

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/279752302>

Apostila de Banco de Dados

Book · March 2012

DOI: 10.13140/RG.2.1.4597.8725

CITATIONS

0

READS

2,040

1 author:



G.F. Cintra

Instituto Federal de Educação, Ciência e Tecnologia do Ceará

24 PUBLICATIONS **116** CITATIONS

SEE PROFILE

Banco de Dados



Glauber Ferreira Cintra
Março/2012

Aula 1 - Introdução

Objetivos da Aula

Nesta aula, apresentaremos os conceitos básicos da disciplina, começando pelas definições de banco de dados e sistema gerenciador de banco de dados. Abordaremos o conceito de abstração de dados apresentando os vários níveis de abstração.

Em seguida, faremos uma breve discussão sobre os diversos tipos de modelos de dados. Finalmente, discorreremos sobre os objetivos de um sistema de banco de dados e apresentaremos seus principais componentes.

1.1 Conceitos Básicos

Um **banco de dados (BD)** é uma coleção de dados interrelacionados. Num BD, os dados são armazenados em tabelas. As tabelas são divididas em linhas (registros) e colunas (campos). Cada registro é constituído por diversos valores, um valor para cada campo da tabela. Cada coluna é identificada por um nome.

O diagrama mostra uma tabela com três colunas: 'Código', 'Nome' e 'Sexo'. Há duas linhas de dados. À esquerda da tabela, o rótulo 'linhas' tem duas setas apontando para as duas linhas da tabela. À direita, o rótulo 'colunas' tem três setas apontando para cada uma das três colunas da tabela.

Código	Nome	Sexo
26	Ana	F
14	Rui	M

Figura 1 - Exemplo de tabela num banco de dados

Um **sistema gerenciador de banco de dados (SGDB)** é um conjunto de programas que permite manipular bancos de dados. Alguns dos SGDB's mais populares são o MySQL, o PostgreSQL, o Oracle e o SQL Server.

1.2 Abstração de Dados

É adequado omitir os detalhes complexos de um sistema de banco de dados fornecendo aos usuários uma visão abstrata dos dados. Existem três níveis de abstração:

- **Nível físico:** descreve *como* os dados são armazenados. É o nível mais baixo de abstração e interessa apenas aos projetistas de SGDB's.
- **Nível lógico:** descreve *quais* dados são armazenados e *como* eles se interrelacionam. É o nível médio de abstração e é de interesse dos projetistas de sistemas de banco de dados.
- **Nível de visão:** descreve apenas uma parte do BD e é o nível mais alto de abstração. Podem existir diversas visões diferentes para grupos de usuários distintos.

1.3 Modelos de Dados

Um *modelo de dados* é uma coleção de ferramentas conceituais que serve para descrever os dados, sua semântica, suas restrições de integridade e de consistência e a forma como eles se relacionam. Os modelos de dados podem ser classificados em:

- **Modelos físicos:** descrevem o nível físico do BD. Estruturas de dados complexas de baixo nível são descritas em detalhes. Existem poucos exemplos de modelos físicos. Os principais são o modelo unificado (*unifying model*) e o modelo de partição de memória (*frame-memory model*).
- **Modelos lógicos:** servem para descrever o nível lógico e o nível de visão do BD.

1.4 Modelos Lógicos de Dados

Os modelos lógicos de dados podem ser classificados em duas categorias:

- **Orientados a objetos:** os dados são vistos como *objetos* que se interrelacionam. Os mais conhecidos são o modelo entidade-relacionamento, o modelo orientado a objetos, o modelo semântico de dados e o modelo funcional de dados. Na aula seguinte, discutiremos o modelo entidade-relacionamento em detalhes.
- **Orientados a registros:** o BD é estruturado na forma de registros que normalmente têm tamanho fixo, o que facilita a implementação física do BD. Os principais exemplos são o modelo relacional, o modelo de rede e o modelo hierárquico. A seguir, discutimos brevemente esses três modelos.

1.4.1 Modelo Relacional

Os registros contêm *chaves* que definem os relacionamentos entre eles. Esse modelo foi proposto por *Edgar Frank Cid* nos anos 70 e de longe é o modelo de dados mais adotado pelos SGDB's atualmente.

Na figura a seguir, a tabela da esquerda contém dados dos clientes e a da direita contém dados das contas dos clientes. O relacionamento entre os clientes e as contas é determinado pela chave *Número_conta*. Observe que o cliente Rui possui duas contas, sendo uma delas conjunta com a cliente Ana.

Código	Cliente	Número_conta
26	Ana	1
14	Rui	2
15	Lia	3
14	Rui	1

Número_conta	Saldo
1	120,00
2	450,00
3	10,00

Figura 2 - Exemplo de banco de dados relacional

1.4.2 Modelo de Rede

Os dados são representados por um conjunto de registros e as relações entre os registros são representadas por ponteiros. Na figura a seguir, são representadas as mesmas informações da figura anterior seguindo o modelo de rede. Observe que não há o uso de chaves para relacioná-las e sim ponteiros.

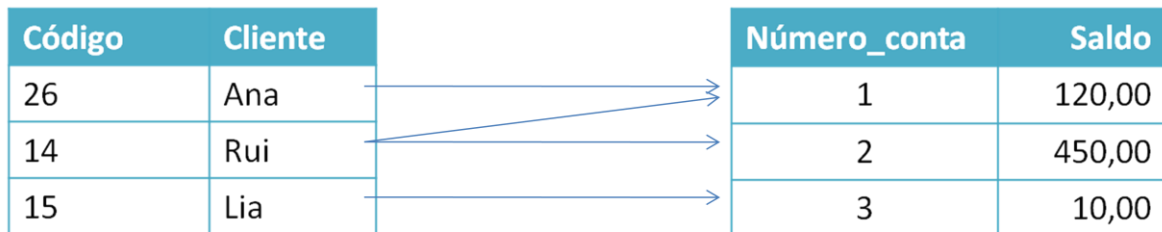


Figura 3 - Exemplo de banco de dados de rede

1.4.3 Modelo Hierárquico

É semelhante ao modelo de rede. No entanto, o relacionamento entre os registros obedece a uma hierarquia no formato de árvore. Na figura abaixo são representadas as mesmas informações da figura anterior. Note que o registro relativo à conta 1 teve que ser replicado para relacioná-la com seus titulares.

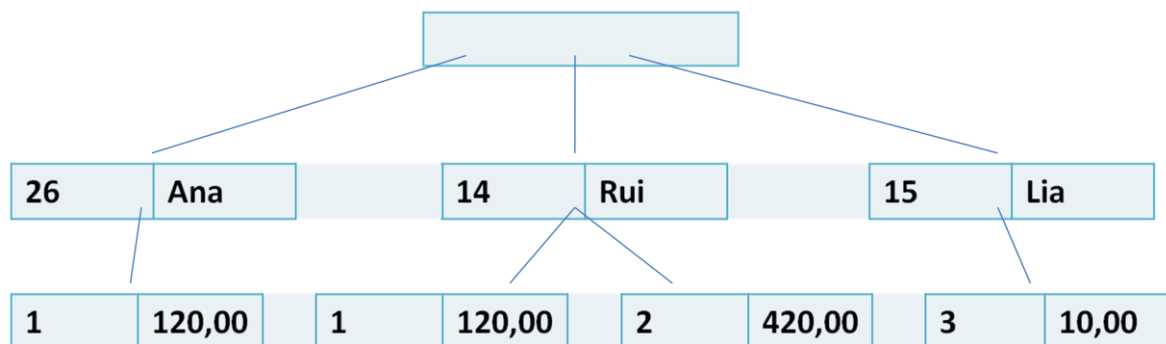


Figura 4 - Exemplo de banco de dados hierárquico

1.5 Instâncias e Esquemas

O conjunto dos dados armazenados no BD num determinado instante é chamado de *instância* do BD. O projeto geral do BD é chamado de *esquema* do BD. Enquanto que a instância do BD é modificada com muita frequência, o esquema do BD, se bem projetado, muda com pouca frequência.

Cada nível de abstração é descrito por um esquema:

- Nível físico → **esquema físico**
- Nível lógico → **esquema lógico**

- Nível de visão → **subesquemas**

1.6 Independência de Dados

A capacidade de modificar um esquema sem afetar o esquema de nível superior é chamada de *independência de dados*. Essa independência pode ser física ou lógica.

A *independência de dados física* é a capacidade de alterar o esquema físico sem que seja necessário reescrever os *programas de aplicação*. Esses programas, juntamente com o SGDB, permitem aos usuários manipular o banco de dados. Os programas de aplicação fornecem interfaces para inserir, remover, alterar e consultar as informações do BD. Além disso, permitem também alterar a estrutura do BD.

A *independência de dados lógica* é a capacidade de alterar o esquema lógico sem que seja necessário reescrever os programas de aplicação. É mais difícil de ser alcançada do que a independência física.

1.7 Objetivos de Sistemas de Banco de Dados

Um sistema de banco de dados é projetado para manipular grandes volumes de informação. Seu principal objetivo é permitir o armazenamento de dados proporcionando aos usuários uma visão abstrata desses dados. Outros objetivos de um sistema de BD são:

- **Facilitar a consulta aos dados:** isso é obtido através do uso de programas de aplicação específicos para a realização de consultas e geração de relatórios. O uso de uma *linguagem de banco de dados* facilita o desenvolvimento de tais programas.
- **Garantir a segurança e integridade dos dados:** as modificações na instância do BD não devem violar as restrições de integridade dos dados. É preciso restringir o acesso ao BD e aos usuários autorizados. Além de prover mecanismos para a consistência do BD, no caso de ocorrências de falhas.

1.8 Linguagens de Banco de Dados

As linguagens de banco de dados são utilizadas para expressar comandos que servem para manipular o BD. Algumas das linguagens de BD mais comuns são: SQL, QUEL, QBE e Datalog.

Uma linguagem de BD pode ser classificada como:

- **Linguagem de definição de dados** (data definition language – DDL): permite definir a estrutura do BD, ou seja, criar tabelas, índices, visões etc.
- **Linguagem de manipulação de dados** (data manipulation language – DML): permite inserir, alterar e remover informação do BD.
- **Linguagem de consulta** (query language): permite realizar consultas de modo a extrair informações do BD.

1.8.1 Linguagens de Consulta

Uma linguagem de consulta pode ser:

- **Procedimental:** requer a especificação de *quais* dados devem ser consultados e *como* chegar até eles.
- **Não procedimental:** requer apenas a especificação de *quais* dados devem ser consultados.

Como veremos mais adiante, a linguagem SQL além de ser uma linguagem de consulta é também uma DDL e uma DML. Tal fato explica, em parte, seu grande sucesso comercial.

1.9 Segurança dos Dados

De modo a garantir a segurança dos dados, um sistema de BD deve possuir os seguintes recursos:

- **Controle de acesso de usuário:** os usuários devem ser cadastrados e suas prerrogativas no sistema devem ser definidas pelo administrador do banco de dados (DBA).
- **Criptografia:** as senhas dos usuários e os dados armazenados nas tabelas devem ser criptografados. Isso evita que usuários não autorizados consigam visualizar os dados. Além disso, as tabelas podem possuir uma *assinatura digital*. Eventuais alterações no BD, realizadas por usuários não autorizados, serão detectadas pelo sistema de banco de dados que cuidará para que a instância do BD retorne a um estado válido.

1.10 Integridade dos Dados

O sistema de BD deve garantir que as restrições de integridade dos dados sejam respeitadas. Além disso, um sistema de BD deve possuir os mecanismos para:

- **Recuperação de falhas:** após a ocorrência de uma falha de hardware ou de software, o sistema de BD deve realizar o procedimento para fazer com que o BD retorne a um estado consistente. As técnicas mais comumente utilizadas envolvem o uso de arquivos de *log* (onde ficam registradas as alterações realizadas no BD) e o uso de arquivos de *backup* (onde ficam armazenados estados válidos da instância do BD).
- **Controle de acesso concorrente:** quando múltiplos usuários acessam o BD ao mesmo tempo é preciso utilizar técnicas que evitem o surgimento de inconsistências no BD. As principais técnicas de controle de concorrência são baseadas em protocolos de bloqueio e protocolos de registros de tempo.

1.11 Componentes de um Sistema de BD

Um sistema de BD típico é constituído dos seguintes componentes:

- **Arquivos:** de dados, de índices, de estatísticas, dicionários de dados.
- **Software:** SGDB, programas de aplicação e sistema de arquivos (módulo do sistema operacional).
- **Hardware:** computadores, dispositivos de armazenamento secundário, dispositivo de entrada/saída e conexões.
- **Usuários:** projetistas, programadores, administradores do banco de dados, usuários sofisticados e usuários navegantes.
- **Linguagens de BD:** DDL, DML e linguagem de consulta.

A figura a seguir ilustra esses componentes e como eles se relacionam. Na figura aparecem os diversos tipos de usuários e os recursos que eles utilizam para interagir com o SGDB. Observe que o SGDB provê uma interface entre os usuários e o banco de dados. Os módulos que compõem o processador de consultas e o gerenciador de memória, que são os componentes principais de um SGDB, também são ilustrados na figura.

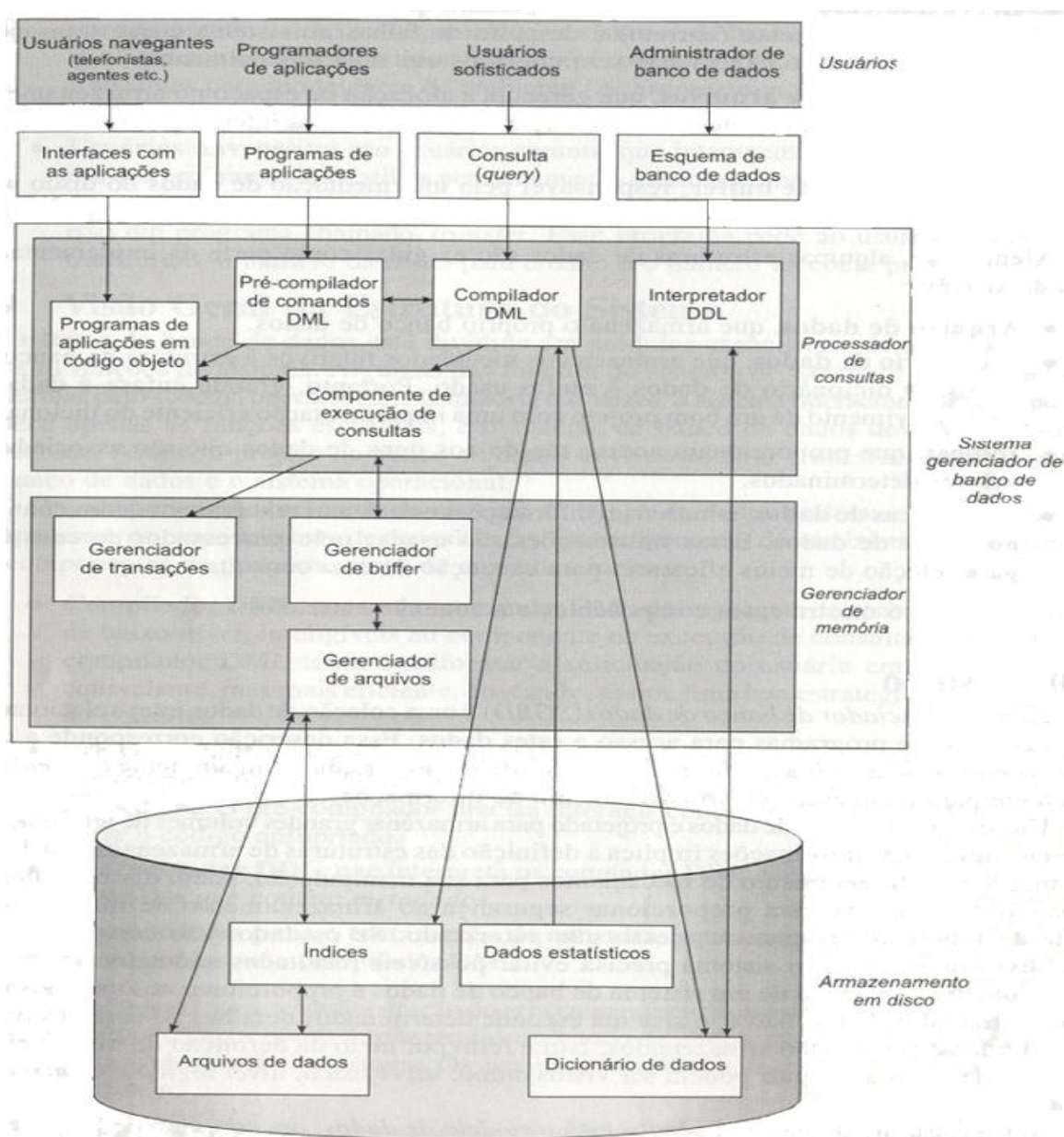


Figura 5 - Componentes de um Sistema de BD

(Figura tirada do livro *Sistemas de Banco de Dados* de Korth et al.)

Aula 2 - Modelo Entidade-Relacionamento

Objetivos da Aula

Nesta aula discutiremos o *Modelo Entidade-Relacionamento*, conforme proposto por Peter Chen. Apresentaremos seus principais elementos que são as entidades, os atributos e os relacionamentos. Explicaremos também o que são chaves. Em seguida, apresentaremos os elementos usados para construir diagramas entidade-relacionamento.

Introdução

O **modelo entidade-relacionamento (MER)** é um modelo lógico de dados orientado a objetos. Ele foi proposto em 1976 por Peter Chen e baseia-se na concepção de que o mundo é formado por *entidades* que se relacionam. Tais entidades possuem *atributos*.

No modelo são representadas as entidades, seus atributos e os relacionamentos entre as entidades. O modelo pode ser expresso graficamente usando *diagramas entidade-relacionamento*. Nas próximas seções abordaremos em detalhes os elementos que compõem o MER.

2.1 Entidades

Uma **entidade** é *algo do mundo* que se deseja modelar. Por exemplo, numa universidade os *alunos* e os *cursos* podem ser vistos como entidades. Observe que um aluno é uma entidade de natureza concreta enquanto que um curso possui natureza abstrata.



Aluno Rui



Figura 1 – Exemplos de entidades.

2.2 Atributos

As entidades são representadas pelos valores de seus atributos. Um **atributo** é uma característica de uma entidade. Por exemplo, a entidade *pessoa* pode ter os atributos *nome*, *sexo*, *data de nascimento* e *endereço*.

A escolha dos atributos que serão utilizados para descrever uma entidade depende do mundo que se está modelando. Por exemplo, em empresas e em escolas temos entidades do tipo *pessoa*. Ao descrever uma pessoa como funcionário de uma empresa é importante incluir na descrição o atributo *salário*. Se queremos descrever a pessoa como um aluno de uma escola, o atributo salário torna-se desnecessário.

2.2.1 Atributos Simples e Compostos

Podemos classificar um atributo como sendo **simples** ou **composto**.

- Atributo simples: seu valor é indivisível. Por exemplo, os atributos *sexo* e *saldo da conta* são simples.
- Atributo composto: seu valor pode ser decomposto em partes. Por exemplo, dependendo da aplicação que se deseja desenvolver, o *endereço* pode ser decomposto em tipo do logradouro, nome do logradouro, número, complemento, bairro, CEP, cidade, estado e país.

2.2.2 Atributos Monovalorados e Multivalorados

Os atributos também podem ser classificados como sendo **monovalorados** ou **multivalorados**.

- Atributo monovalorado: possui apenas um valor num dado instante. Por exemplo, o *nome* de uma pessoa é monovalorado, pois uma pessoa não pode ter dois nomes ao mesmo tempo.
- Atributo multivalorado: pode ter diversos valores ao mesmo tempo. Por exemplo, o *apelido* de uma pessoa é multivalorado, pois uma pessoa pode ter vários apelidos ao mesmo tempo.

Observe que, dependendo da aplicação, o atributo *telefone* pode ser monovalorado ou multivalorado. Se for suficiente conhecer apenas um telefone da entidade, o atributo será monovalorado. Se precisarmos conhecer vários telefones da entidade (residencial, comercial, celular etc), o atributo será multivalorado.

2.2.3 Atributos Requeridos e Valores Nulos

Um atributo é **requerido** se o seu valor não pode ser nulo, ou seja, não pode ser omitido. Por exemplo, o *nome do aluno*, em geral, é requerido, pois não faz sentido cadastrar um aluno sem nome.

Quando um atributo não é requerido, seu valor para uma determinada entidade pode ser *nulo*. Isso pode ocorrer quando a informação está indisponível. Por exemplo, um aluno pode ter valor indisponível para o atributo *CPF*, se esse aluno ainda não tiver tirado seu CPF. Nesse caso, o valor do CPF desse aluno será nulo.

Outra situação que pode levar ao surgimento de valores nulos ocorre quando o a informação não se aplica àquela entidade. Por exemplo, atributo número do certificado de reservista não se aplica às alunas. Nesse caso, pode ser adequado reformular o modelo de dados. Como veremos mais adiante, a introdução de valores nulos no banco de dados pode trazer consequências indesejáveis e por isso deve ser evitada.

2.2.4 Atributos Derivados

Um atributo é **derivado** se o seu valor pode ser obtido a partir de outros dados contidos no BD. Por exemplo, se já tivermos o atributo *data de nascimento*, o atributo *idade* é derivado, pois seu valor pode ser obtido a partir da data de nascimento. Nesse caso, não vale a pena ter o atributo idade.

Vejamos outro exemplo. Num banco é preciso armazenar dados de todas as operações (saques, depósitos, transferências etc.) que ocorrem nas contas dos clientes. Com essas informações é possível calcular o saldo da conta de qualquer cliente somando o valor das operações de crédito e subtraindo o valor das operações de débito naquela conta. No entanto, esse cálculo pode ser bem demorado. Por esse motivo, para melhorar o desempenho do BD, vale a pena incluir o atributo derivado *saldo da conta*.

2.3 Relacionamentos

Um **relacionamento** é uma associação entre entidades. Por exemplo, se o aluno *Rui* faz o curso de *Contabilidade* então existe uma associação entre eles. Nesse exemplo, o relacionamento é *binário*, pois dele participam duas entidades. Embora os relacionamentos binários sejam os mais comuns, podemos ter relacionamentos ternários, quaternários etc. Na figura abaixo temos relacionamentos entre os alunos e os cursos em que estão matriculados.

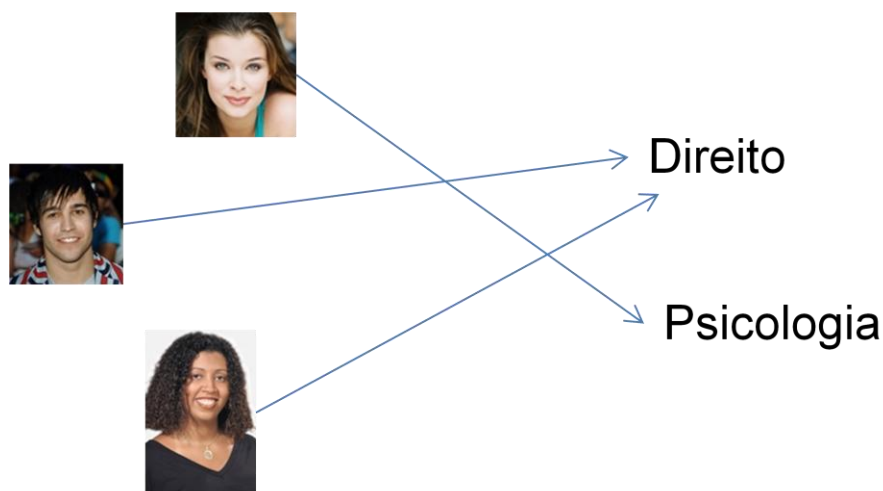


Figura 2 – Exemplo de relacionamento.

2.3.1 Autorrelacionamentos

Um **conjunto de entidades** (CE) reúne entidades de mesmo tipo, ou seja, entidade que possuem os mesmos atributos e participam dos mesmos relacionamentos.

Em geral, relacionamentos servem para associar entidades de um CE a entidades de outro CE. Se um relacionamento associa entidades de um CE a entidades do mesmo CE, dizemos que ele é um *autorrelacionamento*. Por exemplo, o relacionamento de chefia entre os funcionários de uma empresa, ilustrado abaixo, é um autorrelacionamento, pois cada relacionamento de chefia associa um funcionário subordinado a um funcionário chefe.

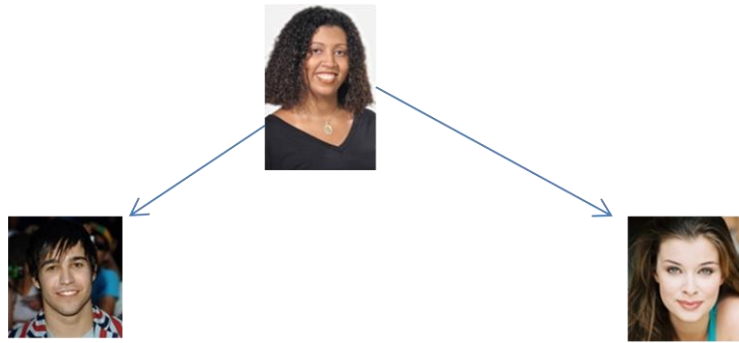


Figura 3 – Exemplo de autorrelacionamento.

2.3.2 Atributos dos Relacionamentos

Assim como as entidades, um relacionamento também pode ter atributos. Eles são chamados de *atributos descritivos*. Por exemplo, os departamentos de uma empresa têm um chefe, que é um funcionário da empresa. Esse relacionamento de chefia pode ter o atributo descritivo *data da nomeação* que indica quando o chefe de um departamento foi nomeado para esse cargo.

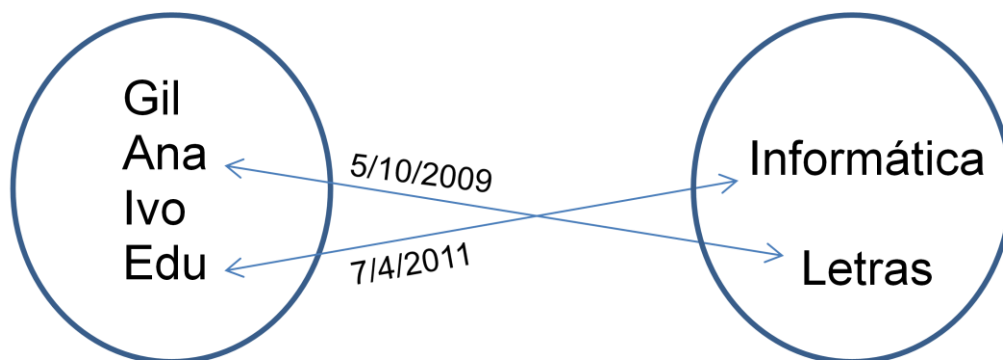


Figura 4 – Exemplo de atributo descritivo.

2.4 Cardinalidade dos Relacionamentos

A **cardinalidade** de um relacionamento binário indica quantas entidades que podem estar relacionadas com outra entidade. A cardinalidade pode ser de três tipos:

- Um para um
- Um para muitos
- Muitos para muitos

2.4.1 Relacionamentos um para um

A cardinalidade de um relacionamento de um conjunto de entidades A para um conjunto de entidades B é do tipo **um para um** se cada entidade de A está associada a no máximo uma entidade de B e cada entidade de B está associada a no máximo uma entidade de A.

Na figura a seguir temos à esquerda o CE *professores* e à direita o CE *departamentos*. As setas indicam relacionamentos de chefia. Observe que cada professor pode chefiar no máximo um departamento.

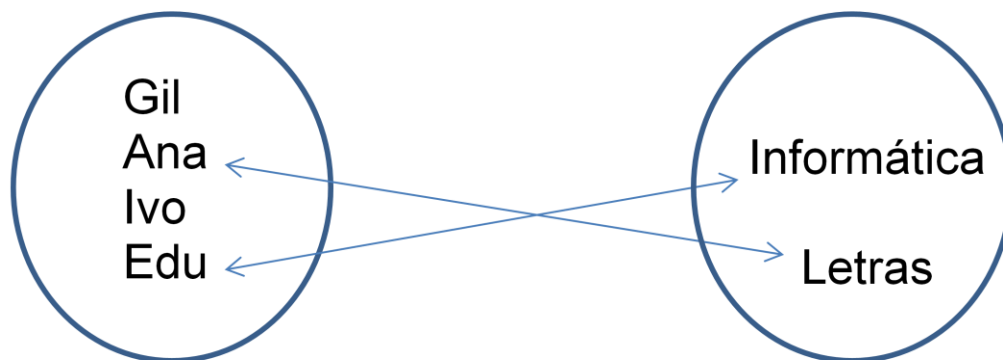


Figura 5 – Relacionamento um para um.

2.4.2 Relacionamentos um para muitos

A cardinalidade de um relacionamento de um conjunto de entidades A para um conjunto de entidades B é do tipo **um para muitos** se cada entidade de A pode estar associada a várias entidades de B, mas cada entidade de B está associada a no máximo uma entidade de A.

Na figura abaixo temos à esquerda o CE *departamentos* e à direita o CE *professores*. As setas indicam que um professor pertence a um departamento. Observe que cada professor pode pertencer a no máximo um departamento, mas um departamento pode ter muitos professores.

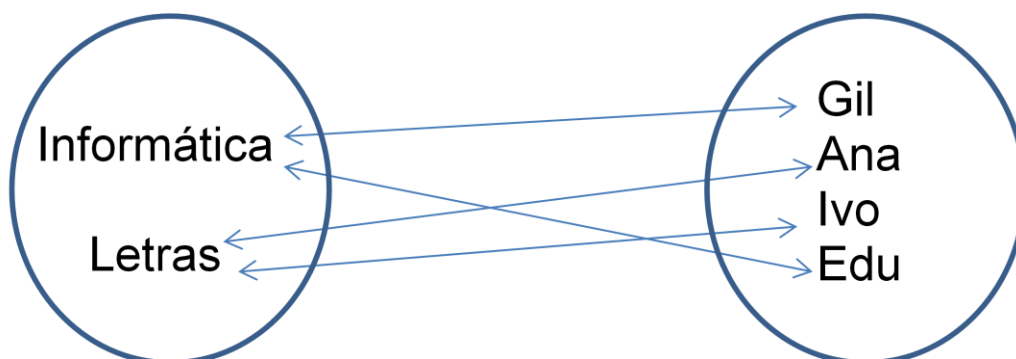


Figura 6 – Relacionamento um para muitos.

2.4.3 Relacionamentos muitos para muitos

A cardinalidade de um relacionamento de um conjunto de entidades A para um conjunto de entidades B é do tipo **muitos para muitos** se cada entidade de A pode estar associada a várias entidades de B e cada entidade de B pode estar associada a várias entidades de A.

Na figura abaixo temos à esquerda o CE *disciplinas* e à direita o CE *professores*. As setas indicam que um professor leciona uma disciplina. Note que o professor *Gil* leciona duas disciplinas enquanto que o professor *Ivo* não leciona nenhuma disciplina. Observe ainda que a disciplina *Inglês* é ministrada por dois professores.

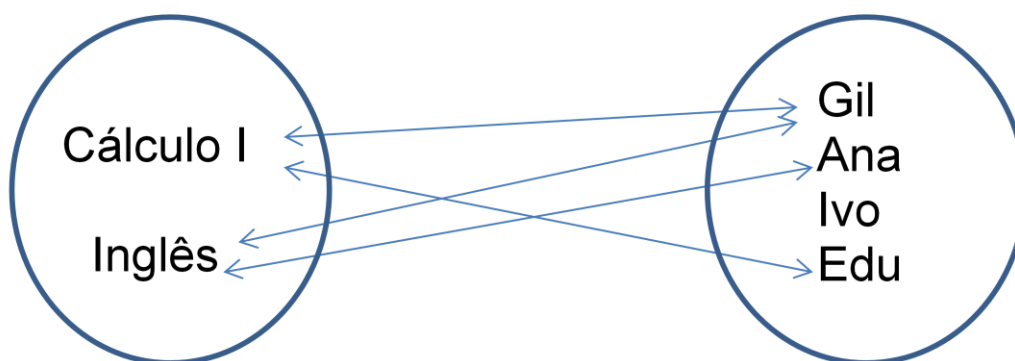


Figura 7 – Relacionamento muitos para muitos.

2.5 Relacionamentos totais e parciais

Um relacionamento é *total* se todas as entidades do CE participam do relacionamento. Se nem todas as entidades participam do relacionamento, dizemos que ele é *parcial*.

Na figura a seguir temos um relacionamento que é total à esquerda, pois toda disciplina tem um professor, mas é parcial à direita, pois há um professor que não leciona nenhuma disciplina (ele pode estar de licença).

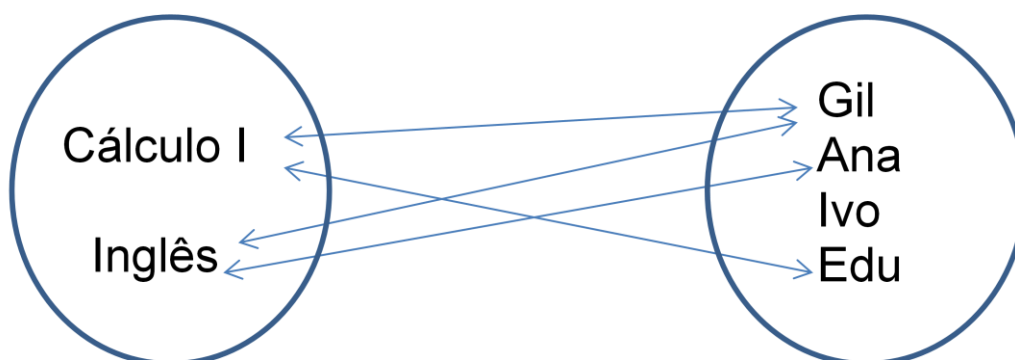


Figura 8 – Relacionamento muitos para muitos.

2.6 Esquemas e Superchaves

Como vimos anteriormente, as entidades são descritas através de seus atributos. O conjunto dos atributos que descrevem as entidades de um CE é chamado de **esquema** do CE.

Uma **superchave** é um subconjunto do esquema de um CE que permite distinguir as entidades umas das outras. Em outras palavras, não pode haver duas entidades distintas que possuam os mesmos valores em todos os atributos que compõem uma superchave. Por exemplo, na tabela abaixo, cujo esquema é constituído dos atributos *código*, *nome*, *sexo* e *CPF*, o conjunto de atributos {*sexo*, *CPF*} constitui uma superchave da tabela *Clientes*.

código	nome	sexo	CPF
26	Ana	F	00587498742
14	Lia	F	29478412635

Figura 9 – Tabela *Clientes*.

2.7 Chaves Candidatas

Voltando ao exemplo anterior, percebemos que bastaria o atributo CPF para distinguir um cliente dos demais. Uma superchave é uma **chave candidata** se todos os atributos que a compõem são necessários para que ela seja superchave. A superchave {*sexo*, *CPF*} não é uma chave candidata. No entanto, {*CPF*} é chave candidata. Como ela é constituída de um único atributo, dizemos que ela é uma chave *simples*.

Na Tabela *Matrículas* abaixo temos apenas uma chave candidata que é {*cod_aluno*, *disciplina*}. Observe que tal chave candidata é constituída de dois atributos. Dizemos que ela é uma chave *composta*.

cod_aluno	disciplina	média final
26	Cálculo 1	10
14	Cálculo 1	8,5
26	Inglês	8,5

Figura 10 – Tabela Matrículas.

2.8 Chaves Primárias

As chaves candidatas têm esse nome porque uma delas pode ser escolhida para ser a chave primária do CE. Uma **chave primária** é a chave candidata que foi escolhida pelo projetista do BD para distinguir as entidades de um CE. É importante ressaltar que cada CE pode ter apenas uma chave primária e que os valores de uma chave primária nunca podem ser *nulos*.

Como é possível distinguir um cliente dos outros clientes através de seu *CPF* ou de seu código, as superchaves {*CPF*} e {*código*} são chaves candidatas da Tabela *Clientes*. Poderíamos escolher {*código*} como chave primária.

código	nome	sexo	CPF
26	Ana	F	00587498742
14	Lia	F	29478412635

Figura 11 – Tabela *Clientes*.

2.9 Entidades Fracas

Um CE que não tenha chave candidata é chamado de **conjunto de entidades fracas**. Por exemplo, a figura abaixo apresenta a tabela *Alunos*. Observe que utilizando apenas os atributos *nome*, *sexo* e *nascimento* para descrever os alunos, embora seja improvável, podemos ter alunos distintos que possuem exatamente os mesmos valores nesses atributos. O CE *Alunos* é um conjunto de entidades fracas. O que normalmente fazemos nesses casos é criar uma chave primária artificial (*surrogate key*).

nome	sexo	nascimento
Ana	F	13/05/1994
Rui	M	15/10/1992
Ana	F	13/05/1994

Alunas distintas

Figura 12 – Tabela *Alunos*.

2.10 Diagramas Entidade-Relacionamento

Podemos expressar um modelo entidade-relacionamento graficamente utilizando um **diagrama entidade-relacionamento (DER)**. Os principais elementos que podem ser utilizados nesses diagramas são:



Retângulo: representa um CE.



Losango: representa um conjunto de relacionamentos.



Elipse: representa um atributo.

Cada CE e cada atributo tem que ter um nome (rótulo). Os conjuntos de relacionamentos podem ter um nome (normalmente esse nome expressa uma ação).

2.10.1 Cardinalidades

Podemos expressar a cardinalidade de um relacionamento de várias maneiras. Adotaremos a notação exemplificada na figura abaixo. Esse diagrama indica que cada conta corrente pertence a exatamente uma agência e que cada agência pode ter zero ou mais contas correntes.

Trata-se, portanto, de um relacionamento do tipo *um para muitos*. Observe que ele é total à esquerda, pois toda conta corrente está relacionada com alguma agência, mas parcial à direita, pois pode existir uma agência que não possua nenhuma conta corrente (a agência pode estar inativa).



Figura 13 – Cardinalidade de um relacionamento.

Podemos utilizar também os símbolos abaixo para construir diagramas entidade–relacionamento.



Retângulo de borda dupla: conjunto de entidades fracas.



Losango de borda dupla: conjunto de relacionamentos envolvendo entidades fracas.



Elipse de borda dupla: atributo multivalorado.

2.10.2 Chaves primárias e papéis

Os atributos que compõem uma chave primária devem ser sublinhados. Ao expressar um autorrelacionamento podemos indicar o *papel* que cada entidade desempenha no relacionamento. Na figura a seguir temos o autorrelacionamento de *chefia*. Note que em cada relacionamento temos um funcionário *chefe* e um funcionário *subordinado*. Observe ainda que cada chefe pode ter vários subordinados, mas cada subordinado tem apenas um chefe.

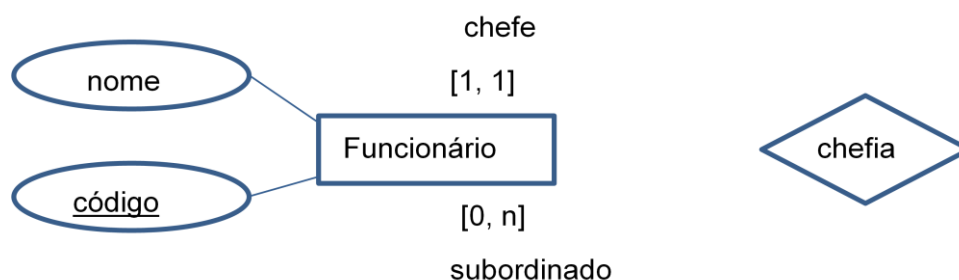


Figura 14 – Papéis em um autorrelacionamento.

Conclusões

Nesta aula apresentamos o modelo entidade-relacionamento (MER). Discutimos seus principais elementos que são as entidades, os atributos e os relacionamentos. Vimos que as entidades são representadas através dos seus atributos. Os atributos podem ser simples ou compostos, monovalorados ou multivalorados, requeridos ou derivados.

Vimos que um relacionamento representa uma associação entre entidades. A cardinalidade de um relacionamento pode ser do tipo *um para um*, *um para muitos* ou *muitos para muitos*. Um relacionamento pode ser total ou parcial. Explicamos também o que são autorrelacionamentos e atributos descritivos. Apresentamos os conceitos de superchave, chave candidata e chave primária.

Finalmente, vimos que podemos expressar graficamente um MER através de diagramas entidade-relacionamento (DER). Apresentamos os símbolos que podem ser utilizados para construir um DER e mostramos como indicar a cardinalidade dos relacionamentos, a chave primária de cada CE e os papéis que as entidades podem ter num autorrelacionamento.

Aula 3 – Modelagem de Dados e Mapeamento

Objetivos da Aula

Nesta aula, através de estudos de caso, mostraremos como produzir um modelo de dados baseado no modelo entidade-relacionamento e como expressar esse modelo através de um diagrama entidade-relacionamento (DER). Veremos também como mapear um DER para tabelas.

3.1 Modelagem de Dados

Modelagem de dados é um processo pelo qual capturamos elementos de mundo real que são relevantes para um determinado sistema de banco de dados e descrevemos tais elementos e seus relacionamentos.

Para fazer modelagem utilizando as ideias do MER devemos identificar as entidades e determinar seus atributos e seus relacionamentos. Esse processo não é mecânico. Exige raciocínio e um elevado grau de compreensão da aplicação de banco de dados que desejamos desenvolver e do mundo relacionado com essa aplicação bem como das suas regras de negócio. Essa compreensão pode ser obtida através de entrevistas com pessoas que conhecem o mundo a ser modelado e suas regras de negócio.

A seguir faremos dois estudos de caso a partir da descrição do mundo e suas regras de negócio. Como esses exemplos têm apenas fins didáticos, as descrições que apresentaremos são severas simplificações do mundo real.

3.2 Caso Restaurante

Um restaurante deseja armazenar no seu banco de dados informações sobre os pratos que serve e sobre os ingredientes usados para prepará-los. Para cada prato é preciso armazenar seu nome, seu preço e o tempo de preparo. Para cada ingrediente, é preciso armazenar seu nome e sua unidade de medida. Queremos saber também, para cada prato, quais ingredientes são usados para prepará-lo e em quais quantidades.

Primeiramente vamos analisar a descrição do caso e identificar as entidades, os atributos e os relacionamentos. No parágrafo abaixo destacamos em **negrito** as entidades, em *itálico* os atributos e sublinhado o relacionamento.

Um restaurante deseja armazenar no seu banco de dados informações sobre os **pratos** que serve e sobre os **ingredientes** usados para prepará-los. Para cada prato é preciso armazenar seu *nome*, seu *preço* e o *tempo de preparo*. Para cada ingrediente, é preciso armazenar seu *nome* e sua *unidade de medida*. Queremos saber também, para cada prato, quais ingredientes são usados para prepará-lo e em quais *quantidades*.

Nem sempre é fácil identificar as entidades, os atributos e os relacionamentos, mas neste caso obtivemos:

Entidades: prato e ingrediente.

Atributos: nome do prato, preço, nome do ingrediente, unidade de medida, quantidade.

Note que esse último atributo não pertence nem ao prato nem ao ingrediente, mas é um atributo descritivo do relacionamento. É conveniente criar chaves primárias para as entidades prato (cod_prato) e ingrediente (cod_ingrediente).

Relacionamento: utiliza.

Como num prato são utilizados vários ingredientes e um ingrediente pode ser utilizado em vários pratos, esse relacionamento é do tipo *muitos para muitos*. Observe ainda que todo prato utiliza pelo menos um ingrediente. Dessa forma, o relacionamento *utiliza* é total à esquerda.

Chegamos então ao seguinte diagrama entidade-relacionamento:

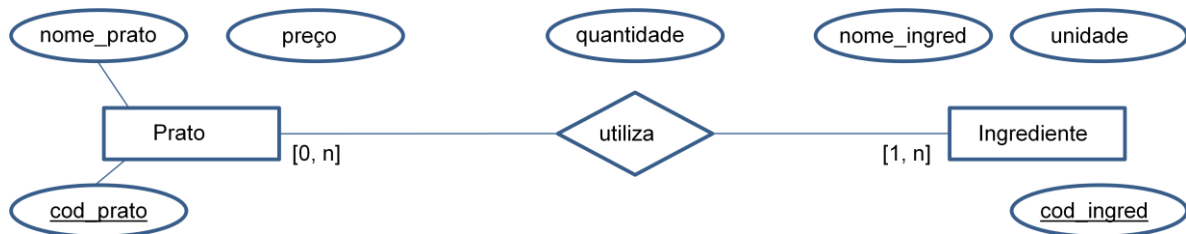


Figura 1 – DER do Caso Restaurante.

3.3 Caso Empresa

Numa empresa cada funcionário está lotado em um departamento, que é chefiado por um funcionário. Desejamos saber o nome do departamento, qual seu chefe e a data da nomeação do chefe. Os funcionários trabalham em diversos projetos, mas cada projeto está vinculado a um único departamento. Precisamos guardar o nome do projeto, saber a qual departamento ele está vinculado e indicar quais os funcionários que trabalham no projeto. Cada funcionário tem um chefe e pode ter vários dependentes. É preciso guardar o nome do funcionário, seus telefones, indicar quem é o seu chefe e quais são os seus dependentes. Cada dependente tem nome e data de nascimento.

Este caso é um pouco mais complicado. Vamos identificar na descrição do caso as entidades, os atributos e os relacionamentos, da mesma forma que fizemos no Caso Restaurante.

Numa empresa cada **funcionário** está lotado em um **departamento**, que é chefeado por um funcionário. Desejamos saber o *nome do departamento*, qual seu chefe e a *data da nomeação* do chefe. Os funcionários trabalham em diversos **projetos**, mas cada projeto está vinculado a um único departamento. Precisamos guardar o *nome do projeto*, saber a qual departamento ele está vinculado e indicar quais os funcionários que trabalham no projeto. Cada funcionário tem um chefe e pode ter vários dependentes. É preciso guardar o *nome do funcionário*, seus *telefones*, indicar quem é o seu chefe e quais são os seus **dependentes**. Cada dependente tem *nome* e *data de nascimento*.

Entidades: funcionário, departamento, projeto, dependente.

Atributos: nome do funcionário, telefone, nome do departamento, data da nomeação, nome do projeto, nome do dependente e data de nascimento.

Note que o atributo telefone é multivalorado, pois um funcionário pode ter vários telefones. Observe ainda que a data da nomeação do chefe do departamento é um atributo descritivo.

Vamos criar chaves primárias para as entidades funcionário, departamento e projeto. Propositamente, vamos deixar dependente sem chave primária. Nesse caso, dependente será uma *entidade fraca*. Essa decisão de projeto está vinculada ao fato de que a existência de um dependente no sistema só se justifica se houver um funcionário ao qual ele está associado. Dizemos, então, que existe uma relação de *dependência existencial* entre dependente e funcionário.

Relacionamentos: lotado, chefiado, trabalha, vinculado, tem chefe, tem dependente.

Os relacionamentos *lotado*, *vinculado*, *tem chefe* e *tem dependente* são do tipo *um para muitos*, pois cada funcionário está lotado em um departamento, mas um departamento pode ter muitos funcionários, cada projeto está vinculado a um departamento, mas cada departamento pode ter vários projetos vinculados a ele, cada funcionário tem um chefe, mas cada chefe pode ter vários subordinados; e cada dependente está associado a um funcionário, mas cada funcionário pode ter vários dependentes. Observe que *tem chefe* é um *autorrelacionamento*.

O relacionamento *chefiado* é do tipo *um para um*, pois um departamento é chefiado por um funcionário e um funcionário pode chefiar um departamento. Já o relacionamento *trabalha* é do tipo *muitos para muitos*, pois um funcionário pode trabalhar em muitos projetos e num projeto podem estar trabalhando muitos funcionários.

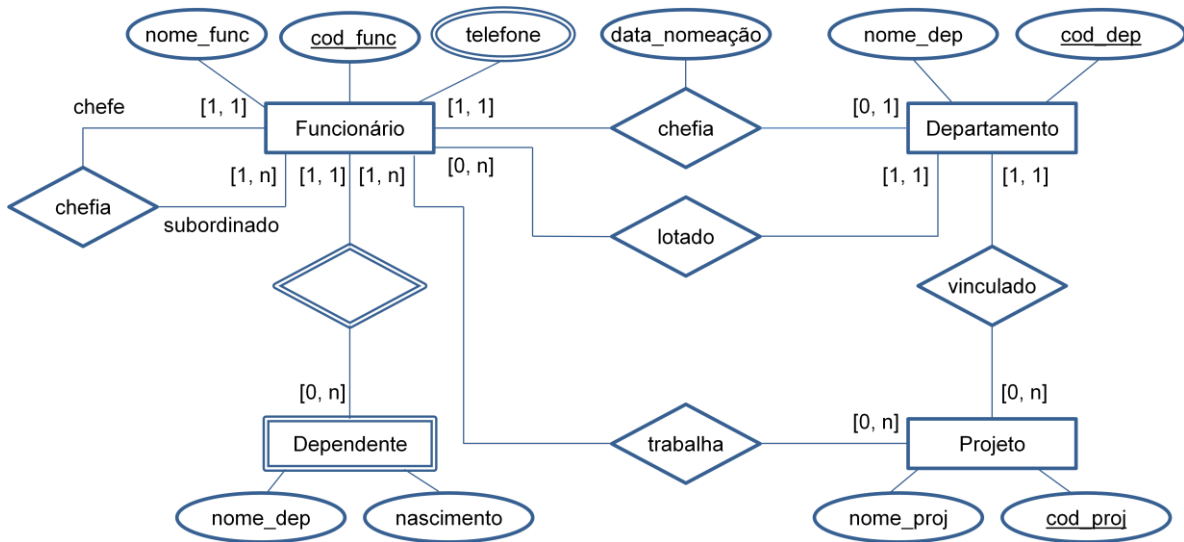


Figura 2 – DER do Caso Empresa.

3.4 Mapeamento para Tabelas

Ao contrário da modelagem de dados, o mapeamento de um DER para tabelas é um processo essencialmente mecânico. De fato, alguns programas fazem o mapeamento de forma automática. O projetista do sistema de banco de dados constrói o diagrama e então o programa faz o mapeamento para tabelas.

Tais programas são chamados de ferramentas CASE (*Computer-Aided Software Engineering*) e podemos citar como exemplos o [ERwin](#), o [System Architect](#) e o [Power Design](#).

O mapeamento para tabelas é feito seguindo-se um conjunto de regras que determina como representar através de tabelas as entidades, os atributos e os relacionamentos contidos no DER. Nas próximas seções apresentamos as principais regras de mapeamento.

3.5 Regras de Mapeamento

Regra 1: Para cada CE contido no DER, criamos uma tabela cujos campos são os atributos monovalorados do CE. No exemplo abaixo, o CE *Ingrediente* deu origem à tabela *Ingrediente*.

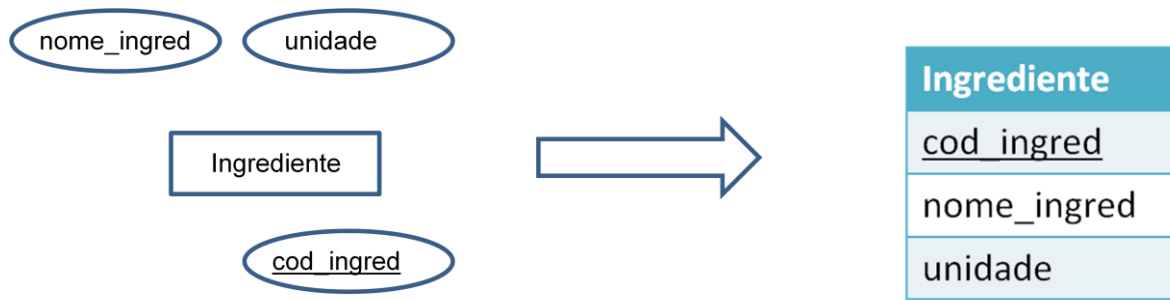


Figura 3: Mapeamento utilizando a regra 1.

Regra 2: Para cada atributo multivalorado a contido num CE cuja chave primária é constituída dos atributos a_1, a_2, \dots, a_n criamos uma tabela cujos campos são a, a_1, a_2, \dots, a_n . Observe que, no exemplo abaixo, o atributo *telefone* deu origem à tabela *Telefone* cujos atributos são *telefone* (monovalorado) e *cod_func*. Na tabela *Telefone*, o atributo *cod_func* é uma *chave estrangeira* (destacamos esse fato colocando o atributo em *itálico*). Uma *chave estrangeira* é um conjunto de atributos de uma tabela que correspondente a uma chave primária de outra tabela.



Figura 4: Mapeamento utilizando a regra 2.

Regra 3: Para cada conjunto de relacionamentos (CR) do tipo *muitos para muitos* criamos uma tabela cujos campos são os atributos descritivos do relacionamento e os atributos que compõem as chaves primárias dos CE envolvidos no relacionamento. No exemplo abaixo, o relacionamento *utiliza* deu origem à tabela que, na ausência de um nome melhor, chamamos de *Composição*.

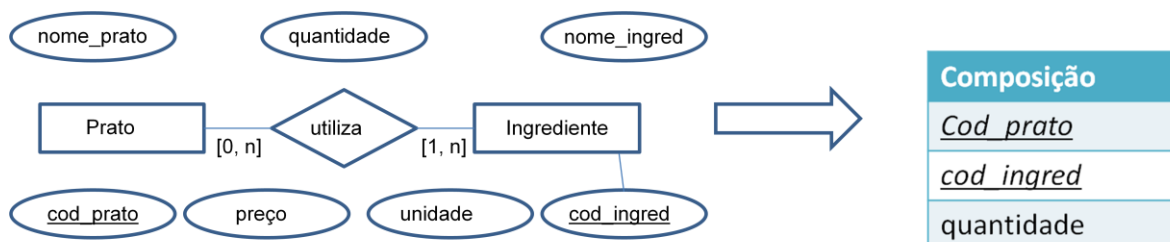


Figura 5: Mapeamento utilizando a regra 3.

Regra 4: Para cada CR do tipo *um para muitos* incluímos na tabela correspondente ao lado *muitos* os atributos que compõem a chave primária do lado *um* e os atributos descritivos do relacionamento. No exemplo abaixo, a chave primária do *Funcionário* é exportada para a tabela *Dependente*. Na tabela *Dependente*, *cod_func* é uma *chave estrangeira*.

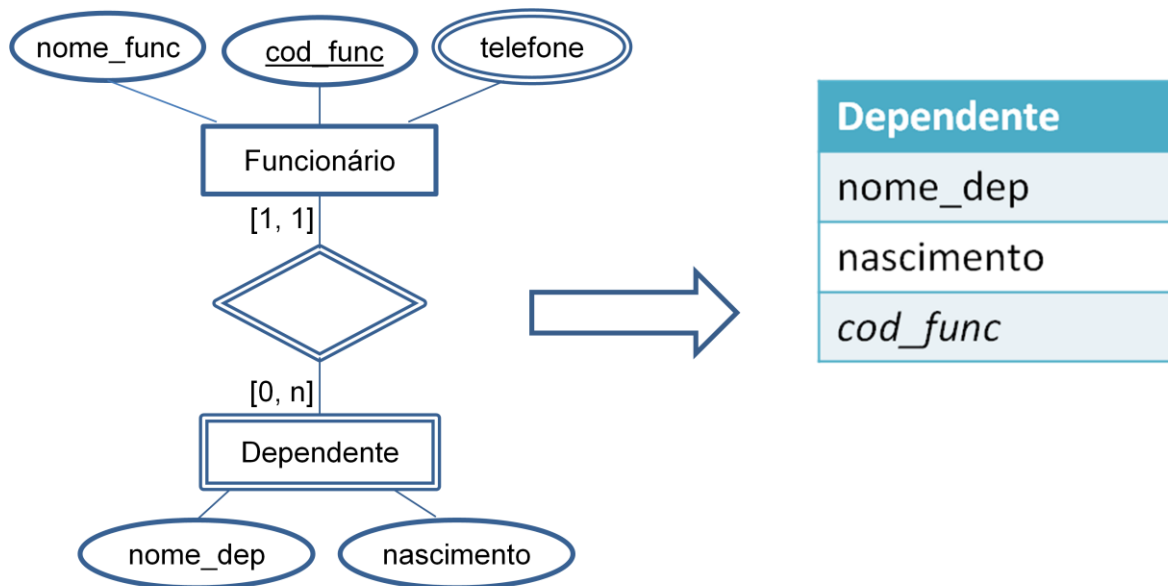


Figura 6: Mapeamento utilizando a regra 4.

Regra 5: Para cada CR do tipo *um para um* incluímos na tabela correspondente ao lado que participa totalmente do relacionamento os atributos que compõem a chave primária do outro lado e os atributos descritivos do relacionamento. Se ambos os lados participam totalmente do relacionamento podemos escolher livremente o lado para o qual exportaremos a chave primária do outro lado. Se nenhum dos lados participa totalmente do relacionamento, é preferível utilizar a regra 3, evitando assim o surgimento de valores *nulos* no banco de dados. No exemplo abaixo, a chave primária do *Funcionário* é exportada para a tabela *Departamento* com o nome de *cod_chefe*, que é mais apropriado. Nessa tabela, *cod_chefe* é uma *chave estrangeira*.

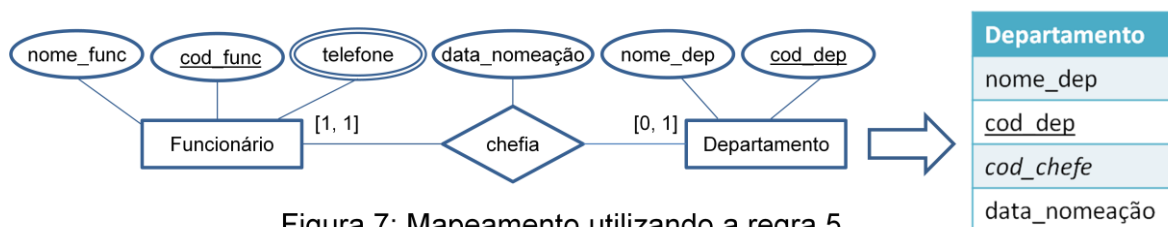


Figura 7: Mapeamento utilizando a regra 5.

Utilizando as cinco regras que mencionamos anteriormente, o mapeamento do DER do Caso Restaurante resulta nas seguintes tabelas.

Composição	Ingrediente	Prato
<u><i>cod_prato</i></u>	<u><i>cod_ingred</i></u>	<u><i>cod_prato</i></u>
<u><i>cod_ingred</i></u>	nome_ingred	nome_prato
quantidade	unidade	preço

O mapeamento do DER do Caso Empresa resulta nas seguintes tabelas.

Projeto	Departamento	Funcionário
<u>cod_proj</u>	nome_dep	<u>cod_func</u>
nome_proj	<u>cod_dep</u>	nome_func
cod_dep	cod_chefe	cod_dep
	data_nomeação	cod_chefe

Func x Projeto	Telefone	Dependente
<u>cod_func</u>	<u>telefone</u>	nome_dep
cod_proj	cod_func	nascimento
		cod_func

Conclusões

Nessa aula, estudamos o Caso Restaurante e o Caso Empresa, tomando por base o modelo entidade-relacionamento para fazermos a modelagem de dados. Destacamos a importância de determinar as entidades, seus atributos e seus relacionamentos e a partir disto chegar ao modelo de dados. Em seguida, expressamos os modelos de dados através de diagramas entidade-relacionamento.

Apresentamos cinco regras que devem ser usadas para mapear um DER para tabelas. Aplicamos essas regras aos diagramas entidade-relacionamento do Caso Restaurante e do Caso Empresa. Finalmente, introduzimos o conceito de chave estrangeira.

Aula 4 – Definição de Dados

Objetivos da Aula

Nesta aula, mostraremos como criar tabelas utilizando a linguagem SQL e o sistema gerenciador de banco de dados MySQL. Veremos também como alterar a estrutura de tabelas já existentes.

4.1 A Linguagem SQL

A linguagem **SQL** (Structured Query Language ou Linguagem de Consulta Estruturada) é uma linguagem de consulta voltada para bancos de dados relacionais. Muitas de suas características foram inspiradas na *álgebra relacional* e no *cálculo relacional*.

SQL foi desenvolvida no início dos anos 70 nos laboratórios da IBM, dentro do projeto *System R*, que tinha como objetivo viabilizar a implementação do modelo relacional de banco de dados proposto por E. F. Codd.

A linguagem SQL obteve enorme sucesso comercial e atualmente é considerada um padrão para banco de dados. Isto decorre da sua simplicidade e facilidade de uso. Embora tenha sido originalmente criada pela IBM, rapidamente surgiram vários "dialetos" de SQL desenvolvidos por outras empresas.

Nesse curso, abordaremos o dialeto de SQL implementado no SGBD **MySQL**. Não é nossa intenção descrever exaustivamente todos os recursos do SQL, mas apenas discutir seus principais comandos e recursos.

4.2 MySQL

MySQL é um sistema gerenciador de banco de dados que utiliza a linguagem SQL como interface. É atualmente um dos SGBD's mais populares, com milhões de instalações pelo mundo. É utilizado por grandes empresas tais como Bradesco, HP, Nokia, Cisco, Google, Sony etc.

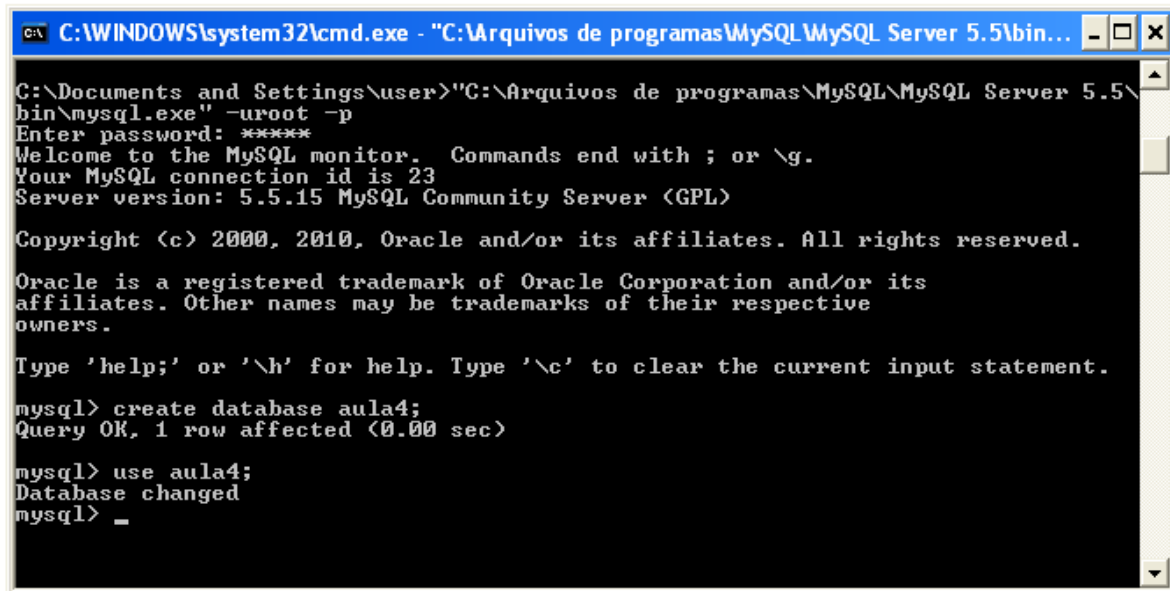
O **MySQL** foi criado na Suécia por dois suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius. Surgiu como software livre e isso contribuiu para a sua rápida disseminação. Em 2008, o MySQL foi adquirido pela *Sun Microsystems*, por US\$ 1 bilhão, um preço jamais visto no setor de software livre. Em 2009 a *Oracle* comprou a *Sun Microsystems* e todos os seus produtos, incluindo o MySQL.

O sucesso do MySQL deve-se em grande medida à fácil integração com o *PHP*, suporte a *Unicode*, *Full Text Indexes*, replicação, *Hot Backup*, *GIS*, *OLAP* e muitos outros recursos de bancos de dados.

4.3 Criando um banco de dados

Antes de criar as tabelas você deverá criar um banco de dados utilizando o comando (cláusula) *create database*. Em geral, os comandos no MySQL devem ser finalizados com ; (ponto-e-vírgula). Para criar um banco de dados chamado *aula4* digite: *create database aula4;*

Para conectar-se a um banco de dados criado anteriormente utilize o comando *connect* ou o comando *use*. É possível remover um banco de dados com o comando *drop database*. Por exemplo, o comando *drop database aula4;* remove o banco de dados *aula4*.



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin\mysql.exe" -uroot -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 23
Server version: 5.5.15 MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database aula4;
Query OK, 1 row affected (0.00 sec)

mysql> use aula4;
Database changed
mysql> _
```

4.4 Criando uma tabela

Para criar uma tabela usamos o comando *create table*. A sintaxe desse comando é:

create table nome_da_tabela (campos da tabela)

Ao especificar um campo da tabela devemos indicar seu nome e seu tipo. Os principais tipos de dados no SQL são:

- **decimal**: armazena um número. Entre parênteses especificamos a quantidade de dígitos do número e a quantidade de casas decimais. Exemplos: *decimal(3)*, *decimal(8,2)*. Esse último exemplo permite guardar números de 8 dígitos, com duas casas decimais.
- **varchar**: armazena uma cadeia de caracteres (palavras). Entre parênteses especificamos o comprimento máximo da cadeia. Exemplo: *varchar(50)*.
- **boolean**: armazena um valor lógico (verdadeiro ou falso).
- **char**: armazena um caractere.
- **date**: armazena uma data.
- **time**: armazena um horário.

Podemos também especificar algumas propriedades dos campos. As principais propriedades são:

- **not null**: indica que o campo é requerido, ou seja, seu valor não pode ser nulo.
- **primary key**: indica que o campo ou conjunto de campos constitui uma chave primária. Os valores dos campos que fazem parte de uma chave primária não podem ser nulos.

- **foreign key:** indica que o campo ou conjunto de campos é uma chave estrangeira e, portanto, está associado à chave primária de outra tabela.
- **unique:** indica que o campo ou conjunto de campos não pode ter valores repetidos, ou seja, têm que ser únicos (mas podem ser nulos). Equivale a indicar que o campo ou conjunto de campos constitui uma chave candidata.
- **check:** permite especificar uma condição para validar o valor do campo. Nessa condição podem ser usados vários operadores da linguagem, como veremos nos exemplos a seguir.
- **default:** permite especificar o valor *default* (padrão) do campo. Se ao inserir uma linha na tabela o valor desse campo não for especificado então tal campo terá o valor padrão.

Como exemplo, vamos criar as tabelas *curso* e *aluno*. Exibimos a estrutura dessas tabelas com o comando *describe*. Os comandos SQL para criar essas tabelas são:

```
create table curso (nome_curso varchar(30) not null, cod_curso decimal(3) primary key, duracao decimal(2) default 4);
```

```
create table aluno (nome_aluno varchar(30) not null, cod_aluno decimal(3), sexo char check (sexo in ('M', 'F')), nascimento date, cod_curso decimal(3), foreign key (cod_curso) references curso(cod_curso));
```

```
mysql> create table curso (nome_curso varchar(30) not null, cod_curso decimal(3)
primary key, duracao decimal(2) default 4);
Query OK, 0 rows affected (0.08 sec)

mysql> describe curso;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nome_curso | varchar(30) | NO | | NULL | |
| cod_curso | decimal(3,0) | NO | PRI | NULL | |
| duracao | decimal(2,0) | YES | | 4 | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> create table aluno (nome_aluno varchar(30) not null, cod_aluno decimal(3)
, sexo char check (sexo in ('M', 'F')), nascimento date, cod_curso decimal(3),
foreign key (cod_curso) references curso(cod_curso));
Query OK, 0 rows affected (0.08 sec)

mysql> describe aluno;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nome_aluno | varchar(30) | NO | | NULL | |
| cod_aluno | decimal(3,0) | YES | | NULL | |
| sexo | char(1) | YES | | NULL | |
| nascimento | date | YES | | NULL | |
| cod_curso | decimal(3,0) | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

4.5 Alterando a estrutura de uma tabela

Podemos alterar a estrutura (esquema) de uma tabela com o comando *alter table*. As principais alterações que podem ser feitas são:

- **adicionar um campo.** Exemplo: `alter table aluno add cpf numeric(11);`
- **alterar o nome e o tipo de um campo.** Exemplo: `alter table aluno change cpf cpf_aluno varchar(11);`
- **alterar o tipo de um campo.** Exemplo: `alter table aluno modify nome_aluno varchar(40);`
- **adicionar uma restrição (primary key, foreign key, unique etc).** Exemplos: `alter table aluno add constraint primary key (cod_aluno);` `alter table aluno add constraint cp unique (cpf_aluno);` (note que a restrição foi criada com um nome, nesse exemplo cp)
- **remover uma restrição.** Exemplos: `alter table aluno drop primary key;` `alter table aluno drop index cp;`
- **remover um campo.** Exemplo: `alter table aluno drop cpf_aluno;`

Exibimos abaixo os resultados obtidos ao submeter estes exemplos de alteração de estrutura de tabelas.

```

C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> alter table aluno change cpf cpf_aluno varchar(11);
Query OK, 0 rows affected (0.14 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table aluno modify nome_aluno varchar(40);
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table aluno add constraint primary key (cod_aluno);
Query OK, 0 rows affected (0.19 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table aluno add constraint cp unique (cpf_aluno);
Query OK, 0 rows affected (0.17 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> describe aluno;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nome_aluno | varchar(40) | YES |     | NULL    |       |
| cod_aluno  | decimal(3,0) | NO  | PRI | 0        |       |
| sexo      | char(1)      | YES |     | NULL    |       |
| nascimento | date         | YES |     | NULL    |       |
| cod_curso  | decimal(3,0) | YES | MUL | NULL    |       |
| cpf_aluno  | varchar(11)  | YES | UNI | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.02 sec)

mysql> alter table aluno drop primary key;
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table aluno drop index cp;
Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table aluno drop cpf_aluno;
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0

```

Para remover uma tabela usamos o comando `drop table`. Exemplo: `drop table aluno;`

```
C:\WINDOWS\system32\cmd.exe
Records: 0 Duplicates: 0 Warnings: 0
mysql> alter table aluno drop index cp;
Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> alter table aluno drop cpf_aluno;
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> drop table aluno;
Query OK, 0 rows affected (0.03 sec)
mysql> drop table curso;
Query OK, 0 rows affected (0.03 sec)
mysql> exit
Bye
C:\Documents and Settings\user>
```

Conclusões

Nessa aula, introduzimos a linguagem de banco de dados SQL. Vimos como criar tabelas e como alterar a estrutura de tabelas já existentes usando SQL. Mostramos como instalar o *SGBD MySQL* e utilizamos o aplicativo *MySQL Monitor* para criar tabelas e alterar sua estrutura.

Apresentamos os principais tipos de dados disponíveis no MySQL e discutimos as principais restrições de integridade que podem ser aplicadas às tabelas (cláusulas *not null*, *primary key*, *foreign key*, *unique* e *check*)

Aula 5 – Manipulação de Dados

Objetivos da Aula

Nesta aula, mostraremos como utilizar os comandos *insert*, *update* e *delete* do SQL, que servem para inserir, alterar e remover dados contidos nas tabelas.

Discutiremos também as principais restrições de integridade e veremos como garantir que elas sejam respeitadas ao realizar operações de inserção, alteração e remoção nas tabelas.

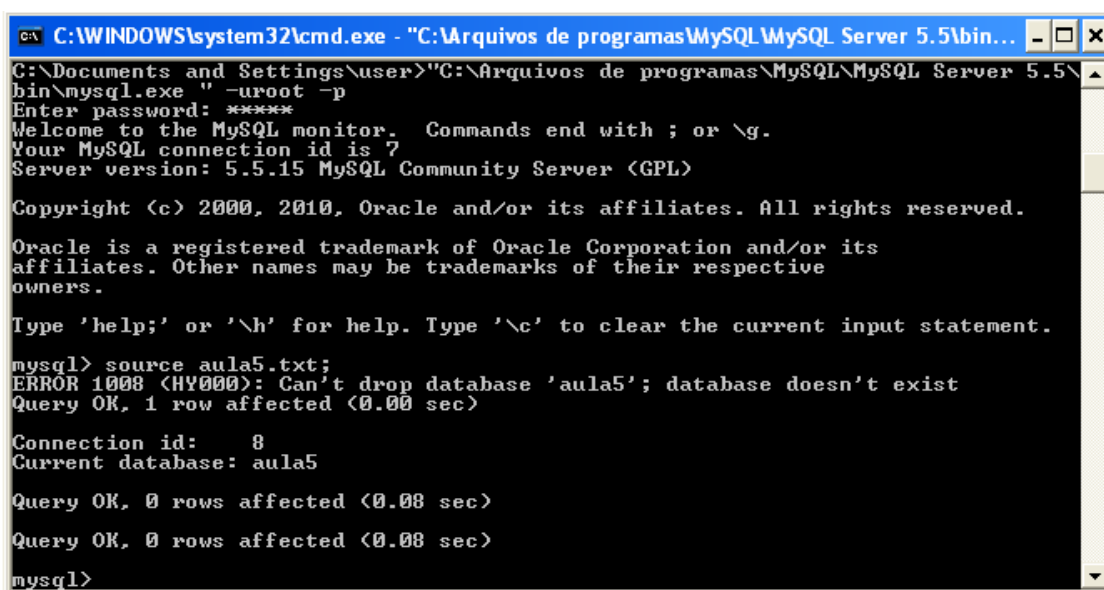
5.1 Executando um Script

Podemos colocar diversos comandos SQL num arquivo de texto e depois de entrar no MySQL Monitor executá-los com apenas uma instrução. Um arquivo contendo diversos comandos é chamado de *script*. Para executar um script basta utilizar o comando *source* seguido do endereço do script.

Por exemplo, suponha que na pasta que estávamos quando entramos no MySQL Monitor tenhamos o arquivo *aula5.txt* com o seguinte conteúdo¹:

```
drop database aula5; # remove o banco de dados aula5, caso ele exista
create database aula5; # cria o banco de dados aula5
connect aula5; # conectando com o banco de dados aula5
# criação das tabelas curso e aluno
create table curso (nome_curso varchar(30) not null, cod_curso decimal(3) primary key, duracao decimal(2) default 4);
create table aluno (nome_aluno varchar(30) not null, cod_aluno decimal(3) primary key, sexo char check (sexo in ('M', 'F')), nascimento date, cod_curso decimal(3), foreign key (cod_curso) references curso(cod_curso));
```

Para executar o script *aula5.txt*, após entrar no MySQL Monitor, devemos digitar: *source aula5.txt*;



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
C:\Documents and Settings\user>"C:\Arquivos de programas\MySQL\MySQL Server 5.5\
bin\mysql.exe " -uroot -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.5.15 MySQL Community Server <GPL>

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> source aula5.txt;
ERROR 1008 (HY000): Can't drop database 'aula5'; database doesn't exist
Query OK, 1 row affected (0.00 sec)

Connection id:      8
Current database: aula5

Query OK, 0 rows affected (0.08 sec)

Query OK, 0 rows affected (0.08 sec)

mysql>
```

¹ O texto após o símbolo # até o final da linha é entendido como comentário.

5.2 Inserindo Linhas

Para inserir uma linha numa tabela usamos o comando *insert* que pode ser usado de duas maneiras:

insert into <tabela> (lista de campos) values (lista de valores);

ou

insert into <tabela> values (lista de valores);

Se usarmos a primeira maneira, será inserida uma linha onde os campos especificados na lista de campos terão os valores especificados na lista de valores. Os campos da tabela omitidos na lista de campos terão valor *nulo* ou o seu valor *default*, caso tal valor tenha sido definido na criação da tabela.

A segunda maneira serve para inserir uma linha com os valores especificados na lista de valores. Nesse caso, é preciso fornecer um valor para cada um dos campos da tabela. Além disso, os valores devem estar na mesma ordem que os campos da tabela.

Vamos inserir uma linha na tabela *curso* e três linhas na tabela *aluno*:

insert into curso (cod_curso, nome_curso) values (1, "Informatica");

insert into aluno (cod_aluno, nome_aluno, sexo, cod_curso) values (1, "Rui", "M", 1);

insert into aluno values ("Ana", 2, "F", "1994/05/16", 1);

insert into aluno values ("Ivo", 3, "M", null, 1);

No primeiro exemplo omitimos o campo *duracao*. Nesse caso, o valor desse campo será 4, que é o seu valor default. No segundo exemplo, omitimos o campo *nascimento* e, portanto, o valor desse campo será *nulo*. Nos últimos dois exemplos especificamos todos os valores de acordo com a ordem dos campos na tabela. Observe que a data tem que estar no formato *ano/mês/dia*. Nas próximas duas aulas discutiremos com detalhes a cláusula *select*, que serve para consultar os dados das tabelas. Na tela abaixo usamos o comando *select * from <tabela>;* para exibir todas as linhas da tabela.

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin\mysql.exe" -uroot -p
mysql> insert into curso (cod_curso, nome_curso) values (1, "Informatica");
Query OK, 1 row affected (0.06 sec)

mysql> insert into aluno (cod_aluno, nome_aluno, sexo, cod_curso) values (1, "Rui", "M", 1);
Query OK, 1 row affected (0.03 sec)

mysql> insert into aluno values ("Ana", 2, "F", "1994/05/16", 1);
Query OK, 1 row affected (0.02 sec)

mysql> insert into aluno values ("Ivo", 3, "M", null, 1);
Query OK, 1 row affected (0.02 sec)

mysql> select * from curso;
+-----+-----+-----+
| nome_curso | cod_curso | duracao |
+-----+-----+-----+
| Informatica | 1 | 4 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from aluno;
+-----+-----+-----+-----+-----+
| nome_aluno | cod_aluno | sexo | nascimento | cod_curso |
+-----+-----+-----+-----+-----+
| Rui | 1 | M | NULL | 1 |
| Ana | 2 | F | 1994-05-16 | 1 |
| Ivo | 3 | M | NULL | 1 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

5.3 Restrições de Integridade

O comando *insert* resultará numa falha se os dados que se deseja inserir violarem alguma restrição de integridade do banco de dados. As principais restrições de integridade são:

Integridade de domínio: cada campo possui um domínio, que é o conjunto dos valores admissíveis para esse campo. O domínio é definido quando especificamos o tipo do campo. O domínio pode ser restringido com o uso da cláusula *check*, na qual especificamos uma condição que deve ser satisfeita para que o valor seja admissível (a cláusula *check* não funciona no MySQL 5.5). A cláusula *not null* também restringe o domínio pois indica que o valor nulo não é admissível para esse campo. Abaixo exibimos exemplos de comandos que resultarão em falha, comentando o motivo da falha.

```
insert into curso (cod_curso, nome_curso) values ("X", "Direito");
```

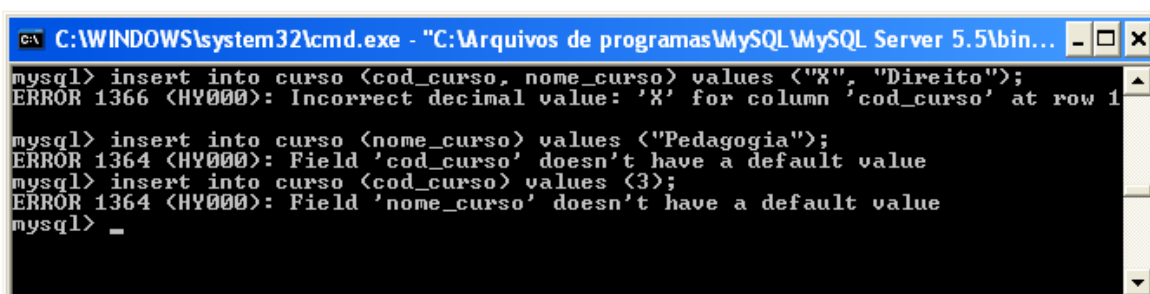
Erro: O valor de *cod_curso* tem que ser um número inteiro.

```
insert into curso (nome_curso) values ("Pedagogia");
```

Erro: Não foi especificado o valor do *cod_curso*. Esse campo é chave primária e não pode ser nulo.

```
insert into curso (cod_curso) values (3);
```

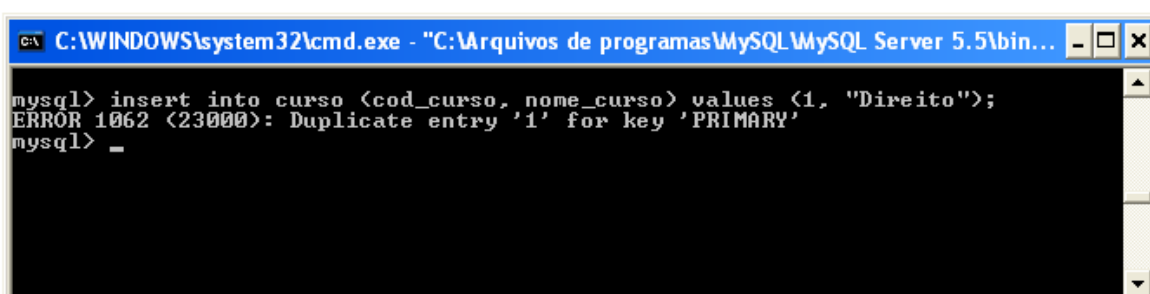
Erro: Não foi especificado o valor do *nome_curso* e esse campo não pode ser nulo.



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> insert into curso (cod_curso, nome_curso) values ('X', 'Direito');
ERROR 1366 (HY000): Incorrect decimal value: 'X' for column 'cod_curso' at row 1
mysql> insert into curso (nome_curso) values ('Pedagogia');
ERROR 1364 (HY000): Field 'cod_curso' doesn't have a default value
mysql> insert into curso (cod_curso) values (3);
ERROR 1364 (HY000): Field 'nome_curso' doesn't have a default value
mysql> _
```

Unicidade das chaves candidatas: os valores das chaves candidatas não podem ser repetidos. As chaves primárias são chaves candidatas. A cláusula *unique* também permite definir uma chave candidata (ver as seções 4.4 e 4.5). No exemplo abaixo ocorre uma falha, pois já havíamos inserido um curso com o código 1.

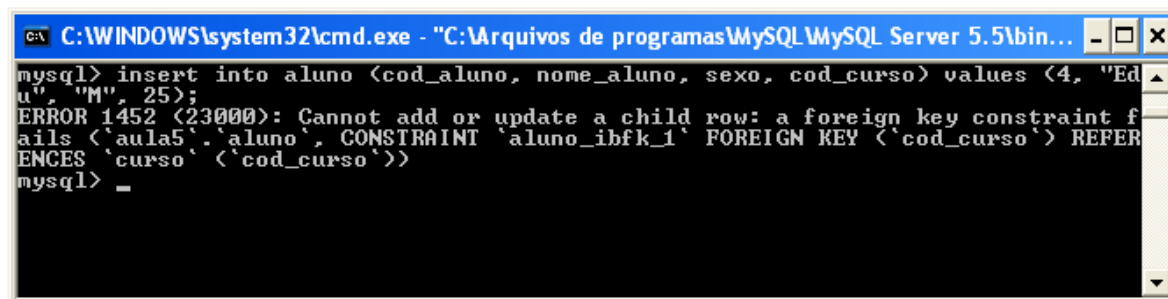
```
insert into curso (cod_curso, nome_curso) values (1, "Direito");
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> insert into curso (cod_curso, nome_curso) values (1, 'Direito');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> _
```


Integridade referencial: os valores das chaves estrangeiras têm que ser um subconjunto das chaves primárias correspondentes a elas. Por exemplo, o `cod_curso` de um aluno tem que ser um dos `cod_curso` existentes na tabela `curso`, pois um aluno não pode ser vinculado a um curso que não está cadastrado. No exemplo abaixo ocorre uma falha, pois na tabela `curso` não existe um curso com código 25.

```
insert into aluno (cod_aluno, nome_aluno, sexo, cod_curso) values (4, "Edu", "M", 25);
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> insert into aluno (cod_aluno, nome_aluno, sexo, cod_curso) values (4, "Edu", "M", 25);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails ('aula5'.aluno, CONSTRAINT 'aluno_ibfk_1' FOREIGN KEY ('cod_curso') REFERENCES 'curso' ('cod_curso'))
mysql> _
```

5.4 Alterando Linhas

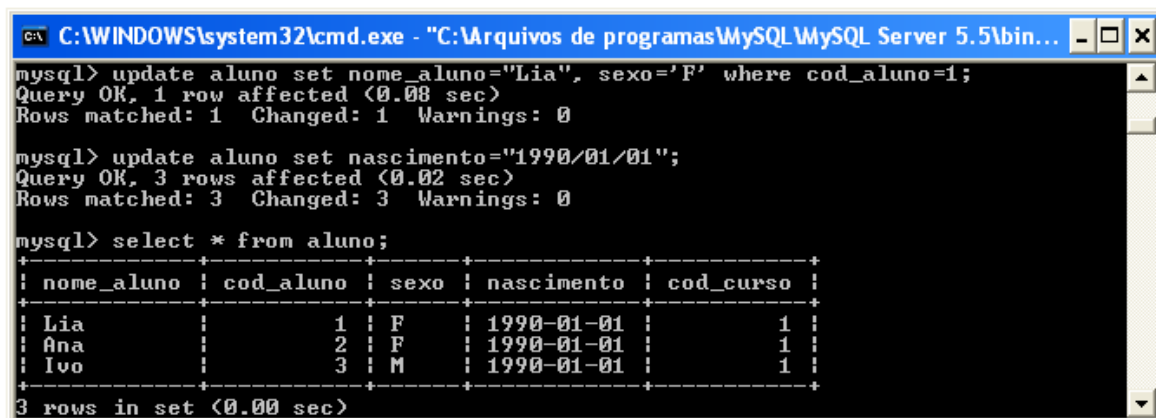
Para alterar linhas de uma tabela usamos o comando `update`. A sintaxe básica desse comando é:

```
update <tabela> set campo1=valor1, campo2=valor2, ... [where <condição>];
```

O uso de colchetes na linha acima indica que a cláusula `where` é opcional. Se ela não for declarada, a alteração será feita em todas as linhas da tabela. A seguir temos dois exemplos de alteração. No primeiro alteramos o nome do aluno cujo código é 1 para "Lia" e o sexo para 'F'. No segundo, alteramos a data de nascimento de todos os alunos para 01/01/1990.

```
update aluno set nome_aluno="Lia", sexo='F' where cod_aluno=1;
```

```
update aluno set nascimento="1990/01/01";
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> update aluno set nome_aluno="Lia", sexo='F' where cod_aluno=1;
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update aluno set nascimento="1990/01/01";
Query OK, 3 rows affected (0.02 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from aluno;
+-----+-----+-----+-----+-----+
| nome_aluno | cod_aluno | sexo | nascimento | cod_curso |
+-----+-----+-----+-----+-----+
| Lia        | 1        | F    | 1990-01-01 | 1         |
| Ana        | 2        | F    | 1990-01-01 | 1         |
| Ivo        | 3        | M    | 1990-01-01 | 1         |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

5.5 Cláusula where

Podemos utilizar diversos operadores para definir condições mais complexas na cláusula *where*. Os principais operadores são:

- **Operadores aritméticos:** +, -, *, /. Exemplo: *where saldo = 2*limite*
- **Operadores relacionais:** =, >, <, >=, <=, <>. Exemplo: *where salario <> 545*
- **Operadores lógicos:** and, or, not. Exemplo: *where nota >= 4 and nota <= 7*
- **Outros operadores:** in, between, like. Exemplos: *where cod_curso in (2, 4, 1)*, *where nascimento between "1980/01/01" and "1989/12/31"*, *where nome_aluno like "_N%"*

Os operadores acima são comuns nas linguagens de programação e bastante conhecidos, com exceção dos três últimos. O operador *in* testa se um valor pertence a um conjunto de valores. O operador *between* verifica se um valor está dentro de uma faixa de valores.

Já o operador *like* serve para testar se uma string se assemelha a outra. O operando do *like* é uma *expressão regular* que denota um conjunto de strings que têm um determinado formato. Na expressão regular podemos usar os metacaracteres "_", que significa *um caractere qualquer*, e "%", que significa uma *sequência qualquer de caracteres*. Por exemplo, a expressão regular "_N%" simboliza todas as strings cujo segundo caractere é a letra "N" (não importando se é minúsculo ou maiúsculo). O nome "Ana" casa com essa expressão regular.

O comando *update* também pode resultar em falha se a alteração solicitada violar alguma restrição de integridade. As restrições de integridade de domínio, unicidade das chaves candidatas e integridade referencial precisam ser respeitadas. As seguintes tentativas de alteração resultarão em falha:

```
update curso set nome_curso=null where cod_curso=1;
```

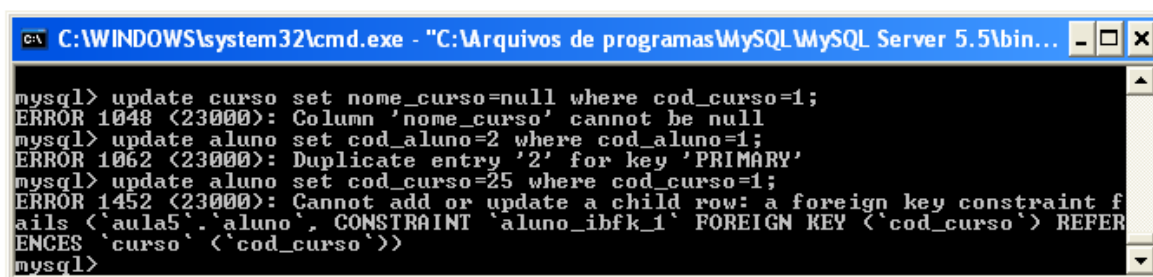
Erro: O valor de nome_curso não pode ser nulo (integridade de domínio violada).

```
update aluno set cod_aluno=2 where cod_aluno=1;
```

Erro: Já existe um aluno com código 1 (unicidade de chave candidata violada).

```
update aluno set cod_curso=25 where cod_curso=1;
```

Erro: Não foi cadastrado curso com código 25 (integridade referencial violada).



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> update curso set nome_curso=null where cod_curso=1;
ERROR 1048 (23000): Column 'nome_curso' cannot be null
mysql> update aluno set cod_aluno=2 where cod_aluno=1;
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
mysql> update aluno set cod_curso=25 where cod_curso=1;
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint f
ails ('aula5`.`aluno`, CONSTRAINT `aluno_ibfk_1` FOREIGN KEY (`cod_curso`) REFER
ENCES `curso` (`cod_curso`))
mysql>
```

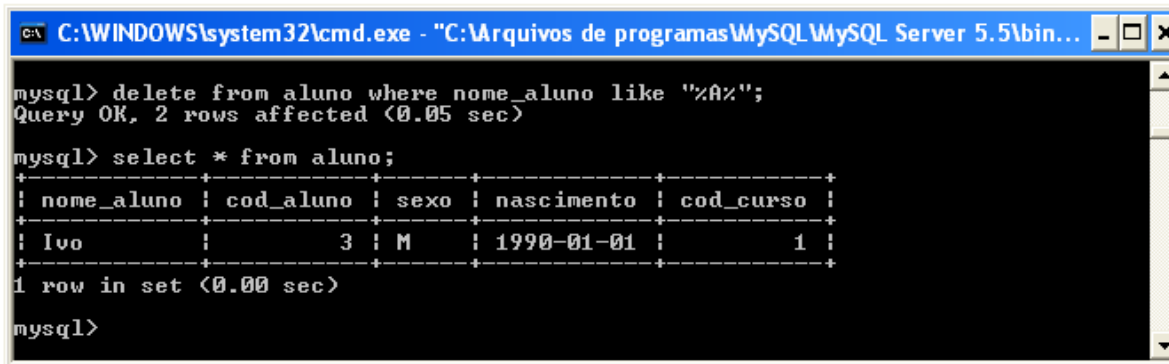
5.6 Removendo Linhas

Para remover linhas de uma tabela usamos o comando *delete*. A sintaxe básica desse comando é:

```
delete from <tabela> [where <condição>];
```

Esse comando remove todas as linhas da tabela que satisfazem a condição. Se a cláusula *where* não for declarada, serão removidas todas as linhas da tabela. No exemplo a seguir removemos todos os alunos cujo nome tem a letra "A".

```
delete from aluno where nome_aluno like "%A%";
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> delete from aluno where nome_aluno like "%A%";
Query OK, 2 rows affected (0.05 sec)

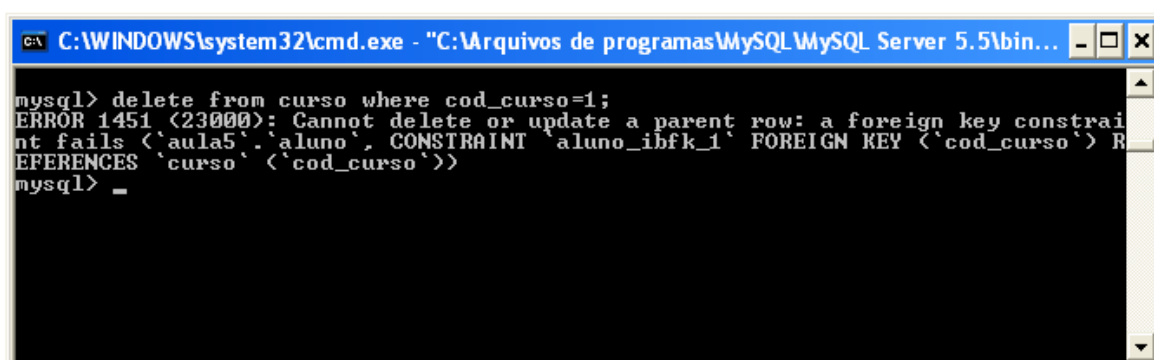
mysql> select * from aluno;
+-----+-----+-----+-----+-----+
| nome_aluno | cod_aluno | sexo | nascimento | cod_curso |
+-----+-----+-----+-----+-----+
| Ivo        | 3        | M    | 1990-01-01 | 1         |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

O comando *delete* também pode resultar em falha se a remoção solicitada causar uma violação na integridade referencial do banco de dados. Por exemplo, a tentativa de remoção abaixo resultará em falha pois não podemos remover o curso 1 visto que existe um aluno vinculado a ele.

```
delete from curso where cod_curso=1;
```

Se quisermos remover o curso 1 precisamos primeiramente nos certificar que nenhum aluno está vinculado a esse curso. Podemos fazer isso removendo os alunos do curso 1 ou transferindo os alunos do curso 1 para outros cursos.



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> delete from curso where cod_curso=1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails ('aula5`.`aluno`, CONSTRAINT `aluno_ibfk_1` FOREIGN KEY (`cod_curso`) REFERENCES `curso` (`cod_curso`))
mysql> _
```

Conclusões

Nesta aula vimos como utilizar os comandos *insert*, *update* e *delete* do SQL para inserir, alterar e remover dados contidos nas tabelas.

Discutimos também as principais restrições de integridade: integridade de domínio, unicidade das chaves candidatas e integridade referencial. Vimos como garantir que elas sejam respeitadas ao realizar operações de inserção, alteração e remoção.

Aula 6 – Consultas

Objetivos da Aula

Nesta aula, veremos como utilizar a cláusula *select* para consultar os dados contidos nas tabelas. Apresentaremos as cláusulas *order by* e *distinct* e as operações de junção e junção natural de tabelas. Veremos também o que são *alias* e *subselect*. Finalmente, abordaremos as operações de união, complemento e interseção de tabelas.

6.1 Cláusula Select

Para realizar consultas em SQL utilizamos a cláusula *select*. A sintaxe básica dessa cláusula é:

```
select <lista de campos> from <lista de tabelas> [where <condição>]
```

Na *lista de campos* especificamos quais os campos que desejamos exibir. Na *lista de tabelas* indicamos quais as tabelas que contém os dados que devem ser consultados. A *condição* indica quais linhas devem ser exibidas no resultado da consulta. Como a condição é opcional, se ela for omitida, todas as linhas serão exibidas.

O resultado do *select* é obtido da seguinte maneira: é calculada uma tabela cujos campos são todos os campos das tabelas que aparecem na lista de tabelas. As linhas dessa tabela são obtidas fazendo-se todas as combinações possíveis das linhas das tabelas envolvidas na consulta. Essa operação é conhecida como *produto cartesiano* das tabelas.

Em seguida, são selecionadas apenas as linhas resultantes do produto cartesiano que satisfazem à *condição*. Após essa seleção, são eliminados os campos que não constam na *lista de campos*, chegando-se assim ao resultado final.

Vamos analisar alguns exemplos de consultas. Nestes exemplos vamos utilizar as tabelas abaixo, que você deve ter criado quando fez o exercício 1 da aula 5.

Agencia

codagencia	nomeagencia	cidade
1	Lapa	SP
2	Sé	SP
3	Lapa	RJ

Conta_Poupança

numero	codcliente	codagencia	saldo
3	3	2	100
1	5	3	200
3	4	3	300

Cliente

codcliente	nomecliente
1	Rui
2	Ana
3	Lia
4	Gil
5	Ivo

Conta_corrente

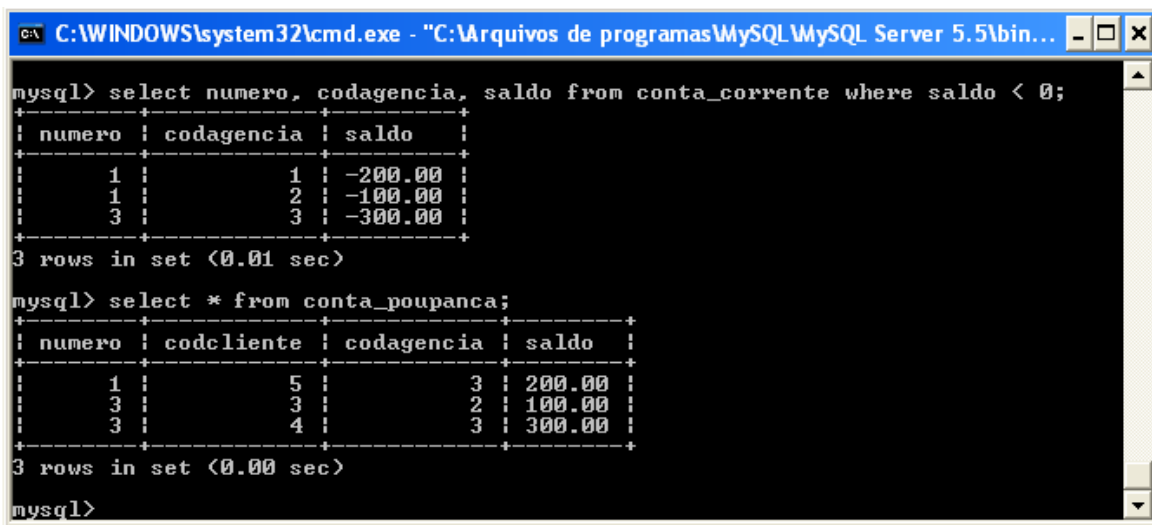
numero	codcliente	codagencia	saldo	limite
1	1	1	-200	200
1	1	2	-100	300
1	1	3	0	0
2	1	1	100	0
3	2	1	200	200
2	2	2	300	0
4	3	1	400	0
3	4	2	500	0
2	4	3	600	100
3	3	3	-300	100
4	5	2	800	600

Vejamos primeiramente alguns exemplos de uso do *select* envolvendo uma única tabela:

```
select numero, codagencia, saldo from conta_corrente where saldo < 0;
```

```
select * from conta_poupanca;
```

No resultado da primeira consulta veremos o número, código da agência e saldo das contas correntes que estão com saldo negativo. Na segunda consulta veremos todas as linhas da tabela *conta_poupanca*. Observe que após o *select* utilizamos o símbolo * (asterisco) que significa "todos os campos". O resultado dessas consultas aparece abaixo.



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe - \"C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...\". The window contains the following text:

```
mysql> select numero, codagencia, saldo from conta_corrente where saldo < 0;
+-----+-----+-----+
| numero | codagencia | saldo |
+-----+-----+-----+
| 1      | 1          | -200.00 |
| 1      | 2          | -100.00 |
| 3      | 3          | -300.00 |
+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> select * from conta_poupanca;
+-----+-----+-----+-----+
| numero | codcliente | codagencia | saldo |
+-----+-----+-----+-----+
| 1      | 5          | 3          | 200.00 |
| 3      | 3          | 2          | 100.00 |
| 3      | 4          | 3          | 300.00 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Podemos ordenar o resultado de um *select* com a cláusula *order by*, cuja sintaxe é:

```
order by campo1 [asc|desc], campo2 [asc|desc], ...
```

Observe que podemos definir vários níveis de ordenação. Note ainda que a ordenação pode ser crescente (*asc*), que é o padrão, ou decrescente (*desc*). Para listar as contas correntes em ordem crescente de agência e, em cada agência, em ordem decrescente de saldo, podemos usar a expressão:

```
select * from conta_corrente order by codagencia, saldo desc;
```

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select * from conta_corrente order by codagencia, saldo desc;
+-----+-----+-----+-----+-----+
| numero | codcliente | codagencia | saldo  | limite |
+-----+-----+-----+-----+-----+
| 4      | 3          | 1          | 400.00 | 0.00   |
| 3      | 2          | 1          | 200.00 | 200.00 |
| 2      | 1          | 1          | 100.00 | 0.00   |
| 1      | 1          | 1          | -200.00| 200.00 |
| 4      | 5          | 2          | 800.00 | 600.00 |
| 3      | 4          | 2          | 500.00 | 0.00   |
| 2      | 2          | 2          | 300.00 | 0.00   |
| 1      | 1          | 2          | -100.00| 300.00 |
| 2      | 4          | 3          | 600.00 | 100.00 |
| 1      | 1          | 3          | 0.00   | 0.00   |
| 3      | 3          | 3          | -300.00| 100.00 |
+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)

mysql>
```

A consulta `select codagencia, limite from conta_corrente;` produzirá linhas repetidas. Podemos eliminá-las usando a cláusula `distinct`, como no exemplo abaixo.

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select codagencia, limite from conta_corrente;
+-----+-----+
| codagencia | limite |
+-----+-----+
| 1          | 200.00 |
| 2          | 300.00 |
| 3          | 0.00   |
| 1          | 0.00   |
| 2          | 0.00   |
| 3          | 100.00 |
| 1          | 200.00 |
| 2          | 0.00   |
| 3          | 100.00 |
| 1          | 0.00   |
| 2          | 600.00 |
+-----+-----+
11 rows in set (0.00 sec)

mysql> select distinct codagencia, limite from conta_corrente;
+-----+-----+
| codagencia | limite |
+-----+-----+
| 1          | 200.00 |
| 2          | 300.00 |
| 3          | 0.00   |
| 1          | 0.00   |
| 2          | 0.00   |
| 3          | 100.00 |
| 2          | 600.00 |
+-----+-----+
7 rows in set (0.00 sec)
```

Vejamos agora um exemplo envolvendo várias tabelas.

```
select nomecliente, nomeagencia from cliente, agencia where cidade="SP";
```

Nesse exemplo, cada linha de `cliente` é combinada com todas as linhas de `agencia`. Em seguida, são selecionadas apenas as linhas onde `cidade="SP"`. Por fim, são exibidos os campos `nomecliente` e `nomeagencia`.

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select nomecliente, nomeagencia from cliente, agencia where cidade="SP";
+-----+-----+
| nomecliente | nomeagencia |
+-----+-----+
| Rui         | Lapa        |
| Rui         | Sé          |
| Ana         | Lapa        |
| Ana         | Sé          |
| Lia         | Lapa        |
| Lia         | Sé          |
| Gil         | Lapa        |
| Gil         | Sé          |
| Ivo         | Lapa        |
| Ivo         | Sé          |
+-----+-----+
10 rows in set (0.00 sec)

mysql> _
```

6.2 Junção de Tabelas

O exemplo anterior produz um resultado um tanto sem sentido. Frequentemente, quando uma consulta envolve duas tabelas, desejamos associar a cada linha de uma das tabelas as linhas da outra tabela que lhe sejam correspondentes. Em geral, essa correspondência é definida pela *chave estrangeira* de uma das tabelas que corresponde à *chave primária* da outra tabela. Esse par de chaves permite fazer a operação conhecida como *junção* de tabelas.

Por exemplo, para associar cada conta de poupança com o seu respectivo titular devemos fazer a junção das tabelas *cliente* e *conta_poupanca*:

```
select * from cliente, conta_poupanca where cliente.codcliente = conta_poupanca.codcliente;
```

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select * from cliente, conta_poupanca where cliente.codcliente = conta_poupanca.codcliente;
+-----+-----+-----+-----+-----+-----+
| codcliente | nomecliente | numero | codcliente | codagencia | saldo |
+-----+-----+-----+-----+-----+-----+
| 3         | Lia         | 3      | 3         | 2         | 100.00 |
| 4         | Gil         | 3      | 4         | 3         | 300.00 |
| 5         | Ivo         | 1      | 5         | 3         | 200.00 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Observe que como a chave estrangeira e a chave primária têm o mesmo nome (*codcliente*) foi preciso indicar o nome da tabela como *prefixo* do nome do campo, de modo a evitar ambiguidade na condição. Ao utilizar a notação *cliente.codcliente* estamos indicando o campo *codcliente* da tabela *cliente*.

6.3 Cláusula Join

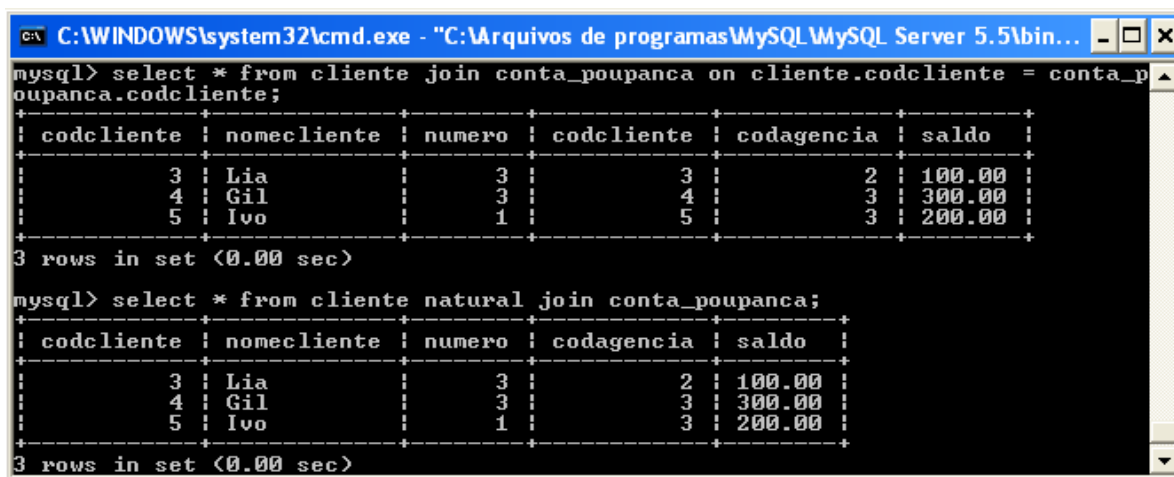
É possível expressar uma junção utilizando a cláusula *join*, como no exemplo a seguir

```
select * from cliente join conta_poupanca on cliente.codcliente = conta_poupanca.codcliente;
```

Observe que o critério de junção é especificado logo após a palavra *on*. É natural que a chave estrangeira e a chave primária correspondente tenham o mesmo nome. Quando isso ocorre podemos utilizar uma forma simplificada de junção chamada de *junção natural*. Nesse tipo de junção o critério utilizado para juntar as linhas das tabelas exige que todos os campos de mesmo nome nas duas tabelas tenham os mesmos valores.

Para indicar que a junção é natural usamos a cláusula *natural join* e omitimos o critério de junção. A consulta a seguir é equivalente à consulta acima, mas note que o campo *codcliente* aparece apenas uma vez o que, em geral, é desejável.

```
select * from cliente natural join conta_poupanca;
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin\mysql.exe"
mysql> select * from cliente join conta_poupanca on cliente.codcliente = conta_poupanca.codcliente;
+-----+-----+-----+-----+-----+-----+
| codcliente | nomecliente | numero | codcliente | codagencia | saldo |
+-----+-----+-----+-----+-----+-----+
| 3 | Lia | 3 | 3 | 2 | 100.00 |
| 4 | Gil | 3 | 4 | 3 | 300.00 |
| 5 | Ivo | 1 | 5 | 3 | 200.00 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from cliente natural join conta_poupanca;
+-----+-----+-----+-----+-----+
| codcliente | nomecliente | numero | codagencia | saldo |
+-----+-----+-----+-----+-----+
| 3 | Lia | 3 | 2 | 100.00 |
| 4 | Gil | 3 | 3 | 300.00 |
| 5 | Ivo | 1 | 3 | 200.00 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Podemos definir um *alias* (apelido) para um campo ou para uma tabela. Esse recurso pode simplificar a consulta e ser usado para evitar ambiguidade quando desejamos fazer a junção de uma tabela com ela mesma. Para definir um alias basta colocá-lo após o nome do campo ou tabela, como no exemplo a seguir.

```
select numero, codagencia, c.codcliente, nomecliente, (saldo + limite) disponivel from cliente c natural join conta_corrente where c.codcliente = 1;
```

Essa consulta exibe os dados das contas correntes do cliente cujo código é 1. Observe que, para cada conta informamos o valor disponível que é obtido através da soma do saldo da conta com o limite de crédito. A coluna relativa a essa informação recebeu o alias *disponivel*. Definimos também o alias *c* para a tabela *cliente*. Esta prática simplificou o prefixo usado no atributo *codcliente*.

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select numero, codagencia, c.codcliente, nomecliente, (saldo + limite)
disponivel from cliente c natural join conta_corrente where c.codcliente = 1;
+-----+-----+-----+-----+-----+
| numero | codagencia | codcliente | nomecliente | disponivel |
+-----+-----+-----+-----+-----+
| 1      | 1          | 1          | Rui         | 0.00       |
| 1      | 2          | 1          | Rui         | 200.00     |
| 1      | 3          | 1          | Rui         | 0.00       |
| 2      | 1          | 1          | Rui         | 100.00     |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

6.4 Subselect

Podemos utilizar o resultado de um *select* dentro de outro. O *select* interno é chamado de *subselect* e deve possuir um *alias*. Vejamos um exemplo.

```
select nomecliente, nomeagencia from cliente, (select * from agencia where cidade="SP") sp;
```

Essa consulta produz o mesmo resultado que a consulta *select nomecliente, nomeagencia from cliente, agencia where cidade="SP";*. No entanto, ela é computada com mais eficiência, pois antes de fazer o produto cartesiano das tabelas selecionamos apenas as agências cuja cidade é "SP".

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select nomecliente, nomeagencia from cliente, (select * from agencia wher
e cidade="SP") sp;
+-----+-----+
| nomecliente | nomeagencia |
+-----+-----+
| Rui         | Lapa        |
| Rui         | Sé          |
| Ana         | Lapa        |
| Ana         | Sé          |
| Lia         | Lapa        |
| Lia         | Sé          |
| Gil         | Lapa        |
| Gil         | Sé          |
| Ivo         | Lapa        |
| Ivo         | Sé          |
+-----+-----+
10 rows in set (0.00 sec)

mysql>
```

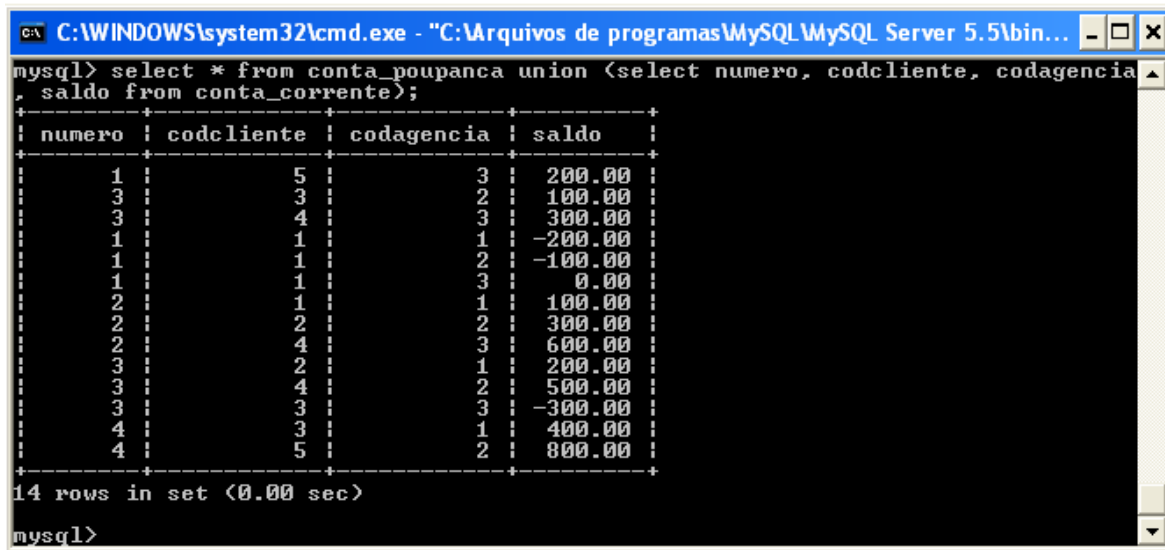
6.5 União de Tabelas

Podemos unir duas tabelas desde que elas sejam *união compatíveis*. Duas tabelas são *união compatíveis* se elas possuem os mesmos campos. Para fazer a *união* utilizamos a cláusula *union*.

Por exemplo, para produzir uma relação de todas as contas correntes e contas de poupança podemos usar a expressão a seguir.

```
select * from conta_poupanca union (select numero, codcliente, codagencia, saldo from conta_corrente);
```

Observe que tivemos que eliminar o campo *limite* da tabela *conta_corrente*, através de um *subselect*, para torná-la união compatível com a tabela *conta_poupança*.



The screenshot shows a MySQL command window with the following query and result:

```
mysql> select * from conta_poupanca union (select numero, codcliente, codagencia, saldo from conta_corrente);
```

numero	codcliente	codagencia	saldo
1	5	3	200.00
3	3	2	100.00
3	4	3	300.00
1	1	1	-200.00
1	1	2	-100.00
1	1	3	0.00
2	1	1	100.00
2	2	2	300.00
2	4	3	600.00
3	2	1	200.00
3	4	2	500.00
3	3	3	-300.00
4	3	1	400.00
4	5	2	800.00

14 rows in set (0.00 sec)

6.6 Complemento de Tabelas

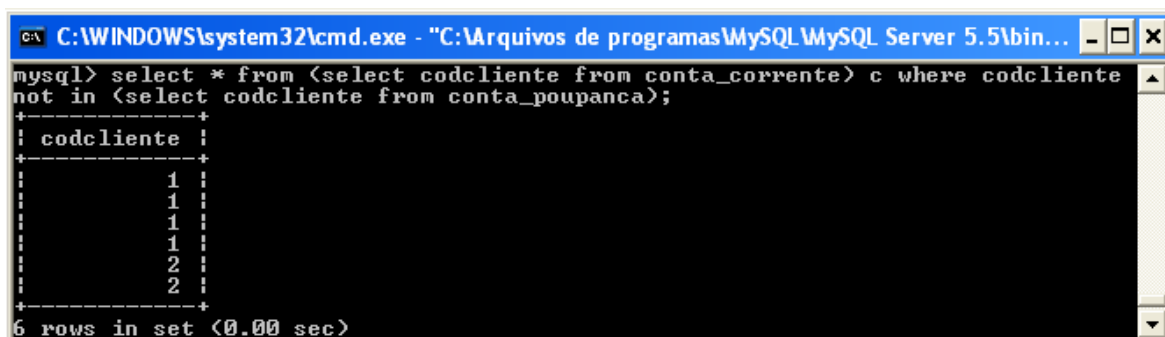
Podemos fazer o complemento de duas tabelas desde que elas sejam *união compatíveis*. Para isso utilizamos a cláusula *minus* ou *except* (dependendo do SGDB).

Por exemplo, para produzir uma relação com os códigos dos clientes que têm conta corrente mas não têm conta de poupança podemos usar a expressão a seguir.

```
select * from (select codcliente from conta_corrente) c minus (select codcliente from conta_poupanca);
```

O MySQL não implementa a cláusula *minus* nem a cláusula *except*. Ainda assim podemos fazer a operação de complemento usando o operador *not in*, como no exemplo a seguir.

```
select * from (select codcliente from conta_corrente) c where codcliente not in (select codcliente from conta_poupanca);
```



The screenshot shows a MySQL command window with the following query and result:

```
mysql> select * from (select codcliente from conta_corrente) c where codcliente not in (select codcliente from conta_poupanca);
```

codcliente
1
1
1
1
2
2

6 rows in set (0.00 sec)

6.7 Interseção de Tabelas

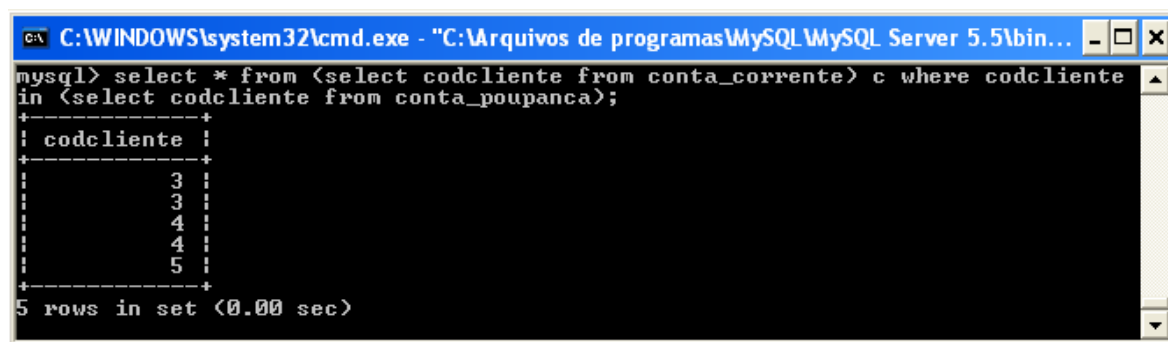
Podemos fazer a interseção de duas tabelas desde que elas sejam *união compatíveis*. Para isso utilizamos a cláusula *intersect*.

Por exemplo, para produzir uma relação com os códigos dos clientes que têm conta corrente e têm conta de poupança podemos usar a expressão a seguir.

```
select * from (select codcliente from conta_corrente) c intersect (select codcliente from conta_poupanca);
```

O MySQL não implementa a cláusula *intersect*. Ainda assim podemos fazer a operação de complemento usando o operador *in*, como no exemplo a seguir.

```
select * from (select codcliente from conta_corrente) c where codcliente in (select codcliente from conta_poupanca);
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select * from (select codcliente from conta_corrente) c where codcliente
in (select codcliente from conta_poupanca);
+-----+
| codcliente |
+-----+
|          3 |
|          3 |
|          4 |
|          4 |
|          5 |
+-----+
5 rows in set (0.00 sec)
```

Conclusões

Nesta aula vimos como utilizar a cláusula *select* para consultar os dados contidos nas tabelas. Nessa cláusula especificamos as tabelas envolvidas na consulta, os campos que serão exibidos no resultado e uma condição que serve para selecionar as linhas que aparecerão no resultado da consulta.

Apresentamos a cláusula *order by*, que permite ordenar o resultado da consulta, e a cláusula *distinct*, que serve para eliminar linhas repetidas. Aprendemos o que são *alias* e como definir um *subselect*.

Vimos também as operações de junção, junção natural de tabelas, união, complemento e interseção de tabelas

Aula 7 – Mais Consultas

Objetivos da Aula

Nesta aula, abordaremos consultas mais complexas do que aquelas estudadas na aula 6. Apresentaremos as funções agregadas *count*, *min*, *max*, *sum* e *avg* e as cláusulas *group by* e *having*. Abordaremos também a operação de junção externa de tabelas. Finalmente, veremos como criar tabelas e visões usando consultas.

Nos exemplos apresentados nessa aula utilizaremos as tabelas criadas no exercício 1 da aula 5.

7.1 Função Count

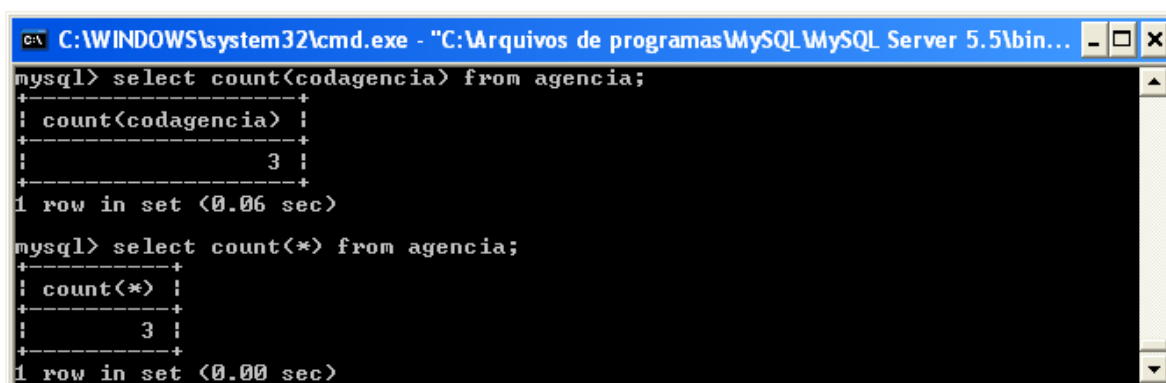
Podemos utilizar as funções *count*, *min*, *max*, *sum* e *avg* para obter informações adicionais das tabelas. Tais funções são chamadas de *funções agregadas* e recebem como argumento o nome de um campo no qual a função será aplicada.

A função *count* serve para contar a quantidade de valores *não nulos* num campo. Por exemplo, para contar a quantidade de agências, podemos usar a expressão:

```
select count(codagencia) from agencia;
```

Alternativamente, podemos utilizar um * (asterisco) como argumento da função *count*. Nesse caso será contada a quantidade de linhas da tabela. A consulta anterior também poderia ser feita com a expressão:

```
select count(*) from agencia;
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select count(codagencia) from agencia;
+-----+
| count(codagencia) |
+-----+
|          3        |
+-----+
1 row in set (0.06 sec)

mysql> select count(*) from agencia;
+-----+
| count(*)          |
+-----+
|          3        |
+-----+
1 row in set (0.00 sec)
```

Para contar a quantidade de valores *não nulos* **distintos** num campo utilizamos a função *count* com a cláusula *distinct* antes do nome do campo. Por exemplo, para contar a quantidade de agências que possuem contas de poupança podemos usar a expressão:

```
select count(distinct codagencia) from conta_poupanca;
```

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select count(distinct codagencia) from conta_poupanca;
+-----+
| count(distinct codagencia) |
+-----+
| 2 |
+-----+
1 row in set (0.00 sec)

mysql>
```

7.2 Função Min

A função *min* serve para obter o menor valor contido em um campo. Por exemplo, para descobrir qual o menor saldo de todas as contas correntes, podemos usar a expressão

:

```
select min(saldo) from conta_corrente;
```

Essa função também pode ser aplicada a campos do tipo data e do tipo caractere. Se aplicada a um campo do tipo data, será usada a ordem cronológica. Se utilizada num campo do tipo caractere, será usada a ordem alfabética.

```
select min(nomecliente) from cliente;
```

```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select min(saldo) from conta_corrente;
+-----+
| min(saldo) |
+-----+
| -300.00 |
+-----+
1 row in set (0.00 sec)

mysql> select min(nomecliente) from cliente;
+-----+
| min(nomecliente) |
+-----+
| Ana |
+-----+
1 row in set (0.00 sec)
```

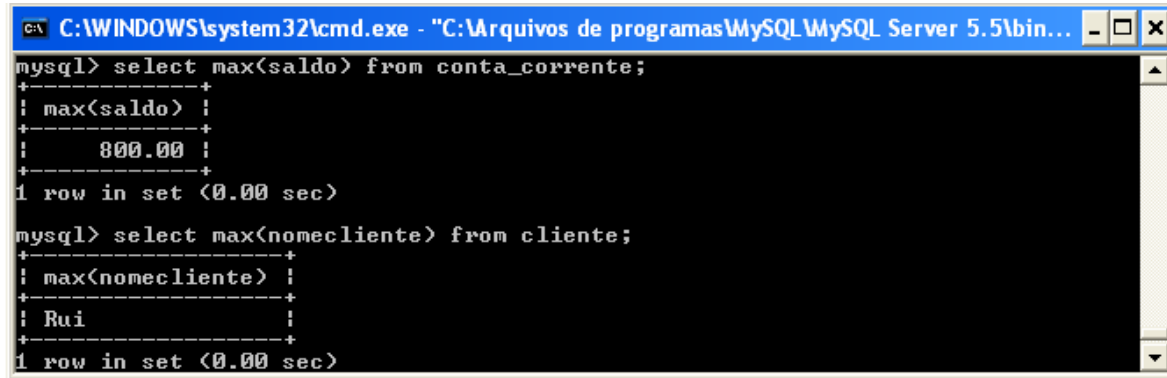
7.3 Função Max

A função *max* serve para obter o maior valor contido em um campo. Por exemplo, para descobrir qual o maior saldo de todas as contas correntes, podemos usar a expressão:

```
select max(saldo) from conta_corrente;
```

Essa função também pode ser aplicada a campos do tipo data e do tipo caractere. Se aplicada a um campo do tipo data, será usada a ordem cronológica. Se utilizada num campo do tipo caractere, será usada a ordem alfabética.

```
select max(nomecliente) from cliente;
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select max(saldo) from conta_corrente;
+-----+
| max(saldo) |
+-----+
|      800.00 |
+-----+
1 row in set <0.00 sec>

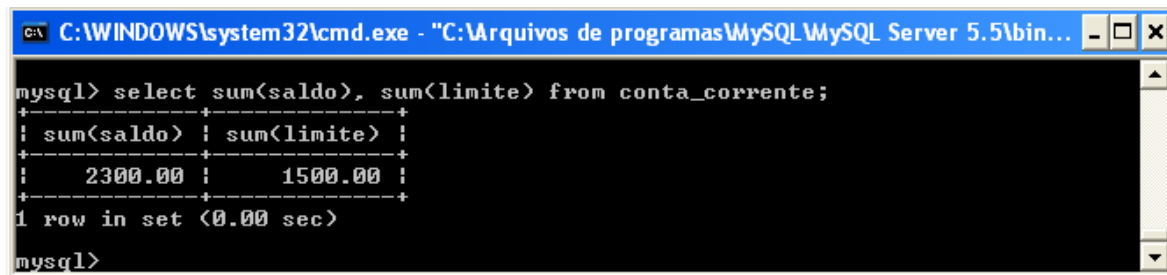
mysql> select max(nomecliente) from cliente;
+-----+
| max(nomecliente) |
+-----+
| Rui               |
+-----+
1 row in set <0.00 sec>
```

7.4 Função Sum

A função *sum* serve para somar os valores contidos em um campo. Por exemplo, para calcular a soma dos saldos de todas as contas correntes, podemos usar a expressão:

```
select sum(saldo), sum(limite) from conta_corrente;
```

Não faz sentido aplicar essa função a campos do tipo caractere. No entanto, essa função também pode ser aplicada a campos do tipo data, embora o resultado seja um tanto esquisito (experimente!).



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select sum(saldo), sum(limite) from conta_corrente;
+-----+-----+
| sum(saldo) | sum(limite) |
+-----+-----+
|    2300.00 |    1500.00 |
+-----+-----+
1 row in set <0.00 sec>

mysql>
```

7.5 Função Avg

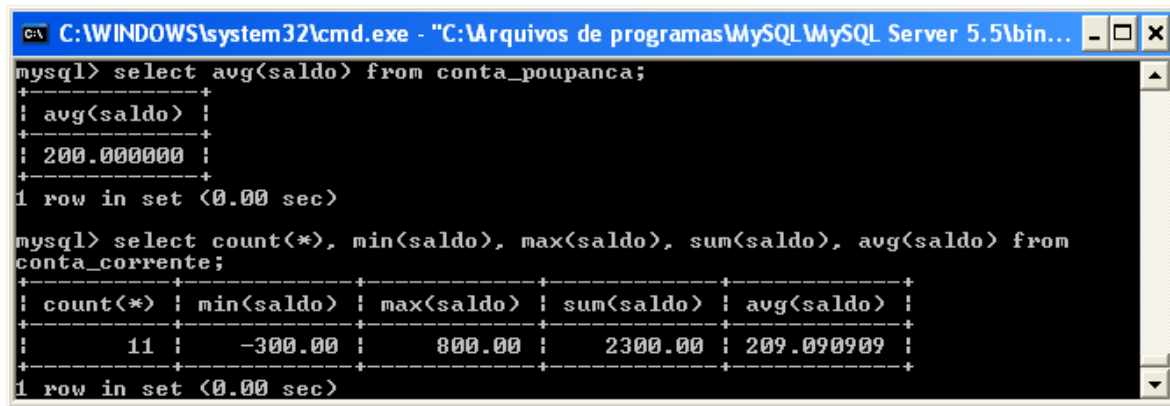
A função *avg* serve para calcular a média aritmética dos valores contidos em um campo. Por exemplo, para calcular o valor médio dos saldos de todas as contas de poupança, podemos usar a expressão:

```
select avg(saldo) from conta_poupanca;
```

Não faz sentido aplicar essa função a campos do tipo caractere, mas é possível aplicá-la a campos do tipo data.

Podemos utilizar todas as funções agregadas numa mesma consulta. Veja um exemplo:

```
select count(*), min(saldo), max(saldo), sum(saldo), avg(saldo) from conta_corrente;
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select avg(saldo) from conta_poupanca;
+-----+
| avg(saldo) |
+-----+
| 200.000000 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*), min(saldo), max(saldo), sum(saldo), avg(saldo) from
conta_corrente;
+-----+-----+-----+-----+-----+
| count(*) | min(saldo) | max(saldo) | sum(saldo) | avg(saldo) |
+-----+-----+-----+-----+-----+
| 11 | -300.00 | 800.00 | 2300.00 | 209.090909 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

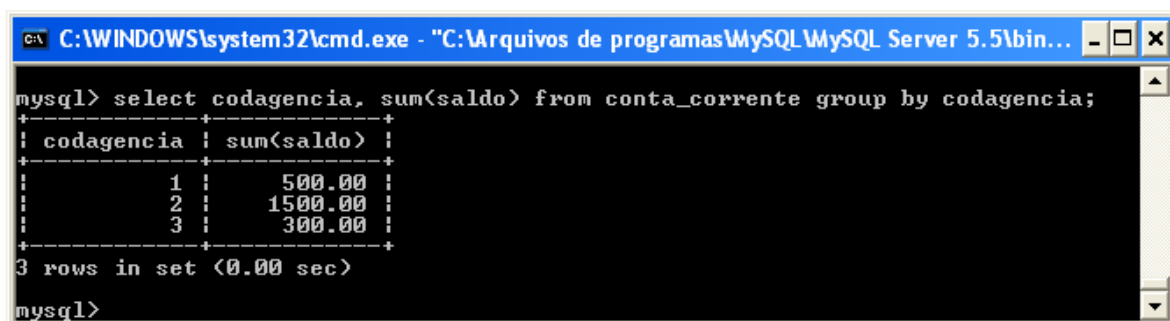
7.6 Cláusula Group by

Podemos agrupar as linhas de uma tabela que possuem os mesmos valores num conjunto de campos utilizando a cláusula *group by*. A sintaxe básica dessa cláusula é:

```
group by <lista de campos>
```

Em cada agrupamento podemos aplicar funções agregadas. Por exemplo, para saber qual a soma dos saldos das contas correntes de cada agência, basta agrupar as linhas da tabela *conta_corrente* por *agência* e aplicar a função *sum* ao campo *saldo*.

```
select codagencia, sum(saldo) from conta_corrente group by codagencia;
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select codagencia, sum(saldo) from conta_corrente group by codagencia;
+-----+-----+
| codagencia | sum(saldo) |
+-----+-----+
| 1 | 500.00 |
| 2 | 1500.00 |
| 3 | 300.00 |
+-----+-----+
3 rows in set (0.00 sec)

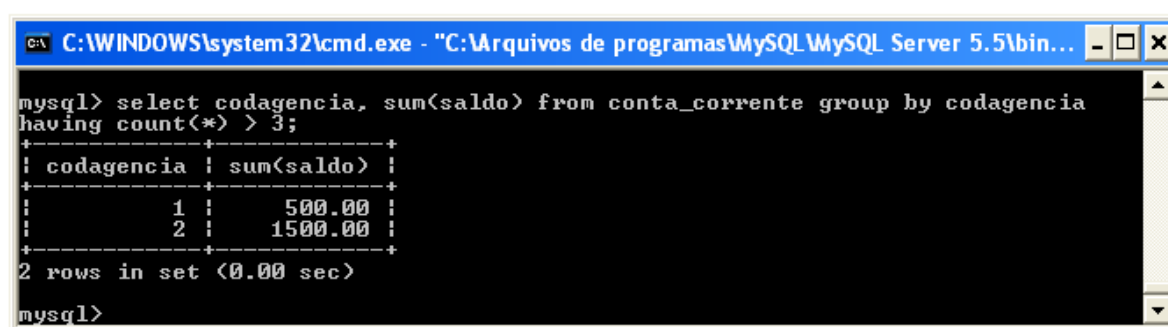
mysql>
```


7.7 Cláusula Having

Quando utilizamos a cláusula *group by*, podemos utilizar a cláusula *having* para definir uma condição que será utilizada para selecionar os agrupamentos que serão exibidos no resultado da consulta. Observe que a cláusula *where* define uma condição que será aplicada a cada linha da tabela, enquanto que a cláusula *having* define uma condição que será aplicada a cada agrupamento. Na cláusula *having* podemos utilizar funções agregadas, o que não pode ocorrer na cláusula *where*.

Por exemplo, para saber qual a soma dos saldos das contas correntes das agências que têm mais de duas contas podemos usar a seguinte expressão:

```
select codagencia, sum(saldo) from conta_corrente group by codagencia having count(*) > 2;
```



```
C:\WINDOWS\system32\cmd.exe - "C:\Arquivos de programas\MySQL\MySQL Server 5.5\bin...
mysql> select codagencia, sum(saldo) from conta_corrente group by codagencia
having count(*) > 3;
+-----+-----+
| codagencia | sum(saldo) |
+-----+-----+
| 1         | 500.00     |
| 2         | 1500.00    |
+-----+-----+
2 rows in set (0.00 sec)
mysql>
```

Conclusões

Nesta aula apresentamos as funções agregadas *count*, *min*, *max*, *sum* e *avg*. Tais funções servem para contar a quantidade de valores, determinar o menor valor, o maior valor, a soma dos valores e o valor médio de um campo.

Vimos a cláusula *group by* que serve para agrupar as linhas da tabela que têm determinados campos com valores iguais. Vimos também a cláusula *having* que permite especificar uma condição para selecionar os agrupamentos formados pelo *group by* que desejamos.

Referências Bibliográficas

- SUDARSHAN, S., SILBERSCHATZ, Abraham, KORTH, Henry F. *Sistemas de banco de dados*, 3ª ed. São Paulo: Makron Books, 1999.
- DATE, Christopher J. *Introdução a sistemas de banco de dados*. 7ª ed. RJ : Campus 2000.
- ELMASRI, Ramez, NAVATHE, Shamkant .B. *Sistemas de Banco de Dados*. 4ª ed. Pearson, 2005.
- SETZER, Valdemar W. *Bancos de dados : conceitos, modelos, gerenciadores, projeto lógico e físico*. 3ª ed. São Paulo : Edgar Blücher, 1989.

Sites Recomendados

- Banco de dados na Wikipédia:
http://pt.wikipedia.org/wiki/Banco_de_dados
- Download do MySQL:
<http://dev.mysql.com/downloads/>
- Tutorial do MySQL
<http://dev.mysql.com/doc/refman/4.1/pt/tutorial.html>
- Bancos de Dados Relacionais
http://pt.wikipedia.org/wiki/Banco_de_dados_relacional