5 – Introdução ao SQL

- Para criar e gerenciar um banco de dados, independente de qual seja, utiliza-se uma linguagem desenvolvida pela IBM, padrão para a maioria dos bancos de dados, denominada **SQL**.

- É por meio da linguagem SQL que praticamente todas as operações são realizadas em um banco de dados, desde a criação e alteração das tabelas e usuários, até o armazenamento e recuperação das informações.

- Para uma melhor organização, a linguagem SQL é dividida em duas categorias, os comandos **DDL**(Linguagem de Definição de Dados) e os comandos **DML**(Linguagem de Manipulação de Dados).

  DDL: São os comandos da linguagem SQL responsáveis pela criação de objetos no banco de dados, como tabelas,views, restrições, etc.
  DML: São comandos da linguagem SQL responsáveis pela inserção e manipulação de dados no banco de dados. Como exemplo, temos os comandos de inserir, alterar e excluir registros

5 – Introdução ao SQL

- A criação de um banco de dados, que é feito basicamente por meio de comandos SQL, dependendo do tamanho do modelo que se deseja implementar, pode ser um processo consideravelmente demorado e trabalhoso para os desenvolvedores.

- Como forma de acelerar o processo, e evitar que os desenvolvedores percam tempo digitando linhas e mais linhas de comandos SQL, existem diversas ferramentas gráficas que auxiliam tanto na criação como no gerenciamento e manipulação dos registros.

- A ferramenta não substitui a linguagem SQL, na verdade o usuário elabora graficamente os objetos que deseja criar, e a ferramenta automaticamente gera os códigos SQL necessários para criação destes objetos, isentando o desenvolvedor da elaboração e digitação destes comandos.

- Atualmente, existem ferramentas que possibilitam aos desenvolvedores efetuarem toda a modelagem do sistema. Feito isto, automaticamente a ferramenta gera os códigos SQL do modelo elaborado.

5 – Introdução ao SQL

- A linguagem SQL possui comandos específicos para criação, exclusão e alteração de toda a estrutura de um banco de dados, tais como tabelas, relacionamentos, restrições, etc.

- Os comandos para criação das estruturas do banco de dados pertencem ao grupo DDL (Linguagem de Definição de Dados), e são de grande importância, visto que eles são responsáveis por gerar o banco.

- Antes da criação de um banco de dados, é de extrema importância que todo um planejamento seja feito, por meio de analises e diagramas, evitando assim que muitas mudanças precisem ser feitas posteriormente.

- Veremos a partir de agora os principais comandos SQL para criação das estruturas do banco de dados.

5.1 – Criação de um Database

- O MySQL (assim como outros SGBD) possibilita que diversos banco de dados (databases) sejam armazenados de forma simultânea, ou seja, é possível que no mesmo servidor sejam armazenados dados de diversas organizações.

- O MySQL trata de forma independente cada um dos **databases**, sendo possível efetuar configurações e definir usuários específicos.

- Não existe um limite específico de número da databases para o MySQL. Contudo, quanto maior o número de dados e de usuários (principalmente de forma simultânea), mais recursos computacionais o servidor precisará ter.

- Para que um banco de dados possa ser implementado, o primeiro passo é a criação do database. Em outros SGBD, database também é conhecido como “**schema**”, mas na prática o coneito é o mesmo.

5.1 – Criação de um Database

- O quadro abaixo mostra o comando para criação de um database chamado “loja”, que será usado para implementar tabelas e outras estruturas.

- Uma vez que o database é criado, é preciso entrar na sua estrutura. Isso é feito por meio do comando “USE” seguido do nome do database.

CREATE DATABASE loja //criar o database loja

USE loja //entrar no database loja

DROP DATABASE loja //excluir o database loja

5.2 – Criação de Tabelas

- Em banco de dados relacionais os dados são armazenados em tabelas, portanto o desenvolvedor deve conhecer os comandos mais adequados para criação destas estruturas.

- Uma tabela é composta por linhas e colunas. O número e a ordem das colunas são fixas e possuem um nome para identificação.

- Cada coluna possui um tipo de dado, que restringe o conjunto de valores que podem ser atribuídos a coluna.

- Alguns dos tipos de dados frequentemente utilizados são o **integer** para números inteiros, **float** para fracionários e **varchar** para cadeias de caracteres.

5.2 – Criação de Tabelas

- A tabela abaixo ilustra alguns dos principais tipos de dados inteiros aceitos pelo MySQL:

Tipo de Dado	Descrição
TINYINT	É um número inteiro com ou sem signo. Com signo (como sinal de negativo) a margem de valores válidos é desde -128 até 127. Sem signo (sem sinal), a margem de valores é de 0 até 255
SMALLINT	Número inteiro com ou sem signo. Com signo a margem de valores válidos é desde -32768 até 32767. Sem signo, a margem de valores é de 0 até 65535.
MEDIUMINT	Número inteiro com ou sem signo. Com signo a margem de valores válidos é desde -8.388.608 até 8.388.607. Sem signo, a margem de valores é de 0 até 16777215.
INT	Número inteiro com ou sem signo. Com signo a margem de valores válidos é desde -2147483648 até 2147483647. Sem signo, a margem de valores é de 0 até 429.496.295
BIGINT	Número inteiro com ou sem signo. Com signo a margem de valores válidos é desde -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807. Sem signo, a margem de valores é de 0 até 18.446.744.073.709.551.615.

5.2 – Criação de Tabelas

- A tabela abaixo ilustra alguns dos principais tipos de dados decimais aceitos pelo MySQL:

Tipo de Dado	Descrição
FLOAT	Os valores válidos vão desde -1.175494351E-38 até 175494351E-38 ou de 0 até 3.402823466E+38 (Não signo). Não indicado para armazenamento de moeda devido a perda de precisão.
DOUBLE	Os valores permitidos vão desde -1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308.
DECIMAL	Tipagem indicada para armazenamento de valores monetários. Ex: decimal(10,2).

5.2 – Criação de Tabelas

- A tabela abaixo ilustra alguns dos principais tipos dedados de data e hora aceitos pelo MySQL:

Tipo de Dado	Descrição
DATE	A margem de valores vai desde o 1 de Janeiro de 1001 ao 31 de dezembro de 9999. O formato de armazenamento é de ano-mês-dia.
DATETIME	Combinação de data e hora. A margem de valores vai desde o 1 de Janeiro de 1001 às 0 horas, 0 minutos e 0 segundos ao 31 de Dezembro de 9999 às 23 horas, 59 minutos e 59 segundos. O formato de armazenamento é de ano-mês-diahoras:minutos:segundos
TIMESTAMP	Combinação de data e hora. A margem vai desde o 1 de Janeiro de 1970 ao ano 2037. O formato de armazenamento depende do tamanho do campo

5.2 – Criação de Tabelas

- A tabela abaixo ilustra alguns dos principais tipos de dados de texto aceitos pelo MySQL:

Tipo de Dado	Descrição
CHAR(n)	Armazena uma cadeia de longitude fixa. A cadeia poderá conter desde 0 até 255 caracteres.
VARCHAR(n)	Armazena uma cadeia de longitude variável. A cadeia poderá conter desde 0 até 255 caracteres.
TEXT	Um texto com um máximo de 65535 caracteres

5.2 – Criação de Tabelas

- Para se criar uma tabela, utiliza-se o comando **CREATE TABLE**, seguido do nome da tabela, dos campos e seus respectivos tipos.

- Para criar uma tabela, utiliza-se a estrutura abaixo:

```
CREATE TABLE<nome da tabela>(
<nome do campo 1> <tipo> [valor padrão] [nulo],
<nome do campo2><tipo> [valor padrão] [nulo],
...
<nome do campoN><tipo> [valor padrão] [nulo]
);
```

5.2 – Criação de Tabelas

1. **<nome da tabela>**: Nome escolhido para tabela. Ex: Cliente, Fornecedor, etc.

2. **<nome do campo>**: Nome escolhido para o campo da tabela. Ex: Nome, Cidade, RG, CPF, Telefone, etc.

3. **<tipo>:**Tipo de dados que o campo irá comportar. Ex: Interger, Varchar, etc.

4. **[Valor Padrão]**: Caso um campo tenha tendência a ter um valor fixo, embora esse valor possa mudar esporadicamente, pode-se determinar um valor padrão para ele.

5. **[nulo]:**Define se o campo é de preenchimento obrigatório ou não. Por isso seus valores possíveis são **NULL** e **NOT NULL**.

5.2 – Criação de Tabelas

- O quadro abaixo ilustra o código para criação de uma tabela de clientes:

```
CREATE TABLE `cliente` (
  `id_cliente` int(11) NOT NULL,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50),
  `id_cidade` int(11) DEFAULT “1” NOT NULL,
  `email` varchar(50)
)
```

5.2 – Criação de Tabelas

- Depois que a tabela está criada, é possível verificar a sua estrutura por meio do comando **DESCRIBE**.

- Alguns bancos aceitam também a sua abreviatura, que seria somente **DESC**.

Sua estrutura é**:**

DESCRIBE<nome da tabela>

Exemplo:

DESCRIBE cliente

ou

DESC cliente.

5.3 – Renomear Tabelas

- Devido a erros de digitação, ou até mesmo por necessidade de uma reestruturação do banco de dados, pode ser necessário mudar o nome de uma ou mais tabelas.

- Para renomear uma tabela, utiliza-se o comando **RENAME TABLE**, como ilustra a sintaxe abaixo:

Sua estrutura é**:**

RENAME TABLE<nome da tabela> **TO** <novo nome da tabela>

Exemplo:

RENAME TABLE cliente **TO** clientes**;**

5.4 – Exclusão de Tabelas

- Excluir uma tabela exige muito cuidado, pois além da tabela, todos os dados também serão excluídos.

- Para excluir uma tabela, utiliza-se o comando **DROP TABLE** seguido do nome da tabela a ser excluída.

Sua estrutura é**:**

DROP TABLE <nome da tabela>

Exemplo:

DROP TABLE cliente

5.5– Alteração de Campos nas Tabelas

- Depois que a tabela está criada, pode ser necessário que algumas alterações em sua estrutura sejam realizadas, tais como inclusão de novos campos, troca do nome ou do tipo do campo, etc.

- Essas alterações podem ser feitas por meio do comando **ALTER TABLE**, seguidos dos comandos que determinam o tipo de operação.

- Os tópicos abaixo irão ilustrar os tipos de operações possíveis:

1. Inclusão de novos campos

2. Alteração do nome do campo

3. Alteração do tipo do campo.

5.5.1– Inclusão de Novos Campos nas Tabelas

- Para se incluir um campo em uma tabela que já foi criada, utiliza-se a seguinte sintaxe:

ALTER TABLE<nome da tabela>**ADD**<nome do campo> <tipo> [valor padrão] [nulo]

Exemplo:

ALTER TABLE cliente **ADD COLUMN** tel **VARCHAR(11) NOT NULL;**

5.5.2– Alteração de Campos nas Tabelas

- Para alterar um campo de uma tabela, utiliza-se a seguinte sintaxe::

ALTER TABLE<nome da tabela>**MODIFY**<nome do campo> <tipo> [valor padrão] [nulo]

Exemplo:

ALTER TABLE cliente **MODIFY** tel **VARCHAR(10) NOT NULL;**

5.5.3– Renomear Campos das Tabelas

- Para renomear um campo de uma tabela, utiliza-se o comando ALTER TABLE juntamente com o CHANGE, como na sintaxe abaixo:

ALTER TABLE<nome da tabela>**CHANGE**<nome do campo> <novo nome> <novo tipo>

Exemplo:

ALTER TABLE cliente **CHANGE** tel telefone **VARCHAR(11);**

Observação:A sintaxe acima é utilizada no MySQL, nos demais banco de dados,essa sintaxe é um pouco diferente, pois utiliza-se o comando **RENAME COLUMN**.

Ex:**ALTER TABLE** cliente **RENAME COLUMN** tel **TO** telefone.

5.5.4– Exclusão de Campos das Tabelas

- Para excluir um campo de uma tabela, utiliza-se o comando ALTER TABLE juntamente com o DROP, como na sintaxe abaixo:

ALTER TABLE <nome da tabela> **DROP COLUMN** <nome do campo>

Exemplo:

ALTER TABLE Cliente **DROP COLUMN** telefone

5.6 – Chave Primária

- A **Chave Primária (Primary Key)** é um objeto de extrema importância em um banco de dados, pois ela é responsável pela identificação dos registros de uma tabela.

- A Chave Primária deve ser formada por um campo cujos valores nunca se repetem, desta forma, será possível identificar cada um dos registros pelo valor de sua chave. Toda tabela deve possuir uma chave primária.

- Um exemplo de campo candidato a ser chave primária é o CPF (ou CNPJ no caso de pessoa jurídica) de um cliente, visto que esse valor é único e nunca se repete.

- Muitos projetistas de banco de dados costumam criar um campo especifico para inserção da chave primária. Ex: Para a tabela de Clientes, cria-se um campo denominado **id_cliente**, cujo valor pode ser um número inteiro qualquer, desde que este nunca se repita.

- É possível definir a chave primária de duas formas: No momento de criação da tabela, ou após a criação da tabela.

5.6 – Chave Primária

- A tabela abaixo exemplifica a utilização da chave primária. O campo “id_cliente” está definido como chave primária da tabela.

- Cada um dos clientes receberá um valor único para a coluna chave primária (é possível atribuir o valor do código manualmente ou gerar de forma automática), possibilitando que o registro seja identificado pelo seu código e não pelo nome (que pode se repetir entre os vários clientes).

id_cliente	nome	endereco	id_cidade	telefone
1	Gabriel Jesus	Rua Juca, 45	1	44999566878
2	Fernando Pras	Rua XV, 55	1	44999564378
3	Dudu da Silva	Rua Guara, 45	2	44999066878
4	Felipe Melo	Rua Brasil, 52	3	23999566878
5	Roger Guedes	4	*NULL*	47988566878
6	William Bigode	Rua 19, 21	5	25999566878
7	Miguel Borja	Rua João, 40	6	44999566878
8	Zé Roberto da Silva	Rua XV, 53	9	44599566870
9	Jean da Silva	Rua Atlantica, 9	1	77699566878

5.6.1 – Chave Primária na Criação da Tabela

- Para criação da chave primária no momento de criação da tabela, utiliza-se o seguinte comando:

```
CREATE TABLE<nome da tabela>(
<nome do campo 1> <tipo> [valor padrão] [nulo],
<nome do campo2><tipo> [valor padrão] [nulo],
...
<nome do campoN><tipo> [valor padrão] [nulo],
CONSTRAINT<nome da restrição>PRIMARY KEY(<campo>)
);
```

5.6.1 – Chave Primária na Criação da Tabela

- Não é obrigatório a definição de um nome para a chave primária.

- Um exemplo de definição de chave primária no momento de criação da tabela é dada pelo código abaixo:

```
CREATE TABLE `cliente` (
  `id_cliente` int(11) NOT NULL,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50),
  `id_cidade` int(11) DEFAULT “1” NOT NULL,
  `email` varchar(50),
CONSTRAINT PK_Cliente PRIMARY KEY (id_cliente)
);
```

5.6.1 – Chave Primária na Criação da Tabela

- Alguns bancos de dados também aceitam a declaração da chave primária sem o nome da restrição. Basta colocar o comando **Primary Key** na frente do campo que será chave primária, como no exemplo abaixo:

```
CREATE TABLE `cliente` (
  `id_cliente` int(11) PRIMARY KEY,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50),
  `id_cidade` int(11) DEFAULT “1” NOT NULL,
  `email` varchar(50)
);
```

5.6.2 – Chave Primária em Tabelas Definidas

- Existem comandos SQL para a definição da chave primária mesmo após a tabela estar criada.

- Portanto, mesmo que o programador esqueça de definir uma chave primária para a tabela, não será necessário que a mesma seja apagada e recriada.

- Para definição da chave primária após a criação da tabela, utiliza-se a seguinte sintaxe:

ALTER TABLE <nome da tabela> **ADD**

(CONSTRAINT<nome da restrição>**PRIMARY KEY (**<campo>**));**

5.6.2 – Chave Primária em Tabelas Definidas

- Como exemplo, vamos supor que a tabela cliente tenha sido criada sem a chave primária.

- Neste caso, o comando abaixo iria criar a chave mesmo após a tabela cliente ter sido criada.

ALTER TABLE cliente **ADD (CONSTRAINT** PK_Cliente **PRIMARY KEY (**id_cliente**));**

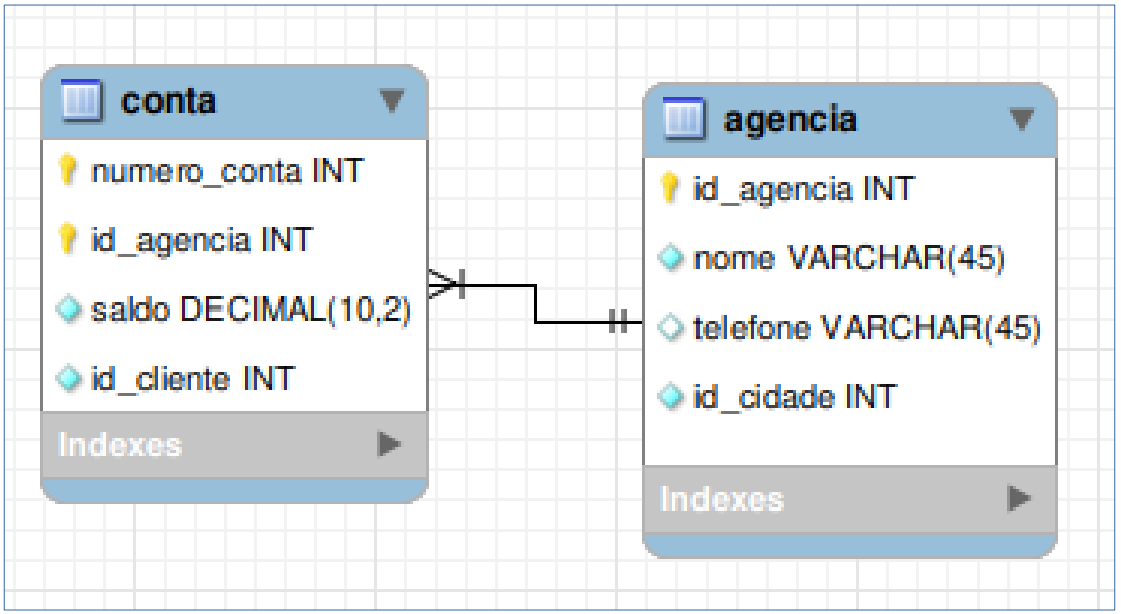
ou

ALTER TABLE cliente **ADD (CONSTRAINT PRIMARY KEY (**id_cliente**));**

5.6.3 – Chave Composta

- É possível definir mais de um campo como chave primária de uma tabela. Essa técnica é denominada **chave composta**.

- Vamos exemplificar o conceito de chave composta por meio de um exemplo. Imaginemos duas tabelas, uma de Agência Bancária e outra de Conta Corrente, como ilustra o diagrama abaixo:



5.6.3 – Chave Composta

- A figura abaixo ilustra um exemplo do uso de chave composta. Os clientes 4 e 9 possuem o mesmo número de conta corrente, mas em agências diferentes. Assim como os clientes 6 e 1.

- Quando existe chave composta, somente se os valores de todos os campos chaves forem iguais é que a inserção será proibida.

conta			
numero_conta	id_agencia	id_cliente	saldo
1000	1	2	3000
1001	1	5	1000
1001	4	6	500
1000	3	5	700
1002	2	4	4000

agencia			
id_agencia	nome	id_cidade	telefone
1	Agência UTFPR	1	4230316875
2	Agência Centro	1	4230316564
3	Agência Zona 1	2	4430363987
4	Agência UEL	3	4336399897

5.6.3 – Chave Composta

- Como é possível observar na tabela “conta”, existem dois campos definidos como chave primária, são eles: numero_conta e id_agencia.

- Se a chave primária estivesse apenas no campo “numero_conta”, nenhum cliente, mesmo que este tivesse aberto a sua conta em uma outra agência, poderia ter o mesmo número da conta corrente.

- Como a chave primária está nos dois campos, pode existir dois clientes com o mesmo número de conta, desde que em agências diferentes. No quadro do exemplo, a conta número 1001 existe na agência número 1 e 4.

- Assim como podem existir clientes na mesma agência, desde que possuam número de conta diferentes.

5.6.3 – Chave Composta

- A chave composta pode ser declarada no momento de criação da tabela, ou após a criação da mesma.

- A estrutura abaixo ilustra a sintaxe para declaração de chave primária no momento de criação da tabela:

```
CREATE TABLE<nome da tabela>(
<nome do campo 1> <tipo> [valor padrão] [nulo],
<nome do campo2><tipo> [valor padrão] [nulo],
...
<nome do campoN><tipo> [valor padrão] [nulo],
CONSTRAINT<nome da restrição>PRIMARY KEY(<campo 1>,<campo2>, <campo n> )
);
```

5.6.3 – Chave Composta

- O exemplo abaixo ilustra a criação de uma tabela com 2 chaves primárias:

```
CREATE TABLE Conta (
numero_conta INTEGER,
id_agencia INTEGER,
id_cliente INTEGER,
saldo DECIMAL(10,2),
CONSTRAINT PK_Conta PRIMARY KEY (numero_conta, id_agencia)
);
```

5.6.3 – Chave Composta

- É possível criar uma chave composta mesmo após a tabela ter sido criada, para isso utiliza-se a sintaxe abaixo:

ALTER TABLE<nome da tabela>**ADD**

(CONSTRAINT<nome da restrição>**PRIMARY KEY**(<campo 1>, <campo 2>, <campo n>**));**

5.6.3 – Chave Composta

- O exemplo abaixo ilustra a criação de uma chave composta para a tabela de contas, caso esta não seja declarada no momento de geração da tabela:

```
ALTER TABLE Conta ADD CONSTRAINT PK_Conta PRIMARY KEY(numero_conta, id_agencia)

ou

ALTER TABLE Conta ADD CONSTRAINT  PRIMARY KEY(numero_conta, id_agencia)
```

5.6.4 – Exclusão de Chave Primária

- Para excluir a chave primaria de uma tabela, os seguintes comandos são usados:

ALTER TABLE <nome da tabela> **DROP PRIMARY KEY;**

Exemplo:

ALTER TABLE cliente **DROP PRIMARY KEY**

5.7 – Relacionamento entre Tabelas

- As tabelas tem a função de subdividir os assuntos existentes em uma base de dados.

- Essa subdivisão das informações somente se torna útil se pudermos cruzar os dados existentes nas tabelas, formando uma base de dados única e consistente.

- Dentro do contexto de banco de dados, podemos dizer que relacionamento entre as tabelas possibilita cruzar os dados das mesmas, garantindo a integridade das informações.

- Como exemplo, vamos utilizar a tabela de clientes, onde existe o campo id_cidade, responsável por indicar a cidade que o cliente reside.

- Não teria sentido cadastrarmos na tabela de clientes um valor para id_cidade que não exista na tabela de cidades, pois estaríamos quebrando a integridade do banco de dados.

5.7 – Relacionamento entre Tabelas

- Para garantir esse tipo de integridade, é necessário relacionar as tabelas de clientes e de cidades. Desta forma, toda vez que um cliente for inserido, o próprio SGBD verificará se na chave estrangeira “id_cidade” da tabela de cliente foi inserido um valor existente na tabela de cidade, como ilustra o exemplo abaixo:

id_cidade	nome	id_estado
1	Guarapuava	1
2	Curitiba	1
3	Londrina	1
4	Maringá	1
5	São José dos Campos	2
6	São Paulo	2
7	Campinas	2
8	São José dos Campos	2
9	Florianópolis	3
10	Blumenau	3
11	Ponta Grossa	*NULL*

id_cliente	nome	endereco	id_cidade	telefone
1	Gabriel Jesus	Rua Juca, 45	1	44999566878
2	Fernando Pras	Rua XV, 55	1	44999564378
3	Dudu da Silva	Rua Guara, 45	2	44999066878
4	Felipe Melo	Rua Brasil, 52	3	23999566878
5	Roger Guedes	4	*NULL*	47988566878
6	William Bigode	Rua 19, 21	5	25999566878
7	Miguel Borja	Rua João, 40	6	44999566878
8	Zé Roberto da Silva	Rua XV, 53	9	44599566870
9	Jean da Silva	Rua Atlantica, 9	1	77699566878

5.7 – Relacionamento entre Tabelas

- No caso das tabelas anteriores não estarem relacionadas, a tabela de cliente aceitará qualquer valor inteiro no campo “id_cidade”, inclusive valores que não existem.

- Se estiverem relacionadas, somente serão aceitos no campo “id_cidade” da tabela de cliente valores que são existentes na tabela de cidade.

- O relacionamento se dá sempre da tabela filha para a tabela pai. No exemplo anterior, a tabela de cidade é a tabela pai, pois não é possível cadastrar um cliente se não existir uma cidade cadastrada para a mesma.

- Porém, é possível cadastrar uma cidade sem que exista nenhum cliente cadastrado. Logo, a tabela de cliente é filha nesse relacionamento.

5.7 – Relacionamento entre Tabelas

- Para identificar qual é a tabela pai, basta verificar qual tabela fornece sua chave primária para as demais tabelas envolvidas no relacionamento. Esta tabela será a entidade forte do relacionamento (tabela pai).

- A tabela que recebe uma chave primária de outra tabela (que se torna uma chave estrangeira) é a entidade fraca (tabela filha) do relacionamento.

- E o relacionamento é sempre da entidade fraca para entidade forte (tabela filha para tabela pai). No caso do exemplo anterior, o relacionamento será da tabela de cliente para a tabela de cidade.

- O relacionamento pode ser feito no momento de criação da tabela filha, ou após a criação da tabela filha e pai, cabendo ao desenvolvedor decidir qual técnica deseja usar.

5.7.1 – Relacionamento na Criação da Tabela

- Para criar o relacionamento no momento de criação da tabela, utiliza-se a sintaxe abaixo:

```
CREATE TABLE <nome da tabela>(
<nome do campo 1> <tipo> [valor padrão] [nulo],
<nome do campo2><tipo> [valor padrão] [nulo],
...
<nome do campoN><tipo> [valor padrão] [nulo],
CONSTRAINT<nome da restrição>FOREIGN KEY(<chave estrangeira>)
REFERENCES<tabela externa>(<chave primária da tabela externa>)
);
```

5.7.1 – Relacionamento na Criação da Tabela

- Como exemplo, no momento de criação da tabela de Conta Corrente, temos o seu relacionamento sendo feito com a tabela de Agência:

```
CREATE TABLE `cliente` (
  `id_cliente` int(11) PRIMARY KEY,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50) DEFAULT “Rua”,
  `id_cidade` int(11) NOT NULL,
  `email` varchar(50),
   CONSTRAINT FK_Cidade FOREIGN KEY(id_cidade) REFERENCES cidade(id_cidade)
);
```

5.7.2 – Relacionamento para Tabelas Definidas

- É possível relacionar as tabelas mesmo após estas já terem sido criadas no banco de dados, basta utilizar a sintaxe abaixo:

```
ALTER TABLE<nome da tabela>ADD
CONSTRAINT<nome da restrição>FOREIGN KEY(<chave estrangeira>)
REFERENCES<tabela externa>(<chave primária da tabela externa>)
);
```

5.7.2 – Relacionamento para Tabelas Definidas

- Como exemplo, podemos utilizar o relacionamento entre as tabelas de Conta Corrente e de Agência.

```
CREATE TABLE `cliente` (
  `id_cliente` int(11) PRIMARY KEY,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50) DEFAULT “Rua”,
  `id_cidade` int(11) NOT NULL,
  `email` varchar(50),
);
```

```
ALTER TABLE cliente ADD (CONSTRAINT FK_Cidade
FOREIGN KEY(id_cidade) REFERENCES cidade(id_cidade)
);
```

5.8 – Auto Incremento

- Não seria viável deixar a cargo do usuário inserir o valor de um campo chave primária manualmente, pois este não saberia se esse valor seria único ou não.

- Em muitos casos, o usuário final nem irá visualizar o código (chave primária de uma tabela), visto que ela é usada para a estruturação do banco de dados, não sendo utilizada visivelmente no sistema.

- A grande maioria dos bancos de dados possuem recursos para inserir automaticamente valores na chave primária.

- O mysql possibilita a inserção automática por meio do comando **AUTO_INCREMENT.**

- É permitida apenas uma coluna com auto numeração por tabela, e está coluna deve ser chave primária.

5.8 – Auto Incremento

- O código abaixo mostra um exemplo de criação de tabela com chave primária auto numerada:

```
CREATE TABLE `cliente` (
 `id_cliente` int(11) PRIMARY KEY AUTO_INCREMENT,
 `nome` varchar(50) NOT NULL,
 `endereco` varchar(50) DEFAULT “Rua”,
 `id_cidade` int(11) NOT NULL,
 `email` varchar(50)
);
```

Exercícios

- Crie o banco de dados comercial abaixo: