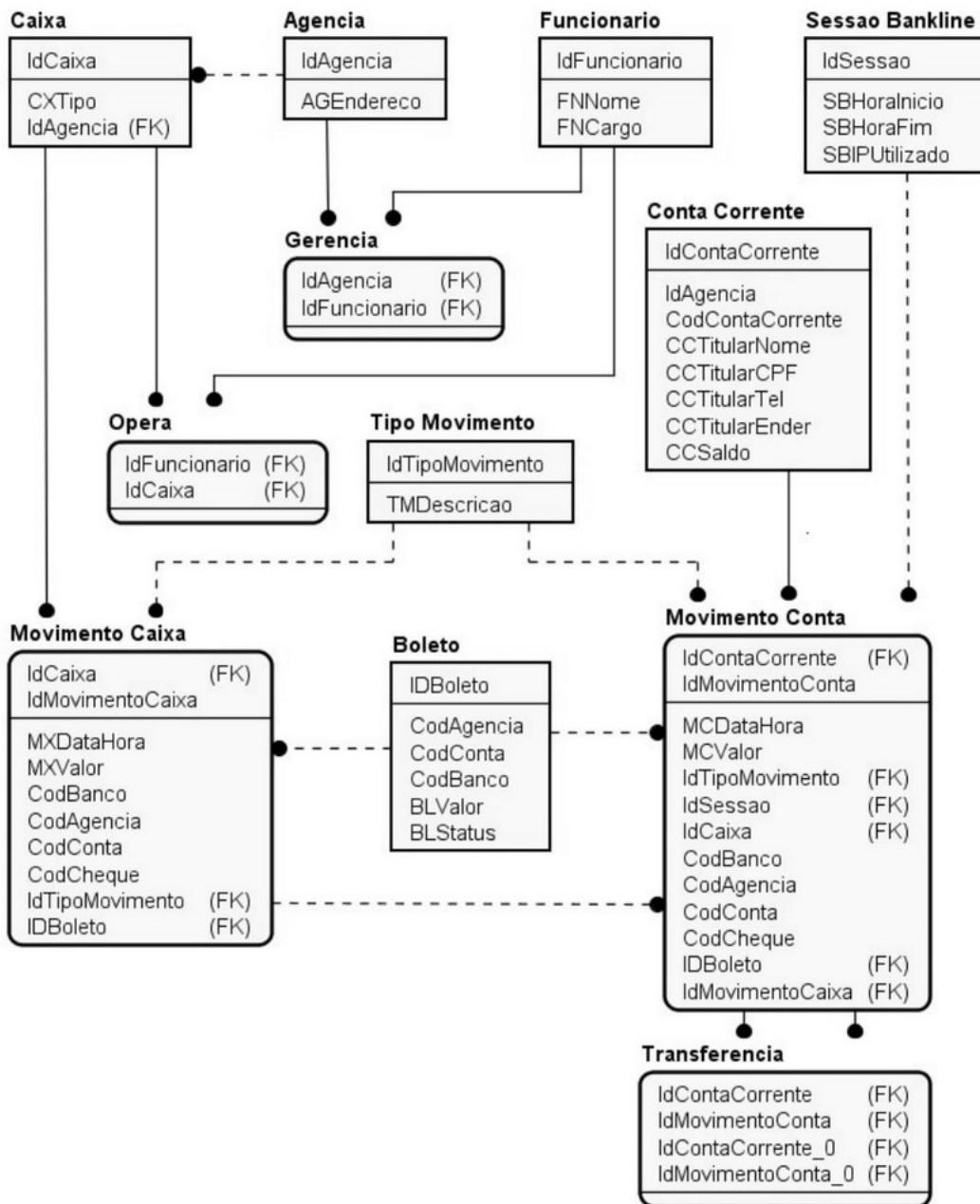


Figura 21: Um sistema bancário (simplificado)

Um exemplo de transações em um sistema comercial é o sistema bancário. Apresentado na figura 21. Por exemplo, se vamos pagar uma conta (Boleto) no caixa eletrônico (Caixa) com o cartão de conta corrente (Conta Corrente) o sistema irá processar uma transação com as seguintes atualizações:

1. Diminuir CCSaldo de Conta Corrente o valor do Boleto (BLValor);

2. Criar um registro em Movimento_Conta com MCValor preenchido com o valor do Boleto, IdBoleto preenchido com a chave do Boleto correspondente e IdTipoMovimento correspondente a “Pagamento de Boleto”;
3. Criar um registro em Movimento_Caixa com MXValor preenchido com o valor do Boleto, IdBoleto preenchido com a chave do Boleto correspondente e IdTipoMovimento correspondente a “Pagamento de Boleto”;
4. Atualizar o campo BLStatus para “P” (Pago);

As operações listadas acima devem ser executadas por completo para que a transação de pagar um Boleto seja corretamente registrada no banco de dados. A execução parcial deste comando poderá levar ao banco de dados a uma situação de inconsistência, como por exemplo, pagar o Boleto e não atualizar o valor de CCSaldo em Conta_Corrente ou registrar o pagamento mas não atualizar o campo BLStatus do Boleto.

Denominamos esta sequência de operações entre cada situação de consistência do banco de dados de Transação. O sistema gerenciador de banco de dados provê mecanismos para que os dados nele armazenados possam mudar de uma situação de consistência para outra situação de consistência.

Vale lembrar que esta situação de consistência dos dados é relativa a cada usuário do sistema gerenciador de banco de dados. O sistema gerenciador de banco de dados possui mecanismos para processar e gravar as atualizações dos diferentes usuários de forma consistente, sem que ocorram atualizações indevidas entre dados de diferentes usuários.

Quando falamos de usuário, este usuário pode ser um programa sendo executado, processando uma série de comandos (como o pagamento de um boleto no caixa eletrônico) ou por um usuário executando uma série de comandos SQL diretamente no banco de dados. Estes comandos são processados, mas ficam em uma área temporária aguardando sua efetivação. Quando se atinge o ponto de consistência de dados, executa-se um comando denominado **commit**, que torna permanente os comandos da área temporária deste usuário, atingindo uma nova situação de consistência dos dados. O sistema gerenciador de banco de dados também pode ser configurado para guardar os comandos na área temporária até um determinado limite de tempo. Caso este limite de tempo de espera seja excedido, o sistema de banco de dados descarta estes comandos temporários, de forma que se retorne a situação de consistência imediatamente anterior a que estes comandos começaram a ser executados. O comando que desfaz os comandos temporários é denominado **rollback**.

SQL (*Structured Query Language*)

O SQL (*Structured Query Language*) ou em português, Linguagem de Consulta Estruturada é uma linguagem utilizada para pesquisa em banco de dados relacionais.

História do SQL

Sua história inicia em 1970, quando o Dr. Ted Codd, pesquisador da IBM, criou um artigo sobre banco de dados relacionais.

O SQL teve inicio com outra linguagem chamada SQUARE, criada para o projeto System R da IBM. O SQUARE era uma linguagem de natureza matemática, após testes e experiências com fatores humanos a linguagem passou uma revisão e passou a ser chamada SEQUEL (*Structured English Query Language*) em português Linguagem de Consulta Estruturada, em Inglês, devido a problemas legais envolvendo o nome e devido a novas revisões o nome passou para SQL. Até os dias de hoje o SQL ("és-kiú-él") ainda é pronunciado como SEQUEL ("síquel") por muitos DBAs.

Atualmente a linguagem SQL é um padrão internacional, mas nem sempre existiu um padrão. pode parecer surpreendente mas a IBM não foi a primeira empresa a comercializar o SQL. O SQL foi utilizado por várias empresas e variações foram criadas.

Esforços para criar o padrão para a linguagem foram feitos pela ANSI (*American National Standards Institute*), a ISO (*International Organization for Standardization*) e a IEC (*International Electrotechnical Commission*). O tamanho do padrão cresceu muito de um padrão para outros. O padrão inicial (SQL-86) era formado por 150 páginas, enquanto um novo padrão (SQL-92) era formado por mais de 600 páginas.

O ponto fraco dos padrões de SQL é a falta de testes de compatibilidade, até 1996 testes eram executados para manter a compatibilidade entre os diferentes fornecedores de SGBD, porém os testes foram substituídos por testes independentes.

Histórico da Evolução do SQL

- 1972 – Projeto System R nos laboratórios de pesquisa da IBM.
- 1974 – Desenvolvimento da linguagem SQUARE.
- 1975 – Revisão da linguagem e mudança do nome para SEQUEL.
- 1976 – Revisão da linguagem e mudança do nome para SEQUEL 2.
- 1977 – Mudança do nome para SQL.
- 1978 – Primeira implementação comercial pela Oracle Corporation.
- 1981 – Produto SQL/DS da IBM com recursos de SQL.
- 1986 – Aprovação do padrão SQL-86 (SQL 1).
- 1989 – Aprovação do padrão SQL-89 (revisão do SQL- 86).
- 1992 – Aprovação do padrão SQL-92 (SQL 2).
- 1999 – Aprovação do padrão SQL:1999 (SQL 3).
- 2003 – Aprovação do SQL:2003.
- 2006 – Aprovação do SQL:2006.
- 2008 – Aprovação do SQL:2008.

- 2011 – Aprovação do SQL:2011.

Escopo do SQL

O SQL possui subconjuntos que são divididos em relação as operações que são destinadas.

- Definição de Dados (DDL)
- Manipulação de Dados (DML)
- Controle de Dados (DCL)
- Transação de Dados (DTL)

SQL – DDL - Definição de Dados

A *Data definition language* (DDL) é a linguagem utilizada para definição dos dados. É uma linguagem estruturada utilizada para criar o banco de dados e seus objetos. A definição de DDL pode ser utilizada para qualquer sintaxe de criação de estruturas de dados. DDL não é uma linguagem separada mas sim uma categoria dentro da própria SQL. A linguagem utilizada verbos que são utilizados para mudar a estrutura do banco de dados, adicionando, alterando ou removendo objetos.

Declarações comuns e básicas de DDL:

- CREATE é utilizada para criar banco de dados, tabelas, índices, etc.
- DROP é utilizado para destruir / remover um objeto do banco de dados.
- ALTER é utilizado para modificar os objetos.

CREATE DATABASE

Para a criação de um novo banco de dados o comando de criação será utilizado:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nome  
[especificação de criação]
```

Para criação do banco de dados “Faculdade” :

```
CREATE DATABASE Faculdade;
```

O comando DROP DATABASE remove um banco de dados, conforme o exemplo a seguir:

Exemplo: Para remoção do banco de dados “Faculdade”:

```
DROP {DATABASE | SCHEMA} [IF EXISTS] nome DROP DATABASE Faculdade;
```

CREATE TABLE

O comando de criação das tabelas é utilizado para definir a formação dos campos e os tipos de dados que serão utilizados. Os campos da tabela serão especificados na parte de “definição de criação”.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nome (definição de
criação, ...) [opções da tabela]
```

As colunas são especificadas durante a criação da tabela. Definir as colunas é simples, informar o nome e o tipo e também se o campo permite valores vazios ou se possui valores padrões de cada uma das colunas da tabela que está sendo criada.

```
nome_coluna tipo [NOT NULL | NULL] [DEFAULT valor padrão]
```

Tabela 10. Tipos de atributos do SQL.

Tipo de Atributo	Descrição
CHAR(L)	Coluna do tipo texto fixo onde o tamanho é definido por L, valor máximo 255.
VARCHAR(L)	Coluna do tipo texto variável, onde o tamanho é definido por L.
FLOAT()	Coluna do tipo numérico com ponto flutuante, quando informado UNSIGNED os valores negativos são desativados.
INTEGER ou INT	Coluna do tipo numérico inteiro.
BOOL ou BOOLEAN	Coluna do tipo verdadeiro / falso ou sim / não. Valor 0 é FALSE e 1 TRUE.
DECIMAL(W, R)	Coluna para dados numéricos com precisão fixa onde W é o tamanho total e R o valor decimal.
DATE	Coluna para valor de data, é exibido como AAAA-MM-DD
DATETIME	Coluna para valor de data com horário, é exibido como AAAA-MM-DD HH:MM:SS
TIMESTAMP	Coluna para armazenar data e horário com precisão de milionésimos.

Exemplo:

```
SQL File 1* 
USE Faculdade;
CREATE TABLE alunos (
    CPF CHAR(11) NOT NULL,
    Nome VARCHAR(50),
    Sobrenome VARCHAR(50),
    Cidade VARCHAR(50),
    UF CHAR(2),
    CEP CHAR(10),
    Turma CHAR(6)
);
```

Figura 24: Comandos para criar a tabela aluno no MySQL.

No MySQL é apresentada uma estrutura hierárquica dos objetos. Por meio dela é possível visualizar as tabelas de um banco de dados na guia “SCHEMAS”, até mesmo listar as colunas de uma tabela.



Figura 25: Objetos do MySQL.

DROP TABLE

O comando DROP apaga objetos do banco de dados. O comando DROP TABLE pode ser usado para apagar uma tabela. Várias tabelas podem ser removidas no mesmo comando.

Sintaxe:

```
DROP [TEMPORARY] TABLE [IF EXISTS] nome_tabela[, nome_tabela] ...
```

Exemplo:

```
DROP TABLE alunos;
```

Restrições de Integridade

Restrições de integridade ou *Constraints* são regras de integridade que podem ser definidas na criação das tabelas. Podem ser de diversos tipos, conforme listado a seguir:

- **Integridade de entidade:** é quando uma tabela possui uma coluna ou mais de uma coluna com valores únicos. Garante que os valores não se repetirão em diferentes linhas da mesma tabela.
- **Integridade referencial:** é quando uma ou mais colunas de uma tabela devem corresponder ao(s) valor(es) de uma ou mais colunas da tabela que original (que os valores destas colunas é único).

Os seguintes conceitos devem ser entendidos para podermos compreender melhor as regras de Integridade referencial:

- **Superchave:** uma coluna ou várias colunas combinadas contendo valores únicos para cada linha.
- **Chave candidata:** é uma superchave mínima, a superchave é mínima se com a remoção de qualquer coluna ela deixa de ser única.
- **Valor nulo:** é um valor que representa a ausência de valor.
- **Chave primária:** chave candidata especialmente designada, a chave primária de uma tabela não pode conter valores nulos.
- **Chave estrangeira:** uma coluna ou combinação de colunas em que os valores devem corresponder aos valores de uma chave candidata de uma outra tabela. O campo deve ter o mesmo tipo de dado da chave candidata que está sendo referenciada. A chave que está sendo referenciada na outra tabela deve ser uma chave primária, de forma que um valor da chave estrangeira seja associado a apenas uma linha da tabela que está sendo referenciada, caso contrário haveria ambiguidade na referência.

CHAVE PRIMÁRIA - CONSTRAINT PRIMARY KEY

A chave primária é a regra básica da unicidade da informação na tabela. Ela garante que cada linha (*tupla*) da tabela representa um grupo coeso de informações. A regra de chave primária deve ser criada no momento da criação da tabela:

Exemplo

The screenshot shows the SQL Server Management Studio interface. In the main pane, there is a code editor window titled "SQL File 1". The code is as follows:

```

1 USE Faculdade;
2
3 CREATE TABLE alunos (
4     CPF CHAR(11) NOT NULL,
5     Nome VARCHAR(50),
6     Sobrenome VARCHAR(50),
7     Cidade VARCHAR(50),
8     UF CHAR(2),
9     CEP CHAR(10),
10    Turma CHAR(6),
11    CONSTRAINT PKaluno PRIMARY KEY(CPF));

```

In the bottom right corner of the code editor, there is an "Output" tab with a sub-tab "Action Output". It displays two entries:

Time	Action
1 00:38:02	USE Faculdade
2 00:38:02	CREATE TABLE alunos (CPF CHAR(11) NOT NULL, Nome VARCHAR(50), Sobrenome VAR...

Figura 26: Comando para criar a tabela aluno com chave primária.

Quando a tabela possuir outras chaves candidatas que não são chaves primárias, estas devem ser declaradas como chaves únicas apenas.

RESTRIÇÕES DE CHAVES ESTRANGEIRAS (*CONSTRAINT FOREIGN KEY*)

As chaves estrangeiras devem ser definidas utilizando o comando FOREIGN KEY. Podem ser adicionadas na criação da tabela ou posteriormente como ALTER TABLE.

Exemplo:

The screenshot shows a SQL File 1 window in SSMS. The code entered is:

```
1 USE Faculdade;
2
3 ALTER TABLE matriculas
4     ADD CONSTRAINT FKnumofer FOREIGN KEY(Número) REFERENCES oferecimentos (Número),
5     ADD CONSTRAINT FKCPFaLUNO FOREIGN KEY(CPF) REFERENCES alunos (CPF);
```

The output pane shows two log entries:

Time	Action	Message
1 01:06:45	USE Faculdade	0 row(s) affected
2 01:06:45	ALTER TABLE matriculas ADD CONSTRAINT FKnumofer FOREIGN KEY(Número) REFER... 0 row(s) affected Records: 0	Records: 0

Figura 27 Adicionando chaves estrangeiras na tabela aluno.

SQL – DML

A DML (*Data Manipulation Language*) do SQL tem como principal função a manipulação dos dados armazenados no banco de dados (inclusão, modificação e recuperação das informações). A DML é formada pelos seguintes comandos para executar a manipulação de dados:

- SELECT: Para se recuperar informações do banco de dados.
- INSERT: Para inserir informações no banco de dados.
- UPDATE: Para alterar informações no banco de dados.
- DELETE: Para remover informações no banco de dados.

Comando SELECT

O comando SELECT permite a recuperação de informações do banco de dados podendo-se utilizar diversos critérios de seleção:

Sintaxe:

```
SELECT [DISTINCT] expressao [AS nom-atributo] [FROM from-list] [WHERE condicao] [ORDER BY attr_name1 [ASC | DESC ] onde:
```

DISTINCT - Para eliminar linhas duplicadas na saída.

Expressão - Define os dados que queremos na saída, normalmente uma ou mais colunas de uma tabela da lista FROM.

AS nom-atributo – Define um apelido para o nome da coluna no resultado do select. Se não for especificado será o próprio nome da coluna;

FROM - lista das tabelas consultadas (não necessariamente selecionadas);

WHERE - critérios da seleção

ORDER BY - Critério de ordenação das tabelas de saída. Podem ser:

ASC - ordem ascendente (crescente) dos valores das colunas informadas;

DESC - ordem descendente (decrescente) dos valores das colunas informadas.

Exemplo de SELECT:

```
Select IdCurso, CrDescri,  
      CrCargaHoraria  
From curso  
Where IdCurso = 2;
```

Resultado:

IdCurso	CrDescri	CrCargaHoraria
2	Técnico em Informática	800

O comando SQL SELECT no Java

```

Connection con = new ConnectionFactory().getConnection();
PreparedStatement stmt = con.prepareStatement("Select IdCurso, CrDescri,
CrCargaHoraria From curso Where IdCurso = ?");
stmt.setLong(1, 2);
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
    Int numcurso      = rs.getInt("IdCurso");
    String crdrescri   = rs.getString("CrDescri")
    Int crcargahoraria = rs.getInt("CrCargaHoraria")
    System.out.println(" Curso " + Integer.toString(numcurso) +
                       crdrescri + " carga Horária: " +
                       Integer.toString(cargahoraria));
}
stmt.close();
con.close();

```

Comando SELECT envolvendo várias tabelas (JOIN).

Como o modelo relacional possui as informações normalizadas, distribuídas pelas tabelas conforme os relacionamentos (1:1 ou 1:N) entre elas, necessitamos de comandos SQL que possam acessar várias tabelas com diversas condições para fornecer as informações necessárias. O comando Select JOIN permite que sejam aplicadas diversas condições diferentes a um grupo de tabelas de forma a recuperar informações do banco de dados sem a necessidade de uma programação complexa, bastando criar um comando SQL com as condições de recuperação da informação:

A seguir um exemplo de como podemos informar as informações a partir de um modelo relacional já normalizado:

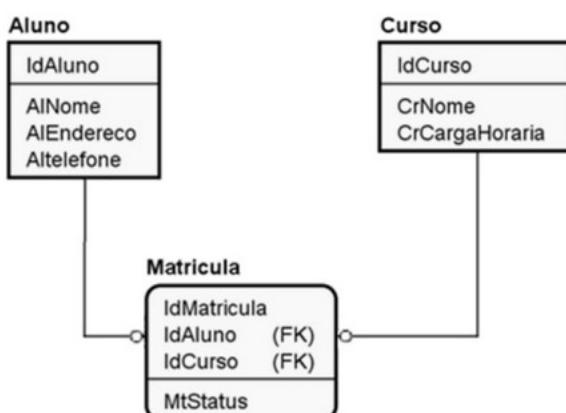


Figura 28 Tabelas do modelo relacional acessadas pelo comando SQL (JOIN).

As tabelas apresentadas na figura 28 estão preenchidas com os dados listados na figura 29, apresentada a seguir:

Tabela Curso

IdCurso	CrDescri	CrCargaHoraria
1	Jogos Digitais	3000
2	Técnico em Informatica	800
3	Gestão de Projetos de TI	1200
4	Medicina	6000

Tabela Matricula

IdMatricula	IdAluno	IdCurso	MtStatus
2018001	1	1	A
2018002	2	1	A
2018003	3	1	A
2018004	4	1	T
2018005	5	3	A
2018006	6	3	A
2018007	7	2	T
2018008	7	1	T
2018009	7	3	A
2018010	8	2	T
2018011	9	1	T
2018012	10	3	A
2018013	11	1	A
2018014	12	2	A
2018015	13	2	T
2018016	13	1	A

Tabela Aluno

IdAluno	AlNome	AlEndereço	AlTelef
1	José Ribeiro da Silva	Rua Carlos Góes 32	9385
2	Alberto Bastos de Faria	Rua Cupertino 40	9375
3	Alexandre Pi da Modulo	Rua Luzia Bruce 12	9756
4	Carlos Batalha Jackob	Rua Terezina 122	9765
5	Erica Gonçalves Ramiro	Rua Afranio 45	9655
6	Alessandra de Brito	Rua Maria da Costa 234	9845
7	José Garcia Ramirez	Rua Casuarina 33	9282
8	Douglas da Ilha	Rua Borges de Medeiros 4545	9232
9	Aroldo de Almeida	Rua Igarapava 12	9345
10	Cintia Galvão de Oliveira	Rua Gilberto Cardoso 23	918231733
11	Claudio Luis Machado Sant	Rua Martin Pena 123	925425322
12	Leonardo Moura Antunes	Av Antunes 34	906967070
13	Pedro Samembau Patrick	Rua Anita Garibaldi 34	923237333
14	Wellington Bernstein	Av Matoso 234	906068333
15	Augusto de Oliveira	Rua Montenegro 123	911121222

Figura 29: Dados das Tabelas acessadas pelo comando SQL (JOIN).

Para executarmos uma consulta ao banco de dados para obtermos todos os alunos matriculados (Alunos associados a um curso, com matrícula ativa), executaremos o seguinte comando SQL:

```
Select t1.AlNome, t2.AlCurso
From Aluno t1,
      Curso t2,
      Matricula t3
Where t1.IdAluno = t3.IdAluno
And t3.IdCurso = t2.IdCurso
And t2.MtStatus = 'A'
```

No caso do join, o select envolve várias tabelas e pode-se utilizar apelidos para cada uma das tabelas envolvidas na consulta, de forma que se possa identificar de que tabela cada coluna pertence. No exemplo apresentado a seguir define-se t1 como apelido da tabela Aluno, t2 para Curso e t3 para Matricula. A utilização desses apelidos faz-se necessária pois muitos campos possuem o mesmo nome em diferentes tabelas, normalmente campos que compõem as chaves estrangeiras e primárias.

```

Select t1.AlNome, t2.AlCurso
From Aluno t1,
     Curso t2,
     Matricula t3
Where t1.IdAluno = t3.IdAluno
And t3.IdCurso = t2.IdCurso
And t2.MtStatus = 'A'
  
```

O diagrama ilustra a execução da query SQL. As linhas de código são associadas a comentários que descrevem cada parte:

- Colunas das tabelas a serem recuperadas na consulta:** Aponta para a lista de colunas no Select.
- Tabelas envolvidas na consulta com os respectivos apelidos:** Aponta para a lista das três tabelas e seus respectivos apelidos (t1, t2, t3).
- Condições de seleção das informações:** Aponta para a cláusula Where.
- Somente serão consideradas as linhas que este campo for igual a este valor fixo:** Aponta para a condição And t2.MtStatus = 'A'.
- Ao forçarmos a um campo de uma tabela a ser igual a um campo de outra tabela estabelemos uma intersecção dos dois conjuntos de dados:** Aponta para a condição And t1.IdAluno = t3.IdAluno.

O resultado da execução do comando SQL será o seguinte:

t1.AlNome	t2.CrDescri
José Ribeiro da Silva	Jogos Digitais
Alberto Bastos de Faria	Jogos Digitais
Alexandre Pi da Modulo	Jogos Digitais
Erica Gonçalves Ramiro	Gestão de Projetos de TI
Alessandra de Brito	Gestão de Projetos de TI
José Garcia Ramirez	Gestão de Projetos de TI
Aroldo de Almeida	Jogos Digitais
Cintia Galvão de Oliveira	Gestão de Projetos de TI
Claudio Luis Machado Sant	Jogos Digitais
Leonardo Moura Antunes	Técnico em Informatica
Pedro Samembau Patrick	Jogos Digitais

A partir dos dados das três tabelas envolvidas pelo Select Join e dos resultados apresentados anteriormente, podemos observar o seguinte:

- O Curso de Medicina não possui alunos, logo ele não apareceu no resultado da *Query*.
- Os alunos 14 e 15 (*Wellington Bernstein* e *Augusto de Oliveira*) não estão matriculados em nenhum curso e não apareceram no resultado do *Select*.
- Os alunos 4, 8, e 9 não possuem matrícula Ativa (*MtStatus = 'A'*) e não apareceram no resultado da *Query*.

Comando INSERT

O comando **INSERT** adiciona um registro (linha ou tupla) a uma tabela.

Sintaxe:

```
INSERT INTO destino [(campo1[, campo2[, ...]])] VALUES (valor1[, valor2[, ...]]) Onde;
```

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

campo1, campo2 - Os nomes dos campos aos quais os dados devem ser anexados.

valor1, valor2 - Os valores para inserir em campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante.

Os valores devem ser separados com uma vírgula e os campos de textos entre aspas duplas ou simples.

Exemplo:

```
Insert into alunos (Id_aluno, nome, endereço, turma, turno) values (1, 'Glaucio', 'Av. das Américas', '1101', 'manhã');
```

Pode-se omitir os nomes de colunas, informando apenas os valores (values). Neste caso deve-se obedecer a ordem das colunas que a tabela apresenta, conforme o exemplo a seguir:

Exemplo:

```
Insert into alunos values (1, 'Glaucio', 'Av. das Américas', '1101', 'manhã');
```

Comando UPDATE

O comando UPDATE executa uma atualização de valores em um ou mais campos (colunas) em determinadas linhas de uma tabela definidas com base em critérios específicos.

Sintaxe:

```
UPDATE tabela SET campo1 = valornovo, ... WHERE critério;
```

Onde:

Tabela - O nome da tabela cujos dados você quer modificar.

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

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

Exemplo:

```
Update alunos Set turno = 'tarde' where turma = '1101';
```

UPDATE é especialmente útil quando você quer alterar muitos registros (linhas ou tuplas) em uma determinada tabela. Você pode alterar vários campos ao mesmo tempo. Observe que o comando UPDATE não gera um conjunto de resultados, se você quiser saber quais resultados serão alterados, utilize o comando SELECT com as mesmas condições de seleção do UPDATE. Desta forma, ao criar um comando UPDATE, primeiramente examine os resultados da consulta utilizando os mesmos critérios do UPDATE para verificar se é o conjunto desejado de linhas a ser alteradoe depois execute a atualização.

O comando SQL UPDATE no Java

O código Java apresentado a seguir tranca a matrícula informanda (coloca o valor “T” no campo MtStatus da matrícula correspondente).

```
public static void updateStudent (
    Connection conn,
    int idMatricula,
)
throws SQLException
{
    try
    {
        PreparedStatement ps = conn.prepareStatement(
            " Update Matricula Set MtStatus = 'T' where idMatricula = ?");
        ps.setInt(1,IdMatricula);

        // executa o update, trancando a matrícula IdMatricula
        ps.executeUpdate();
        ps.close();
    }
    catch (SQLException se)
    {
        throw se;
    }
}
```

Comando DELETE

O comando DELETE Remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE. De forma semelhante ao comando UPDATE, o DELETE não gera um conjunto de resultados, se você quiser saber quais linhas da tabela serão eliminadas, utilize o comando SELECT com as mesmas condições de seleção do UPDATE, examine primeiro os resultados da consulta e depois execute o comando DELETE. Este comando deve ser utilizado com muito cuidado, pois somente com a utilização de *Backup* pode-se restaurar as linhas deletadas.

Sintaxe:

DELETE [tabela.*] FROM tabela WHERE critério onde:

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

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

critério - Uma expressão que determina qual registro deve ser excluído.

Exemplo:

```
Delete from alunos WHERE turno='Manhã';
```

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

DELETE é especialmente útil quando você quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, você pode usar o método Execute com uma instrução DROP. Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa DELETE, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Dependendo de como as restrições de integridade estão definidas no banco de dados, o comando DELETE pode deletar as linhas das tabelas dependentes da tabela que o DELETE está removendo linhas (DELETE CASCADE) ou o banco de dados pode informar que as linhas a serem removidas possuem referências nas tabelas dependente e não executar o DELETE até que estas linhas não estejam mais referencias nas tabelas dependentes (DELETE RESTRICT). Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

O comando SQL DELETE no Java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JavaDeleteALunoExemplo
{
    public static void main(String[] args)
    {
        try
        {
            String myDriver = "org.gjt.mm.mysql.Driver";
            String myUrl = "jdbc:mysql://localhost/test";
            IdALuno = 15;

            Class.forName(myDriver);
            Connection conn = DriverManager.getConnection(myUrl, "root", "");

            String query = "delete from Aluno where IdAluno = ?";
            PreparedStatement preparedStmt = conn.prepareStatement(query);
            preparedStmt.setInt(1, IdALuno);
            preparedStmt.execute();
            conn.close();
        }
        catch (Exception e)
        {
            System.err.println("Erro no SQL! ");
            System.err.println(e.getMessage());
        }
    }
}
```

Normalização de Bancos de Dados Relacionais

Normalização de dados é o processo formal e passo a passo que examina os atributos de uma entidade, com o objetivo de evitar anomalias observadas na inclusão, exclusão e alteração de registros.

O conceito de entidade é muito importante neste processo, ou seja, devemos identificar quais são as entidades que farão parte do projeto de banco de dados. Entidade é qualquer coisa, pessoa ou objeto que abstraído do mundo real torna-se uma tabela para armazenamento de dados. Normalmente usa-se o Modelo de Entidade e Relacionamento para criar o modelo do banco.

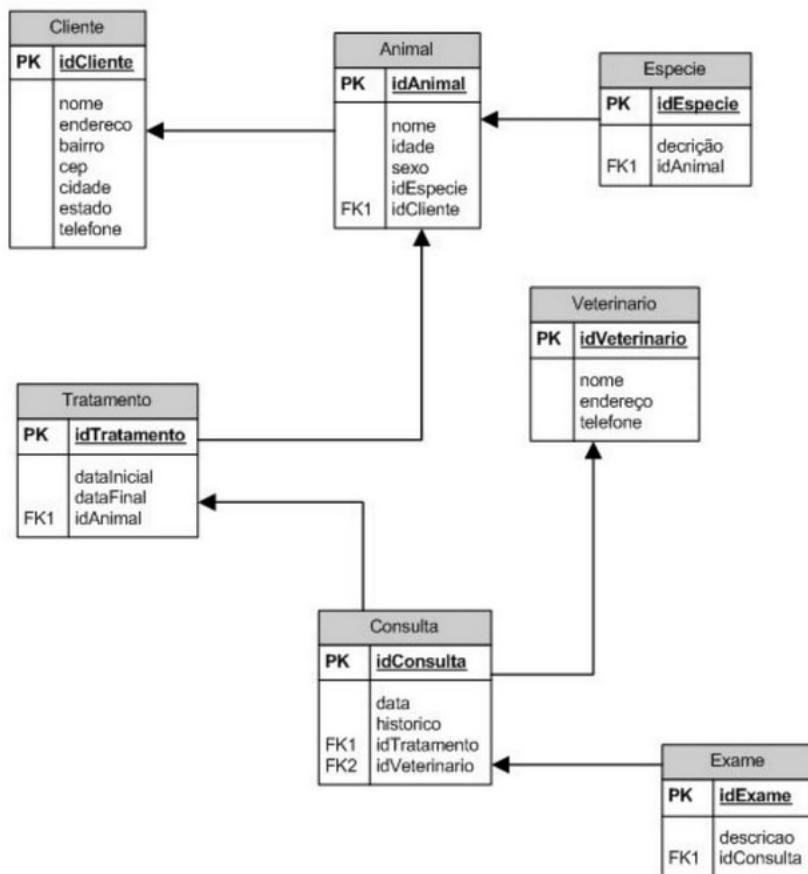


Figura 28: Exemplo de várias entidades já normalizadas:

A regra de ouro que devemos observar no projeto de um banco de dados baseado no Modelo Relacional de Dados é a de "não misturar assuntos em uma mesma Tabela". Por exemplo: na Tabela Clientes devemos colocar somente campos relacionados com o assunto Clientes. Não devemos misturar campos relacionados com outros assuntos, tais como Pedidos, Produtos, etc. Essa "Mistura de Assuntos" em uma mesma tabela acaba por gerar repetição desnecessária dos dados bem como inconsistência dos dados.

Entidade: Filmes								
idFilme	Nome	Gênero	idMidia	Idioma	Tipo	Seção	Preco	
656565	Uma janela suspeita	Drama	1001	dub	DVD	25	3,50	
656565	Uma janela suspeita	Drama	1002	dub	VHS	25	3,50	
656565	Uma janela suspeita	Drama	1003	leg	DVD	25	3,50	
656565	Uma janela suspeita	Drama	1004	leg	VHS	25	3,50	
323232	Minority Report	Ficção	2550	dub	DVD	32	4,20	
323232	Minority Report	Ficção	2550	dub	VHS	32	4,20	
323232	Minority Report	Ficção	2550	leg	DVD	32	4,20	
323232	Minority Report	Ficção	2550	leg	VHS	32	4,20	

Tabela 09: Exemplo com uma tabela não normalizada.

Bancos de dados relacionais são as tecnologias mais utilizadas para armazenamento de dados hoje em dia, de forma a possibilitar o armazenamento de um volume grande de dados. Neste cenário, normalização se apresenta como uma das principais técnicas de modelagem a serem aplicadas a fim de prover melhorias no gerenciamento de dados através da minimização da sua redundância.

As regras de normalização são projetadas para prevenir anomalias e inconsistência de dados. Neste sentido, este artigo descreve um guia para as cinco formas normais que podem ser aplicadas na estrutura de um banco de dados a fim de reduzir, principalmente, redundância nos dados e prover uma melhor estrutura para recuperação das informações em um banco de dados relacional.

FORMAS NORMAIS

As formas normais definidas na teoria de banco de dados relacional representam diretrizes para projeto de como as informações ficarão organizadas no banco. As diretrizes correspondentes da primeira à quinta formas normais são apresentadas no artigo, de forma a não exigir um entendimento da teoria relacional. As diretrizes de projeto são significativas mesmo se não estivermos usando um sistema de banco de dados relacional. As diretrizes são apresentadas sem se referir aos conceitos do modelo relacional para enfatizar sua generalidade, e também para facilitar seu entendimento.

As regras de normalização são projetadas para prevenir anomalias e inconsistência de dados. Com respeito à contrapartida no desempenho, essas diretrizes são enviesadas supondo que todos os campos que não são chaves serão atualizados frequentemente. Elas tendem a penalizar recuperações de dados, pois os dados a serem recuperados a partir de um registro em um projeto não normalizado pode ter que ser recuperado a partir de vários registros na forma normalizada. Por este motivo, não existe obrigação por normalizar completamente todos os registros quando os requisitos de desempenho da aplicação são levados em conta. A normalização é um processo a partir do qual se aplicam regras a todas as tabelas do banco de

dados com o objetivo de evitar falhas no projeto, como redundância de dados e mistura de diferentes assuntos numa mesma tabela.

Ao projetar um banco de dados, se temos um modelo de entidades e relacionamentos e a partir dele construirmos o modelo relacional seguindo as regras de transformação corretamente, o modelo relacional resultante estará, provavelmente, normalizado. Mas, nem sempre os modelos que nos deparamos são implementados dessa forma e, quando isso acontece, o suporte ao banco de dados é dificultado.

Em ambos os casos, é necessário aplicar as técnicas de normalização, ou para normalizar (segundo caso citado), ou apenas para validar o esquema criado (primeiro caso citado). Aplicando as regras descritas a seguir, é possível garantir um banco de dados mais íntegro, sem redundâncias e inconsistências.

Existem 3 formas normais mais conhecidas:

1FN - 1^a Forma Normal:

Todos os atributos de uma tabela devem ser atômicos, ou seja, a tabela não deve conter grupos repetidos e nem atributos com mais de um valor. Para deixar nesta forma normal, é preciso identificar a chave primária da tabela, identificar a(s) coluna(s) que tem(êm) dados repetidos e removê-la(s), criar uma nova tabela com a chave primária para armazenar o dado repetido e, por fim, criar uma relação entre a tabela principal e a tabela secundária. Por exemplo, considere a tabela Pessoas a seguir.

```
PESSOAS = { ID + NOME + ENDERECO + TELEFONES }
```

Ela contém a chave primária ID e o atributo TELEFONES é um atributo multivalorado e, portanto, a tabela não está na 1FN. Para deixá-la na 1FN, vamos criar uma nova tabela chamada TELEFONES que conterá PESSOA_ID como chave estrangeira de PESSOAS e TELEFONE como o valor multivalorado que será armazenado.

```
PESSOAS = { ID + NOME + ENDERECO }
```

```
TELEFONES = { PESSOA_ID + TELEFONE }
```

2FN - 2^a Forma Normal:

Para estar na 2FN é preciso estar na 1FN necessariamente. Além disso, todos os atributos não chaves da tabela devem depender totalmente da chave primária (não podendo depender apenas de parte dela). Para deixar na segunda forma normal, é preciso identificar as colunas que não são funcionalmente dependentes da chave primária da tabela e, em seguida, remover essa coluna da tabela principal e criar uma nova tabela com esses dados. Por exemplo, considere a tabela ALUNOS_CURSOS a seguir.

```
ALUNOS_CURSOS = { ID_ALUNO + ID_CURSO + NOTA + DESCRICAO_CURSO }
```

Nessa tabela, o atributo DESCRICAO_CURSO depende apenas da chave primária ID_CURSO. Dessa forma, a tabela não está na 2FN. Para tanto, cria-se uma nova tabela chamada CURSOS que tem como chave primária ID_CURSO e atributo DESCRICAO retirando, assim, o atributo DESCRICAO_CURSO da tabela ALUNOS_CURSOS.

```
ALUNOS_CURSOS = { ID_ALUNO + ID_CURSO + NOTA }
CURSOS = { ID_CURSO + DESCRICAO}
```

3FN - 3^a Forma Normal:

Para estar na 3FN, é preciso estar na 2FN. Além disso, os atributos não chave de uma tabela devem ser mutuamente independentes e dependentes unicamente e exclusivamente da chave primária (um atributo B é funcionalmente dependente de A se, e somente se, para cada valor de A só existe um valor de B). Para atingir essa forma normal, é preciso identificar as colunas que são funcionalmente dependentes das outras colunas não chave e extraí-las para outra tabela. Considere, como exemplo, a tabela FUNCIONARIOS a seguir.

```
FUNCIONARIOS = { ID + NOME + ID_CARGO + DESCRICAO_CARGO }
```

O atributo DESCRICAO_CARGO depende exclusivamente de ID_CARGO (atributo não chave) e, portanto, deve-se criar uma nova tabela com esses atributos. Dessa forma, ficamos com as seguintes tabelas:

```
FUNCIONARIOS = { ID + NOME + ID_CARGO }
```

```
CARGOS = { ID_CARGO + DESCRICAO }
```

Como exercício, normalize a tabela EMPREGADOS a seguir.

Matri cula	Nome	Cod Cargo	NomeCargo	CodProj	DataFim	Horas
120	João	1	Programador	01	17/07/95	37
120	João	1	Programador	08	12/01/96	12
121	Hélio	1	Programador	01	17/07/95	45
121	Hélio	1	Programador	08	12/01/96	21
121	Hélio	1	Programador	12	21/03/96	107
270	Gabriel	2	Analista	08	12/01/96	10
270	Gabriel	2	Analista	12	21/03/96	38
273	Silva	3	Projetista	01	17/07/95	22
274	Abraão	2	Analista	12	21/03/96	31
279	Carla	1	Programador	01	17/07/96	27
279	Carla	1	Programador	08	12/01/96	20
279	Carla	1	Programador	12	21/03/96	51
301	Ana	1	Programador	12	21/03/96	16
306	Manoel	3	Projetista	17	21/03/96	67

Tabela 10: Primeira Forma Normal 1FN - Normalização de dados.

Normalização para a Primeira Forma Normal

A primeira parte da normalização é chamada de 1FN ou primeira forma normal, em uma escala que vai até cinco. Veja o post normalização de dados e as formas normais que mostra o uso das principais formas normais.

Uma relação estará na primeira forma normal 1FN, **se não houver grupo de dados repetidos**, isto é, se todos os valores forem únicos. Em outras palavras podemos definir que a primeira forma normal não admite repetições ou campos que tenha mais que um valor.

Os procedimentos mais recomendados para aplicar a 1FN são os seguintes:

- Identificar a chave primária da entidade;
- Identificar o grupo repetitivo e removê-lo da entidade;
- Criar uma nova entidade com a chave primária da entidade anterior e o grupo repetitivo.

A chave primária da nova entidade será obtida pela concatenação da chave primária da entidade inicial e a do grupo repetitivo.

Exemplo de normalização de dados. Primeira forma normal

Considere a tabela cliente abaixo:

Cliente

- Código_cliente
- Nome
- Telefone
- Endereço

Agora a tabela com os dados:

Código_cliente	Nome	Telefone	Endereço
C001	José	9563-6352 9847-2501	Rua Seis, 85 Morumbi 12536-965
C002	Maria	3265-8596	Rua Onze, 64 Moema 65985-963
C003	Jania	8545-8956 9598-6301	Praça ramos Liberdade 68858-633

Tabela 11: Tabela desnормizada, ou seja, não está na 1ª forma normal

Desta forma, todos os clientes possuem Rua, CEP e Bairro, e essas informações estão na mesma célula da tabela, logo ela não está na primeira forma normal. Para normalizar, deveremos colocar cada informação em uma coluna diferente, como no exemplo a seguir:

Código_cliente	Nome	Telefone	Rua	Bairro	Cep
C001	José	9563-6352 9847-2501	Rua Seis, 85	Morumbi	12536-965
C002	Maria	3265-8596	Rua Onze, 64	Moema	65985-963
C003	Janio	8545-8956 9598-6301	Praça ramos	Liberdade	68858-633

Tabela 12: Tabela desnormalizada, ou seja, não está na 1ª forma normal

Mesmo com o ajuste acima, a tabela ainda não está na primeira forma normal, pois há clientes com mais de um telefone e os valores estão em uma mesma célula. Para normalizar será necessário criar uma nova tabela para armazenar os números dos telefones e o campo-chave da tabela cliente. Veja o resultado a seguir:

Código_cliente	Nome	Rua	Bairro	Cep
C001	José	Rua Seis, 85	Morumbi	12536-965
C002	Maria	Rua Onze, 64	Moema	65985-963
C003	Janio	Praça ramos	Liberdade	68858-633

Tabela 13: Tabela na primeira forma normal

Código_cliente	Telefone
C001	9563-6352
C001	9847-2501
C002	3265-8596
C003	8545-8956
C003	9598-6301

Tabela 14: Tabela na 1ª forma normal

Na segunda tabela a chave primária está implícita, isto voe poderá encontrar algumas literaturas especializadas, onde nem sempre ela é especificada, mas ela deverá existir.

No exemplo acima foi gerado uma segunda entidade para que a primeira forma normal fosse satisfeita, contudo é importante ressaltar que nem sempre encontramos banco de dados com tabelas normalizadas. Existem casos onde as repetições são poucas ou o cenário permite administrar as repetições sem trazer grandes consequências.

Quais os problemas de uma tabela não normalizada com 1FN?

Muitos. A primeira forma normal tenta resolver um dos maiores problemas de banco de dados que é repetição e a desorganização deles. Imagine um campo telefone que permita a entrada de mais de um valor (dois números de telefones), por exemplo. Isto traria problemas na busca de um dos valores, por exemplo.

Outro problema seria um campo endereço onde as partes não estejam desmembradas. Isto é, um campo que permitisse eu escrever um endereço assim:

Rua das Oliveiras, 256, Parque Novo Mundo, São Paulo, SP.

Como seria possível fazer uma busca por endereços de determinado bairro apenas ou de determinadas cidades? Veja que a normalização irá trazer inúmeros benefícios de performance do banco e claro nos possibilitaria trabalhar com esses dados da forma que fosse necessário.

Normalização para a Segunda Forma Normal

Uma tabela está na Segunda Forma Normal 2FN se ela estiver na 1FN e todos os atributos não chave forem totalmente dependentes da chave primária (dependente de toda a chave e não apenas de parte dela).

Se o nome do produto já existe na tabela produtos, então não é necessário que ele exista na tabela de produtos. A segunda forma normal trata destas anomalias e evita que valores fiquem em redundância no banco de dados.

Procedimentos:

- Identificar os atributos que não são funcionalmente dependentes de toda a chave primária;
- Remover da entidade todos esses atributos identificados e criar uma nova entidade com eles.

A chave primária da nova entidade será o atributo do qual os atributos do qual os atributos removidos são funcionalmente dependentes.

Exemplo de segunda forma normal

Considere a tabela vendas abaixo:

Vendas

- N_pedido
- Código_produto
- Produto
- Quant
- Valor_unit
- Subtotal

Agora a tabela com os dados:

N_pedido	Código_produto	Produto	Quant	Valor_unit	Subtotal
1005	1-934	Impressora laser	5	1.500,00	7.500,00
1006	1-956	Impressora desjet	3	350,00	1.050,00
1007	1-923	Impressora matricial	1	190,00	190,00
1008	1-908	Impressora mobile	6	980,00	5.880,00

Tabela 15: Tabela não está na segunda forma normal

Analisando teremos:

O nome do produto depende do código do produto, porém não depende de N_pedido que é a chave primária da tabela, portanto não está na segunda forma normal. Isto gera problemas com a manutenção dos dados, pois se houver alteração no nome do produto teremos que alterar em todos os registros da tabela venda.

Para normalizar esta tabela teremos de criar a tabela Produto que ficará com os atributos Código_produto e produto e na tabela Venda manteremos somente os atributos N_pedido, código_produto, quant, valor_unit e subtotal. Veja o resultado abaixo:

Código_produto	Produto
1-934	Impressora laser
1-956	Impressora desjet
1-923	Impressora matricial
1-908	Impressora mobile

Tabela 16: Tabela na segunda forma normal

N_pedido	Código_produto	Quant	Valor_unit	Subtotal
1005	1-934	5	1.500,00	7.500,00
1006	1-956	3	350,00	1.050,00
1007	1-923	1	190,00	190,00
1008	1-908	6	980,00	5.880,00

Tabela 17: Tabela na 2ª forma normal

Conforme visto na Primeira forma normal, quando aplicamos normalização é comum gerar novas tabelas a fim de satisfazer as formas normais que estão sendo aplicadas.

Normalização para a Terceira Forma Normal

Uma tabela está na Terceira Forma Normal 3FN se ela estiver na 2FN e se nenhuma coluna não-chave depender de outra coluna não-chave. Desta forma, é necessário eliminar aqueles campos que podem ser obtidos pela equação de outros campos da mesma tabela.

Procedimentos:

- Identificar todos os atributos que são funcionalmente dependentes de outros atributos não chave;
- Removê-los.

A chave primária da nova entidade será o atributo do qual os atributos removidos são funcionalmente dependentes.

Exemplo de normalização na terceira forma normal

Considere a tabela abaixo:

N_pedido	Codigo_produto	Quant	Valor_unit	Subtotal
1005	1-934	5	1.500,00	7.500,00
1006	1-956	3	350,00	1.050,00
1007	1-923	1	190,00	190,00
1008	1-908	6	980,00	5.880,00

Tabela 18: Tabela não está na terceira forma normal

Considerando ainda a nossa tabela Venda, veremos que a mesma não está na terceira forma normal, pois o subtotal é o resultado da multiplicação Quant X Valor_unit, desta forma a coluna subtotal depende de outras colunas não-chave.

Para normalizar esta tabela na terceira forma normal teremos de eliminar a coluna subtotal, como no exemplo a seguir:

N_pedido	Codigo_produto	Quant	Valor_unit
1005	1-934	5	1.500,00
1006	1-956	3	350,00
1007	1-923	1	190,00
1008	1-908	6	980,00

Tabela 19: Tabela na terceira forma normal

Conforme visto nos post primeira forma normal e segunda forma normal, a normalização torna o banco de dados menos redundante e consistente.

Unity 3D e Banco de Dados para Jogos

Unity

O *Unity* foi criado para auxiliar no desenvolvimento de jogos, é composto por uma coleção de ferramentas para atender fins específicos. O sistema em si, é dividido por varias **visões**, onde é possível manipular os objetos e suas propriedades. Em *Unity* é possível criar seus próprios assets ou importar de outras ferramentas.

A ferramenta utiliza o conceito de **cenas**, onde é possível criar várias para cada **jogo**, onde cada “tela”, **fase** ou **nível** de um jogo se torna uma cena. Os **objetos** então são adicionados às cenas, onde podem ser manipulados, terem propriedades específicas e até mesmo scripts.

O Unity também conta com uma opção para criar **Prefabs**, que funcionam como instâncias de cada objeto para melhorar o desenvolvimento. É possível trabalhar com as **câmeras** do jogo e a **iluminação**, deixando o efeito mais próximo da realidade.

O Unity oferece diversas formas de publicação, para diferentes plataformas, como web player, uma versão para ser jogada pela Internet, ou PC/Mac, iOS ou Android e até mesmo para consoles como Xbox e PlayStation.

É essencial que para o desenvolvimento de jogos, scripts sejam criados para controlar as ações dos personagens, manipulação dos objetos, controlar as entradas dos jogadores e até mesmo implementar Inteligência artificial.

Os scripts podem ser escritos em C# ou JavaScript, e serão interpretados pela própria ferramenta.

Unity - XML

É possível utilizar acesso a arquivos XML diretamente pelo script C# ou JavaScript no Unity. Basta utilizar referência a bibliotecas de manipulação de XML e salvar e carregar o arquivo, de forma que os dados são então carregados para variáveis ou propriedades de objetos do jogo, como posicionamento de um personagem, ou nível de vida.

Unity - SQLite

Uma outra alternativa para banco de dados, é a utilização do SQLite, que funciona como um banco de dados relacional, porém com apenas um arquivo e uma biblioteca de acesso ao arquivo. O SQLite não necessita que seja instalado ou que tenha processos, o que permite que o script C# ou JavaScript acesse de forma direta utilizando o plugin do SQLite.

A desvantagem é que não é possível publicar jogos utilizando o *WebPlayer*, uma vez que não existe um servidor para rodar o SQLite. Outra desvantagem é em relação a programação utilizando Script C#, sendo apenas viável utilizando JavaScript.

Referências

- Kevin Lindeman, *Server Side Highscores*. 2013.
- MANNINO, M V. Projeto Desenvolvimento de Aplicações e Administração de Banco de Dados. McGraw-Hill, 2008.
- MySQL 5.7 Reference Manual Oracle Corporation, 2014.
<http://dev.mysql.com/doc/refman/5.7/en/index.html>
- NAVATHE, Shamkant B.; ELMASRI, Ramez E. Sistemas de Banco de Dados. 6.ed. Addison- Wesley, 2010.
- <http://profmfpsilva.wordpress.com/> prof.mfpsilva@gmail.com Referências
- Simon Donkers e Johannes Stoop, GameMaker Manual. 2006-2007
<http://gamemaker.info/en/manual>
- Scirra All Tutorials. 2015. <https://www.scirra.com/tutorials/all>
- The PHP Group, Manual do PHP. 2001-2015. http://php.net/manual/pt_BR/
- http://wiki.unity3d.com/index.php/Server_Side_Highscores
- W3SCHOOLS, AJAX Tutorial. 1999-2015. <http://www.w3schools.com/ajax/>
- <http://www.w3schools.com/xml/default.asp>