Conteúdo

Introdução

O que pode ser considerado um dado? Um dado é qualquer fragmento de informação. No mundo real, um dado pode ser simplesmente uma cadeira ou uma mesa, por exemplo. Na computação, um dado é um pedaço de informação, ou seja, um conjunto de caracteres que formam uma unidade significativa.

Banco de dados nada mais é que um conjunto de dados, que estão armazenados (salvos). Podemos citar como princípios básicos de bando de dados:

- Criação de um ambiente centrado em dados, no qual os dados de uma empresa possam ser considerados de fato como um significativo recurso corporativo. Um recurso é a habilidade de compartilhar dados entre os que estão dentro como fora da empresa;
- Realizar a integração dos dados ao mesmo tempo que faz seu armazenamento de forma não redundante;
- Estabelecimento de uma ambiente que gerencie determinadas questões de controle de dados, como segurança de dados, cópias se segurança;
- Ambiente com alto grau de independência de dados.

Em se tratando de banco de dados, temos o que definimos como entidade, que é um objeto representado na realidade. Por exemplo, uma pessoa, é uma entidade, sendo que esta possui um conjunto de características que a definem. Cada característica é definida, em banco de dados, como atributo. E cada entidade deve ser única.

Estrutura de um banco de dados

Podemos dividir a estrutura das informações contidas em um banco de dados em campos e registros. Campos são os atributos definidos nas colunas, e registros são os dados contidos nas linhas.

Na tabela abaixo veja um exemplo prático:

Matricula	Nome	Endereço
001	Maria da Silva	Rua A, 244
002	Mariana Souza	Rua C, 28
003	Tiago Souza	Rua M, 122

Os campos são: Matricula, Nome e Endereço.

Já os registros são: as linhas que começam em 001, 002 e 003.

Modelagem de dados

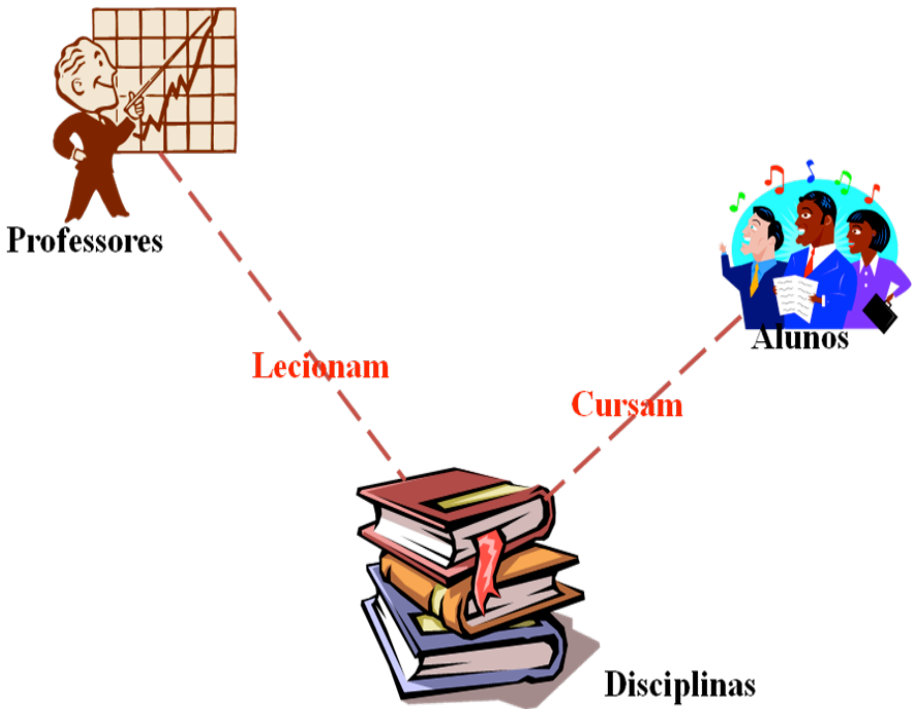
O modelo apresentado a seguir para criar os diagramas para a modelagem de um banco de dados é conhecido como entidade-relacionamento, ou simplesmente modelo E-R. Este modelo define as entidades (e seus atributos) que irão compor o banco de dados e o relacionamento entre essas entidades. Esta etapa é denominada como modelagem conceitual.

Pessoa
Matricula
Nome
Endereço

No exemplo, o nome da entidade é Pessoa e seus atributos são: Matricula, Nome e Endereço.

Relacionamentos

Os relacionamentos especificam como acontecerá a integração entre as entidades.



Relacionamentos Binários

Um relacionamento binário refere-se ao relacionamento entre dois tipos de entidades.

Cardinalidade

A cardinalidade representa o número máximo de entidades que podem estar envolvidas em um determinado relacionamento.

Modalidade

A modalidade representa o número mínimo de ocorrências de entidades que podem estar envolvidas no relacionamento.

Relacionamento binário Um-para-Um (1-1)

Significa que uma única ocorrência de um tipo de entidade pode estar associada a uma única ocorrência do outro tipo de entidade e vice-versa.

Relacionamento binário Um-para-Muitos (1-*)

Refere-se que uma entidade pode ser relacionada uma ou mais vezes, enquanto que a outra entidade da relação só poderá se associar uma vez.

Relacionamento binário Muitos-para-Muitos (*-*)

Já este relacionamento significa que pode ocorrer uma ou mais associações entre as entidades.

Entidade associativa

Em relacionamento muitos-para-muitos temos a presença das entidades associativas, que são aquelas que possuem atributos para definir o relacionamento, como forma de tornar esse relacionamento único.

Relacionamentos Unários e Ternários

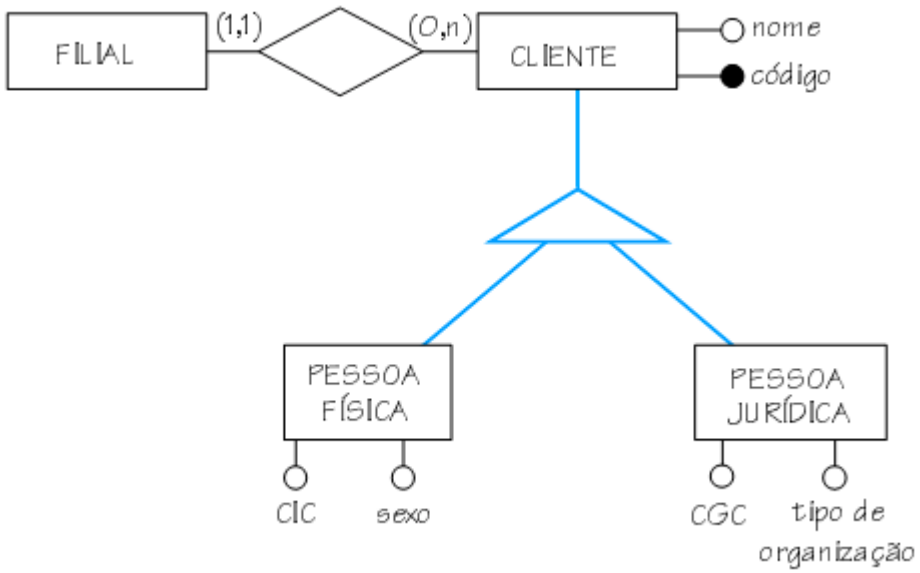
Os relacionamentos unários referem-se os relacionamentos de entidades do mesmo tipo, enquanto que os relacionamentos ternários envolvem três tipos diferentes de entidades. São relacionamentos que ocorrem com pouca freqüência.

Sua cardinalidade e modalidade são representadas da mesma forma que o relacionamento binário, ou seja, um-para-um, um-para-muitos e muitos-para-muitos.

Generalização/Especialização

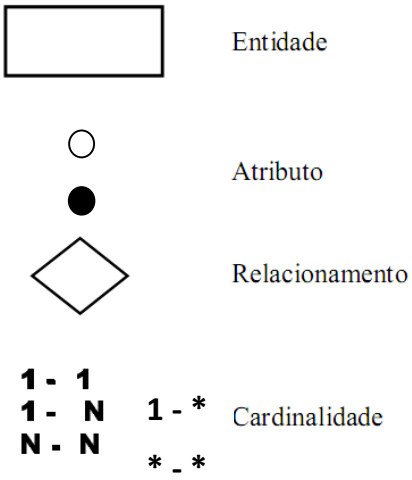
Através deste conceito é possível atribuir propriedades particulares a um subconjunto das ocorrências (especializadas) de uma entidade genérica. O símbolo para representar generalização/especialização é um triângulo isóscele. A generalização/especialização mostrada na figura abaixo expressa que a entidade cliente é dividida em dois subconjuntos, as entidades pessoa física e pessoa jurídica cada um com propriedades próprias.

Associada ao conceito de generalização/especialização a idéia de herança de propriedades. Herdar propriedades significa que cada ocorrência da entidade especializada possui, além de suas próprias propriedades (atributos, relacionamentos e generalizações/especializações), também as propriedades da ocorrência da entidade genérica correspondente. Assim, a entidade pessoa física possui, além de seus atributos particulares, CIC e sexo, também todas as propriedades da ocorrência da entidade cliente correspondente, ou seja, os atributos nome e código, o seu identificador (atributo código), bem como o relacionamento com a entidade filial. Resumindo, o diagrama expressa que toda pessoa física tem como atributos nome, código, CIC e sexo, é identificada pelo código e está obrigatoriamente relacionada a exatamente uma filial. Da mesma maneira, toda pessoa jurídica tem como atributos nome, código, CGC e tipo de organização, é identificada pelo código e está obrigatoriamente relacionada a exatamente uma filial.

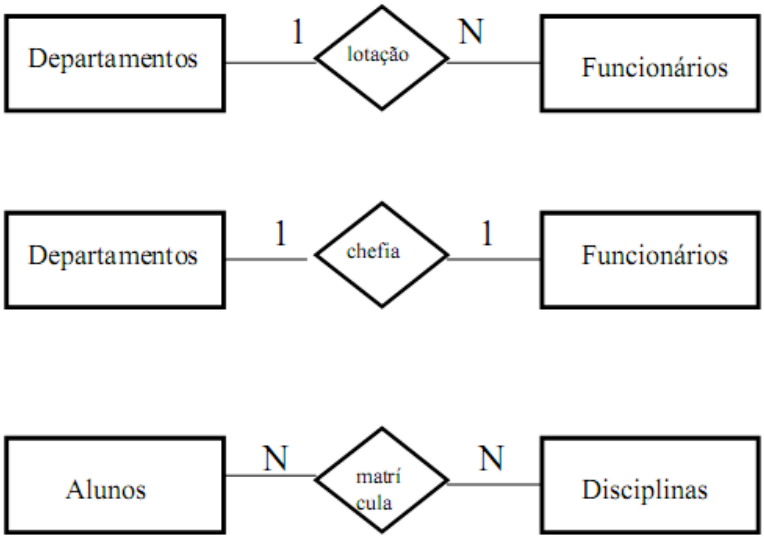


Representação gráfica para a modelagem

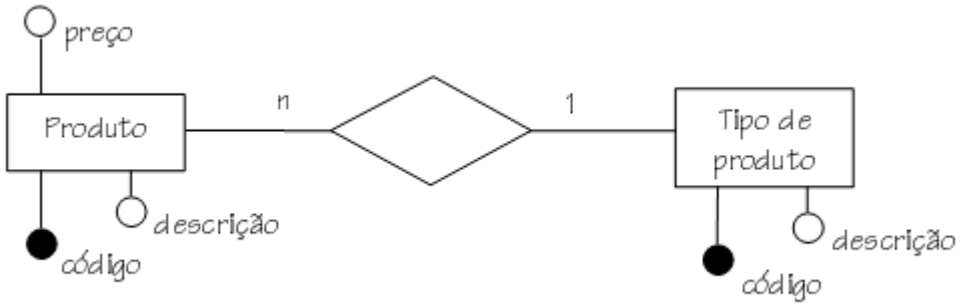
Abaixo a representação de forma gráfica para modelagem conceitual de uma estrutura de banco de dados.



A figura abaixo representa um modelo inicial da modelagem, informando as entidades, os relacionamentos e suas cardinalidades.

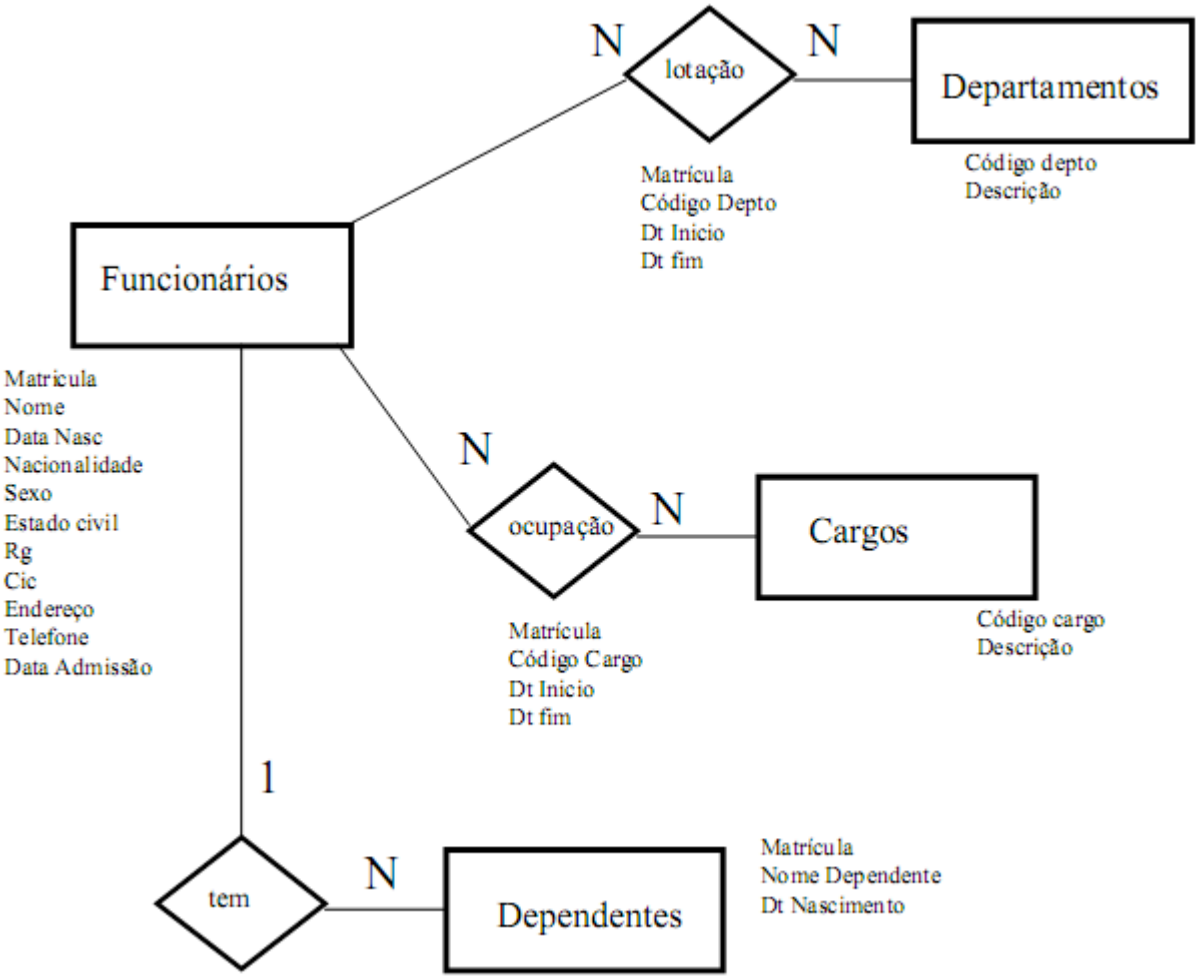


A imagem abaixo mostra um exemplo de modelagem com os atributos para as entidades.



A seguir, um exemplo completo de modelagem, a partir das necessidades apresentadas na figura abaixo:

Dados Cadastrais do Funcionário		
Matrícula:	Nome:	
Data Nasc:	Nacionalidade:	Sexo:
Est.Civil:	RG:	CIC:
Endereço:		Telef:
Data Admissão:		
Cargos Ocupados		
Cargo:	Dt Início:	Dt Fim:
Cargo:	Dt Início:	Dt Fim:
Departamentos de lotação		
Depto:	Dt Início:	Dt Fim:
Depto:	Dt Início:	Dt Fim:
Dependentes		
Nome:		Data Nasc:
Nome:		Data Nasc:


N
lotação
N
Departamentos
Matrícula
Código Depto
Dt Inicio
Dt fim
Código depto
Descrição
Funcionários
Matricula
Nome
Data Nasc
Nacionalidade
Sexo
Estado civil
Rg
Cic
Endereço
Telefone
Data Admissão
N
ocupação
N
Cargos
Matrícula
Código Cargo
Dt Inicio
Dt fim
Código cargo
Descrição
1
tem
N
Dependentes
Matrícula
Nome Dependente
Dt Nascimento

Sistema de Gerência de Banco de Dados

Conhecido, também, pela sigla SGBD, um sistema de gerência de bando de dados é um sistema o qual é responsável por armazenar e recuperar dados.

Uma tabela é um dos elementos de um SGBD responsável pelo armazenamento das informações.

São exemplos de SGBDs: Oracle, SQLServer, Mysql, Postgre, etc.

Modelo de banco de dados Relacional

Chave primária

A chave primária (ou simplesmente chave) é o identificador único de um registro em um arquivo. Pode ser constituída de um campo (chave simples) ou pela combinação de dois ou mais campos (chave composta), de tal maneira, que não existam dois registros no arquivo com o mesmo valor de chave primária.

Em regra, todo arquivo deve possuir uma chave primária, que permita a identificação inequívoca do registro, especialmente, para dar maior consistência aos processos de inclusão, alteração e exclusão de dados.

Para que não ocorram duplicatas nos valores da chave, os campos que a compõem são de preenchimento obrigatório (NOT NULL).

Na escolha da chave primária de um arquivo deve-se buscar campos que possuam estabilidade no valor armazenado. A escolha do *número do telefone* como chave de um cadastro de clientes, por exemplo, seria inadequada, por que esse valor pode mudar com freqüência. Sem considerar que o cliente pode ter mais de um telefone.

Deve-se também evitar a escolha de campos que possam causar ambiguidade em relação aos valores nele contidos. Nesse sentido, seria inadequada a escolha do campo nome para chave de um cadastro de clientes, haja vista, que um mesmo nome pode ser escrito de várias formas. Por exemplo: luís, luiz, louis, loys, luys.

Dicas para escolha da chave primária:

- Todo arquivo deve possuir uma chave primária.
- Valor único para cada registro.
- Simples ou composta.
- Campos de preenchimento obrigatório.
- Valor estável.

- Não ambíguo.
- Pequena extensão (menor possível).
- Preferência por campos numéricos

Chave secundária

A chave secundária pode ser formada por um campo ou pela combinação de campos (simples / composta). Ë utilizada como parâmetro (filtro) para seleção de registros no arquivo em consultas, emissão de relatórios ou processos de atualização simultânea de um grupo de registros.

Por exemplo, para aumentarmos o valor do salário dos analistas em 10%, poderíamos utilizar o campo do arquivo cadastro de funcionários como parâmetro (chave secundária) no processo de seleção dos registros a serem alterados.

Em síntese, a chave secundária é o campo ou combinação de campos que permite a recuperação de mais de um registro no arquivo. Portanto, não possui a característica de unicidade proposta para a chave primária.

Chave candidata

Pode ocorrer uma situação em que mais de um campo satisfaça a condição de chave primária, constituindo duas ou mais chaves candidatas. Neste caso, o analista deverá eleger somente uma delas como chave primária, as demais permanecerão na condição de candidatas, indicando que tratam-se de campos de preenchimento obrigatório e com valores únicos para cada registro, o que será garantido através de mecanismos de integridade de coluna, que veremos no capítulo relativo a banco de dados.

Tipos de dados

- Smallint - Armazena valores numéricos, em dois bytes binários, compreendidos entre o intervalo -32768 a +32767.
- Integer - Armazena valores numéricos, em quatro bytes binários, compreendidos entre o intervalo -2147483648 a +2147483647
- Decimal(n,m) - Armazena valores numéricos com no máximo 15 dígitos. Nesta opção deve ser definida a quantidade de dígitos inteiros (*n*) e casas decimais (*m*) existentes no campo.
- Varchar (n) - Definir um campo alfanumérico de até *n* caracteres, onde *n* deve ser menor ou igual a 254 caracteres.
- Char (n) - Definir um campo alfanumérico de n caracteres, onde *n* deve ser menor ou igual a 254 caracteres.
- Long Varchar - Definir um campo alfanumérico de comprimento maior que 254 caracteres.

- Date - Definir um campo para datas.
- Time - Definir um campo para horário.

Linguagens de banco de dados

As linguagens de banco de dados consistem na interface do usuário para interagir com o SGBD.

SQL(Structured Query Language)

A linguagem SQL (anteriormente escrita Sequel) foi criada junto com o Sistema R, primeiro protótipo de SGBD-R, desenvolvido de 1974 a 1979 no IBM San Jose Research Laboratory. A versão original do SQL foi baseada em uma linguagem anterior chamada SQUARE. As duas linguagens são essencialmente a mesma, mas a SQUARE usa uma sintaxe bem mais matemática, enquanto a SQL é mais parecida com o inglês.

A linguagem SQL é mais do que somente uma linguagem de consulta, sem que isto se oponha ao "query" no seu nome. Ela fornece funções de recuperação e atualização de dados, além de criação, manutenção da estrutura de dados e controle do ambiente do banco.

Suas principais características são:

- Padrão ANSI (American National Standard Institute). O ANSI estabeleceu-se como um "padrão de fato" de SQL para os fornecedores de produtos relacionais, que atualmente lideram o mercado. Este aspecto facilita a interoperabilidade entre bancos de diferentes fornecedores.
- Padrão de acesso. Todo o acesso ao banco é feito em SQL, mesmo que embutida em outra linguagem.
- Interpretada (não compilada), característica que provê maior grau de independência de dados aos bancos relacionais, ou seja, faz com que a aplicação reconheça alterações nas estruturas de dados, sem necessidade de ser recompilada.
- DDL, DML e DCL. O SQL possui esses três grupos de comandos, montados conforme a função do comando no banco de dados. Esta característica também relaciona-se à independência de dados, uma vez que pode-se descrever os dados (DCL) de forma independente das aplicações (DML).
- DDL (Data Definition Languge). Linguagem para definição de dados, que contempla um conjunto de comandos.

Comandos SQL

Create database

Comando utilizado para criar um banco de dados.

Sintaxe:

```
create database nome_banco;
```

Exemplo:

```
create database folhaescola;
```

O comando acima cria o banco de dados **folhaescola**.

Create table

Sintaxe:

```
create table nome_tabela
(campo tipo_de_dado not null,
campo tipo_de_dado,
campo tipo_de_dado,
primary key (nome_campo))
```

A opção *primary key* é utilizada para definir um campo como chave primária da tabela. Isso significa que essa chave não pode se repetir nunca na tabela, ou seja, que a chave de ser única. Assim, garantindo a integridade do banco de dados, não permitindo a duplicidade de informação.

Exemplo:

```
create table CadastroAluno(
        id_aluno integer not null,
        nome varchar(40),
        endereco varchar(40),
        numerointeger,
        CPF varchar(11),
        nome_mae varchar(40),
```

```
    nome_pai        varchar(40),
    primary key (id_aluno));
```

O exemplo acima cria um tabela com nome **CadastroAluno**, com os seguintes campos: id_aluno, nome, endereço, numero, CPF, nome_mae, nome_pai. A chave primária dessa tabela é o campo **id_aluno**. Os tipos de dados escolhidos foram: varchar para campos de texto e integer para campos numéricos. Observa-se que o campo **id_aluno** está setado para não aceitar valores nulos (*not null*), essa opção é importante nesse caso, pois esse campo é uma chave primária.

Update

Comando utilizado para atualizar um ou mais campos de um registro na tabela.

Sintaxe:

```
    Update          nome_tabela
set     nome_coluna=novo_valor
where           condição
```

Exemplo:

```
    update pagto
set dt_pagto=null
where id_pagto in (271,472,493,498,502,501,486,489,512,481,389,376,393);
```

O exemplo acima atualiza o campo da tabela **dt_pagto** para *null*, para os registros com valores definidos entre os parênteses na cláusula **where**.

Alter table

Altera ou atualiza uma informação na tabela. Quando é criado um campo novo, esse terá seu conteúdo com valor nulo (NULL)

Sintaxe:

```
    Alter   table   nome_tabela
drop    nome_coluna
add     nome_coluna   tipo
rename          nome_coluna   nome_nova_coluna
modify          nome_coluna   tipo
add     primary/foreign  key   coluna          references      tabela(campo)
```

Exemplo:

```
Alter   table   cadastroaluno
add     email   varchar(30);
```

O comando acima cria o campo email com o tipo varchar, na tabela cadastroaluno. Lembrando que o conteúdo desse campo será NULL para todos os registros da tabela.

Insert into

Insere valores em uma tabela.

Sintaxe:

```
insert  into    nome_tabela  (campos)    values  (valores)
```

Exemplo:

```
insert  into    CadastroAluno
values  (015,'Ricardo Teixeira','Av. Castro Alves',565,88746479312,'Márcia Alves
        Teixeira','Rodolfo Teixeira');
```

O exemplo mostra a inserção na tabela **Cadastroaluno** dos valores determinados na cláusula *values*. Como queremos inserir valores em todos os campos, não há necessidade de informar os nomes dos campos, mas isso implica em informar valores para todos os campos da tabela. Caso não ocorra, ao executar a query, irá informar erro na tentativa de inserir os dados.

Delete

Exclui um registro na tabela de acordo com a condição definida.

Sintaxe:

```
delete  from    nome_tabela  where   condição
```

Exemplo:

```
delete  from    CadastroAluno
where id_aluno=010;
```

O exemplo acima excluí o registro com o **id_aluno=010** da tabela **CadastroAluno**.

Drop table

Deleta uma tabela do banco de dados.

Sintaxe:

drop table nome_tabela

Exemplo:

drop table MatriculaAluno;

O comando acima deleta a tabela **MatriculaAluno**.

Select

Comando utilizado para recuperar registro do banco de dados.

Sintaxe:

select campos

from tabela

Exemplo:

select * from CadastraAluno;

Lista todos os registros da tabela **CadastroAluno**. O asterístico (*) é utilizado nesse caso para listar todos os campos.

O comando **select** apresenta um conjunto de cláusulas que podem ser utilizadas em sua sintaxe. Abaixo a lista de opções:

where – cláusula usada em conjunto com o select, para criar regras de condicionamento.

Selec nome

From cadastroaluno

Where id_aluno=003;

Na cláusula where podemos utilizar ainda os operadores AND, OR, BETWEEN, IN, LIKE, EXISTS.

AND: condição *E*

Selec nome

From cadastroaluno

Where id_aluno=003 and id_aluno=007;

OR: condição *OU*

```
Select  nome
        From    cadastroaluno
        Where   id_aluno=003    or      id_aluno=007;
```

BETWEEN: especifica uma faixa de valores numéricos

```
Select  nome
        From    cadastroaluno
        Where   id_aluno   between     002     and     005;
```

LIKE: permite utilizar uma cadeia de caracteres. Utilizar o caractere % como curinga (que especifica qualquer cadeia de caracteres)

```
        Select  nome
        From    cadastroaluno
        Where   nome    like    'M%';
```

IN: permite especificar uma lista de cadeias de caracteres. Permite o uso do operador NOT para negação.

```
Select  nome
        From    cadastroaluno
        Where   id_aluno  in      (004,006,008,010,011);
```

EXISTS: exibe o conteúdo existente de acordo com a condição. Também permite o uso do operador NOT para negação. Este operador é mais utilizado em subqueries, visto mais adiante.

```
        Select  nome
From    cadastroaluno
```

```
Where   exists
        (select * from pagto where id_aluno=a.id_aluno);
```

inner join – cláusula usada em conjunto com select para selecionar informações de tabelas diferentes

```
select cadastroaluno.nome, pagto.dt_venc, pagto.dt_pagto
from pagto
inner join cadastroaluno on pagto.id_aluno = cadastroaluno.id_aluno;
```

O exemplo acima seleciona o nome do aluno da tabela cadastroaluno e a data de vencimento e pagamento da tabela pagto. Juntamente com o inner join devemos utilizar o operador ON, para realizar a relação entre as tabelas, isto é, realizar a igualdade entre campos iguais pertencentes em ambas as tabelas. É obrigatória essa relação, para que não ocorra duplicidade dos registros retornados pela consulta. Posso utilizar a cláusula where depois do inner join.

Quando a query utilizar mais de uma tabela, é preciso referenciar o nome da tabela antes do campo, para o banco de dados saber em qual tabela deve buscar a informação daquele campo. Podemos utilizar nesses casos um alias para o nome da tabela, ou seja, um apelido, que deve ser inserido logo após o nome da tabela, como no exemplo abaixo:

```
select c.nome, p.dt_venc, p.dt_pagto
from pagto p
inner join cadastroaluno  c  on p.id_aluno = c.id_aluno;
```

Uma vez utilizado alias, é obrigatório o uso do alias como referencia, e não mais o nome todo da tabela. Utiliza-se alias para diminuir o tempo de criação de uma query.

outer join – Uma forma de relacionamento entre tabelas sem forçar um relacionamento de 1 para 1 é através do uso de “Outer Join”. Com este comando é possível apresentar registro de uma tabela A, e as relações existentes com a tabela B. Temos duas formas de utilizar o outer join:

left outer join – onde a tabela de comparação passa a estar “a esquerda” do comando.

```
select cadastroaluno.nome, pagto.dt_venc, pagto.dt_pagto
from pagto
left outer join cadastroaluno on pagto.id_aluno = cadastroaluno.id_aluno
where dt_pagto is null;
```

rigth outer join – onde a tabela de comparação passa a estar “a direita” do comando.

```
select cadastroaluno.nome, pagto.dt_venc, pagto.dt_pagto
from pagto
rigth outer join cadastroaluno on pagto.id_aluno = cadastroaluno.id_aluno;
```

distinct – cláusula usada em conjunto com select que elimina duplicações de registros numa consulta.

```
select distinct cadastro.nome
from cadastroaluno
inner join pagto on cadastroaluno.id_aluno=pagto.id_aluno;
```

O exemplo acima irá trazer apenas uma vez o nome de cada aluno, onde o campo id_aluno for igual nas tabelas de cadastroaluno e pagto.

order by campo – cláusula usada em conjunto com select que ordena por um ou mais atributos especificados em ordem crescente ou decrescente (desc). Por padrão, a ordenação ocorre em ordem crescente.

```
Select *
From cadastroaluno
Order by nome;
```

group by campo – cláusula usada em conjunto com select que agrupa linhas por um atributo definido.

```
select c.nome 'Nome do aluno'
from cadastroaluno c
where c.id_aluno not in (select m.id_aluno from matriculaaluno m where
                         m.semestre_mat=200601)
group by c.nome;
```

No exemplo acima, a cláusula group by irá agrupar todos os nomes de alunos iguais em uma única linha.

having condição – cláusula utilizada em conjunto com *group by* para limitar resultados

Funções de Agregação

As funções de agregação são utilizadas em conjunto com o comando select, na parte de definição dos campos.

Segue uma lista de funções:

COUNT - Retorna o número de linhas.

Exemplo:

```
Select count(*)
From cadatroaluno;
```

SUM - Retorna a soma de uma coluna específica.

Exemplo:

```
Select   sum(valor)
From     pagto;
```

AVG - Retorna o valor médio de uma coluna específica.

Exemplo:

```
Select   avg(valor)
From     pagto;
```

MAX - Retorna o valor máximo de uma coluna específica.

Exemplo:

```
Select   max(valor)
From     pagto;
```

MIN - Retorna o valor mínimo de uma coluna específica.

Exemplo:

```
Select   max(valor)
From     pagto;
```

Subquery

Subquery significa o uso de um comando SQL dentro de outro comando. Por exemplo, um comando select dentro de outro, como no exemplo abaixo:

```
Select    nome
From      cadastroaluno c
Where     exists
                (select * from pagto  p   where  c.id_aluno=p.id_aluno);
```

Utilizamos uma subquery quando precisamos de informações que através da estrutura básica do comando não nos permite o retorno esperado.

Create View

Para queries complexos ou que serão utilizados várias vezes durante um programa possui com como alternativa o uso de Views, pois além de padronizar a forma de realizar uma mesma consulta, visando a otimizando de índices.

Sintaxe:

```
Create   view      nome_view
AS
Comandos_SQL
```

Exemplo:

```
Create View vw_AlunosPagto
As
Select    nome
From      cadastroaluno c
Where     exists
                (select * from pagto  p  where   c.id_aluno=p.id_aluno);
```

Para consultar uma View, basta realizar um select no nome da View, conforme exemplo abaixo:

```
Select  *       from    vw_AlunosPagto;
```

A consulta acima retorna todos os nomes de alunos que já realizaram algum pagamento de mensalidade.

O uso de View permite a aplicação de agrupamento (Group By) ou ordenação (Order by) sem o uso de subqueries, separando a consulta principal. Em alguns sistemas gerenciadores de bancos de dados como Oracle e SQL Server é possível aplicar índices sobre campos de uma View.

Create Index

O uso de índices é muito importante para a performance de consultas em banco de dados relacionais, muitas vezes passam a fazer parte das validações e integridade dos dados dos sistemas. Ordenando fisicamente os registros no momento da sua inserção, normalmente aplicado sobre a chave primária da tabela, pois necessita que haja uma única ocorrência do índice.

Podemos classificar um índice como primário (ou índice de clustering) – quando a chave do índice é a chave primária da tabela, ou secundário – quando a chave do índice é uma chave candidata, ou seja, não é chave primária da tabela.

Sintaxe:

```
Create  index   nome_indice
ON      nome_tabela(nome_campo_que_sera_indice);
```

Exemplo:

```
create index aluno
ON cadastroaluno(id_aluno);
```

Exemplo de uso de índices em um banco de dados:

Um cadastro de produto onde a chave primária é o código interno do produto, porém o Código de Barra não pode se repetir no cadastro, porém este campo não faz parte da chave primária.

CodProduto	**Descric aoProduto**	**CodEAN**
123	Refrig Pepsi-Cola 2L	7891234
150	Refrig Cola-Cola 1,5L	7892542

167 Refrig Cola-Cola 1,5L 7892542 ← Este registro não será inserido na tabela, haverá erro de Chave candidata, evitando um segundo cadastro de um mesmo produto.