Introdução ao SQL

Professor Norton B. Glaser

1- Engenharia da Informação

1.1- Dados

O armazenamento de dados data do inicio dos tempos quando antigas civilizações criavam esculpiam na pedra símbolos que representavam futuramente uma historias, um calendários e etc. A necessidade de armazenar um conhecimento em forma de símbolos para ser utilizado por outros indivíduos compõem de certa forma a principal ferramenta do conhecimento humano.

Podemos definir dados como seqüência de símbolos quantificados ou quantificáveis
Originados de alguma entidade ou evento. Que serão utilizados futuramente na composição de uma informação.
Exemplos:

Uma palavra e composta por uma seqüência de símbolos(Letras) que por sua vez compõem um texto que fornece alguma informação.

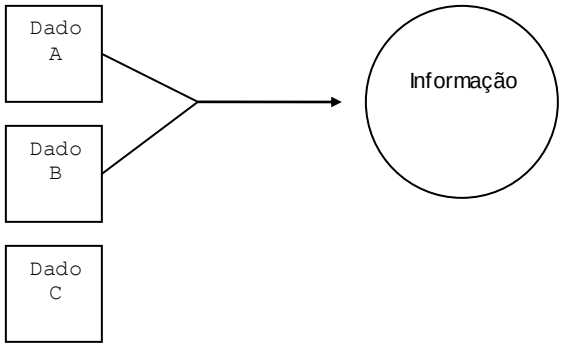
Por exemplo, um documento de identificação pode conter vários dados de uma pessoa como nome, sexo, data de nascimento

Outros exemplos de dados são a temperatura de uma cidade, ou a área de um território. Ainda que estes pareçam, por vezes, isolados, podem sempre englobar-se em conjuntos (as temperaturas das cidades de uma província ou país, ou as áreas de um conjunto de territórios) ou séries (as temperaturas de uma cidade ao longo do tempo).

1.2- Informação

A informação por sua vez e uma abstração dos dados e suas co-relação com o objetivo de resolver uma determinada questão.
Para cada questão uma abstração diferente deve ser formulada com base nos dados resultado em informações diferentes.
Exemplo:

Em um texto uma palavra isolada traz somente a informação do significado da palavra. Dentro de um contexto relacionando-se com as outras palavras a informação resultante e totalmente diferente.



1.3- Armazenamento de Dados

O termo banco de dados foi criado inicialmente pela comunidade de tecnologia, para indicar coleções organizadas de dados armazenados em Sistemas computacionais, porém hoje e utilizado para indicar bases de dados que utilizam um software de SGBD (Sistema de gerenciamento de banco de dados).

O termo **base de dados** e utilizado quando se mencionam outros tipos de bancos de dados (planilhas, arquivos textos).

Aceitando uma abordagem mais técnica, um **banco de dados** é uma coleção de registros salvos em um computador em um modo sistemático, de forma que um programa de computador possa consultá-lo para gerar informações.

Normalmente um registro está associado a um conceito completo e é dividido em campos, ou atributos, que dão valores a propriedades desses conceitos.

Possivelmente alguns registros podem apontar diretamente ou referenciar indiretamente outros registros, o que faz parte da caracterização do modelo adotado pelo banco de dados.

A descrição de quais são os tipos de registros existentes em um banco de dados e ainda quais são os campos de cada registro é conhecida como esquema do banco de dados.

Estritamente falado, o termo banco de dados deve ser aplicado apenas aos dados, enquanto o termo sistema gerenciador de bancos de dados deve ser aplicado ao software com a capacidade de manipular bancos de dados de forma geral. Porém, é comum misturar os dois conceitos.

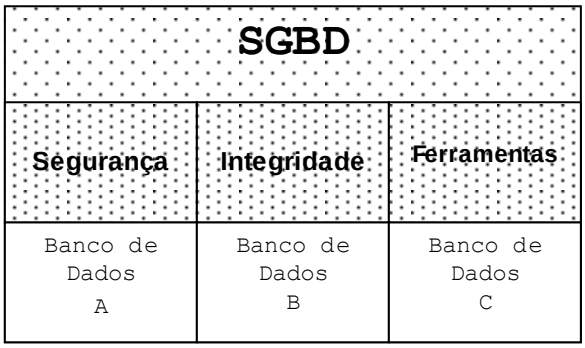
1.4- SGBD

Um Sistema Gerenciador de Banco de Dados ou Sistema Gestor de Base de Dados (SGBD) é o conjunto de programas de computador (softwares) responsáveis pelo gerenciamento de uma base de dados. O principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, manipulação e organização dos dados. O SGBD disponibiliza uma interface para que os seus clientes possam incluir, alterar ou consultar dados. Em bancos de dados relacionais a interface é constituída pelas APIs ou drivers do SGBD, que executam comandos na linguagem SQL.

Um sistema gerenciador de banco de dados é um sistema extremamente complexo, responsável pela persistência, organização e recuperação dos dados.

Exemplos de SGBD:

- Microsoft SQL Server
- Oracle
- MySql
- Ibm DB2



1.5- Banco de dados Relacional

Um Banco de Dados Relacional é um banco de dados que segue o Modelo Relacional.

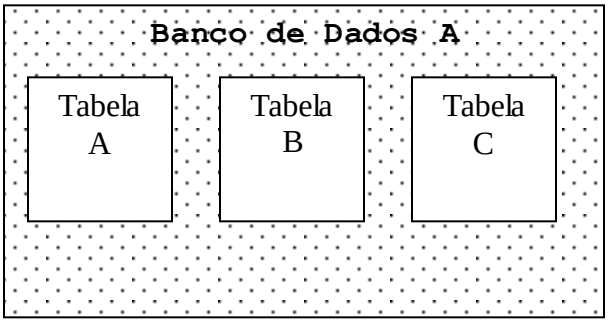
De forma mais detalhada, um Banco de Dados Relacional é um conceito abstrato que define maneiras de armazenar, manipular e recuperar dados estruturados unicamente na forma de tabelas, construindo um banco de dados.

Os Bancos de Dados Relacionais foram desenvolvidos para prover acesso facilitado aos dados, possibilitando que os usuários utilizassem uma grande variedade de abordagens no tratamento das informações. Pois, enquanto em um banco de dados hierárquico os usuários precisam definir as questões de negócios de maneira específica, iniciando pela raiz do mesmo, nos Bancos de Dados Relacionais os usuários podem fazer perguntas relacionadas aos negócios através de vários pontos.

A linguagem padrão dos Bancos de Dados Relacionais é a **Structured Query Language**, ou simplesmente SQL, como é mais conhecida.

Um Banco de Dados Relacional segue o Modelo Relacional.

A arquitetura de um banco de dados relacional pode ser descrita de maneira informal ou formal. Na descrição informal estamos preocupados com aspectos práticos da utilização e usamos os termos tabela, linha e coluna. Na descrição formal estamos preocupados com a semântica formal do modelo e usamos termos como relação(tabela), tupla(linha) e atributo(coluna).



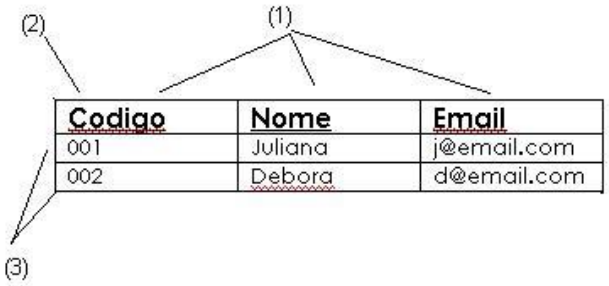
1.6- Tabelas

Todos os dados de um banco de dados relacional (BDR) são armazenados em tabelas. Uma tabela é uma simples estrutura de linhas e colunas. Em uma tabela, cada linha contém um mesmo conjunto de colunas, e estas linhas devem seguir a ordem que foi especificada pelo projetista do BDR. Em um banco de dados podem existir uma ou centenas de tabelas, sendo que o limite pode ser imposto tanto pela ferramenta de software utilizada, quanto pelos recursos de hardware disponíveis no equipamento.

As tabelas associam-se entre si através de regras de relacionamentos, estas regras consistem em associar um atributo de uma tabela com um conjunto de registros de outra tabela.

Exemplo:

A tabela funcionário relaciona-se com a tabela cargo. Através deste relacionamento esta última tabela fornece a lista de cargos para a tabela funcionário.



(1) Campos
(2) Chave primaria
(3) Registros

1.7- Registros (ou tuplas)

Cada linha formada por uma lista ordenada de colunas representa um registro, ou tupla. Os registros não precisam conter informações em todas as colunas, podendo assumir valores nulos quando assim se fizer necessário.
Resumidamente, um registro é uma instância de uma tabela, ou entidade.

Exemplo:
O empregado Pedro é uma instância (registro) da tabela funcionário, e a função Analista Comercial é a instância (registro) da tabela cargo. Uma associação entre estas duas tabelas criaria a seguinte instância de relacionamento: Pedro é Analista Comercial, onde o verbo é representa uma ligação entre os registros distintos.

1.8 Colunas (ou atributos)

As colunas de uma tabela são também chamadas de Atributos. Ao conjunto de valores que um atributo pode assumir chamamos de domínio, por exemplo: em um campo do tipo numérico, serão somente armazenados números.
O conceito mais similar a domínio é o de Tipo Abstrato de Dados em linguagens de programação.

1.9 Chave

As tabelas relacionam-se umas as outras através de chaves. Uma chave é um conjunto de um ou mais atributos que determinam a unicidade de cada registro.

Por exemplo, se um banco de dados tem como chaves Código do Produto e ID Sistema, sempre que acontecer uma inserção de dados o sistema de gerenciamento de banco de dados irá fazer uma consulta para identificar se o registro já não se encontraria gravado na tabela. Neste caso, um novo registro não será criado, resultando esta operação apenas da alteração do registro existente.

A unicidade dos registros, determinada por sua chave, também é fundamental para a criação dos índices.

Temos dois tipos de chaves:

Chave primária: (PK - Primary Key) é a chave que identifica cada registro dando-lhe unicidade. A chave primária nunca se repetirá.

Chave Estrangeira: (FK - Foreign Key) é a chave formada através de um relacionamento com a chave primária de outra tabela. Define um relacionamento entre as tabelas e pode ocorrer repetidas vezes. Caso a chave primária seja composta na origem, a chave estrangeira também o será.

Produto(PK)	Nome	Valor
B01	Lápis	R$ 10,20
C43	Caneta	R$ 35,20

Tabela de Produto

Pedido(PK)	Produto(fk)	Qtd
1	B01	3
2	C45	2

Tabela de Pedidos

1.10 SQL

Structured Query Language, ou Linguagem de Consulta Estruturada ou SQL, é uma linguagem de pesquisa declarativa para banco de dados relacional (bases de dados relacionais). Muitas das características originais do SQL foram inspiradas na álgebra relacional.

Embora o SQL tenha sido originalmente criado pela IBM, ele foi padronizando pela American National Standards Institute (ANSI) e ISO em 1992.

2- Criando Tabelas

O primeiro comando a ser executado e o de criação de tabelas, neste comando vamos definir toda a sua estrutura.
Sintaxe:

```
CREATE TABLE <NOME DA TABELA> (
       <CAMPO><TIPO>[DEFAULT()][NULL],
       .
       .
       .
)
```

Exemplo:

```
CREATE TABLE tb_usuario (
       codigo  int,
       nome varchar(100),
       email varchar(50),
       telefone varchar(35))
```

A criação dos nomes de tabela e dos campos e livre, mas as regras abaixo são boas praticas que devem ser seguidas:

- Nome em minúsculo
- As tabelas em singular
- Não utilizar nome compostos separados por espaço

2.1 – Tipos de campos

As colunas de uma tabela precisam ter um tipo vinculado, um tipo e a definição de um formato de armazenagem e de exibição.
Entre os principais tipos nos temos:

Nome	Tipo
Char(9)	Texto com caracteres fixos **Exemplo:** Char(9) = "Ola" Vai armazenar 9 posições
Varchar(99)	Texto com caracteres dinamicos **Exemplo:** varchar(10) = "Ola" Vai armazenar 3 posições
Int	Armazena números inteiros **Exemplo:** int = 587
Decimal(10,2)	Armazena números com casas decimais **Exemplo:** Decimal(10,2) Aonde 10 e o numero de dígitos e 2 e o numero de casas decimais.
Currency	Armazena valores em moeda **Exemplo:** currency = 10.58
Date	Armazena data e hora **Exemplo:** Date = "2006-09-31 15:00"
Text	Textos longos sem tamanho definido.

2.2 – Auto incremento

Um campo de auto incremento e um campo numérico inteiro que e acrescido automaticamente. Este campo normalmente e utilizado como chave primaria em tabelas por não permitir duplicação de valores
Sintaxe:

```
<CAMPO><INT>[IDENTITY(<INICIO>,
                      <INCREMENTO>)]
```

Aonde:
<INICIO> - Número que inicia a contagem.

<INCREMENTO> - De quantos em quantos números vai ser feita a soma.

Exemplo:

```
CREATE TABLE tb_usuario (
       codigo  int identity(50,1),
       nome varchar(100),
       email varchar(50),
       telefone varchar(35))
```

2.3 – Valores Default

Podemos definir valores padrão para as colunas, caso nenhum valor seja associado a coluna será preenchida com o valor padrão.

Sintaxe:

```
<CAMPO><TIPO>[DEFAULT(<VALOR>)]
```

Aonde:
<VALOR> - E o valor padrão a ser associado ao campo.

Exemplo:

```
CREATE TABLE tb_cotacao (
      codigo  int identity(50,1),
      nome varchar(50),
      valor money default(1))
```

2.4 – Obrigatoriedade

Podemos definir a obrigatoriedade de preenchimento de um campo ou não, utilizando os paramentos NULL(Permite Nulo) e NOT NULL (Não permite nulo).

Sintaxe:

```
<CAMPO><TIPO>[NULL | NOT NULL]
```

Aonde:
<NULL> - Campo não obrigatório
<NOT NULL> - Campo obrigatório.

Exemplo:

```
CREATE TABLE tb_cotacao (
      codigo  int identity(50,1),
      nome varchar(50) NOT NULL,
      valor money default(1) NULL)
```

2.5 – Removendo Tabelas

Para removermos tabelas do nosso banco de dados utilizamos o comando abaixo:

Sintaxe:

```
<DROP TABLE><TABELA>
```

Exemplo:

```
DROP TABLE tb_cotacao
```

2.6 – Alterando Tabelas

Muitas vezes esquecemos de alguma coluna, ou algum parâmetro esta errado, para isto podemos modificar uma tabela através do campo ALTER TABLE:

Sintaxe:

Para incluir uma nova coluna:

```
<ALTER TABLE><TABELA><ADD><COLUNA> <TIPO>
[DEFAULT(<PADRAO>)]
[NOT NULL | NULL]
```

Para remover uma nova coluna:

```
<ALTER TABLE><TABELA><DROP COLUMN> <COLUNA>
```

Exemplo:

Para incluir uma nova coluna:

```
Alter table tb_cotacao add status int
```

Para remover uma nova coluna:

```
Alter table tb_cotacao drop column status
```

2.7 – Limpando Tabelas

Para limparmos o conteúdo de uma tabela utilizamos o comando TRUNCATE TABLE, este comando também reinicia os campos auto-incremento.

Sintaxe:

```
<TRUNCATE TABLE><TABELA>
```

Exemplo:

```
truncate table tb_cotacao
```

Exercícios

1-Crie a tabela(vendedor) no banco de Dados com os seguintes campos:
código (auto incremento)
vendedor
telefone
comissão

2-Adicione uma nova coluna
3-Remova esta coluna
4-Remova a tabela

(Obs: Para ver a estrutura da tabela use o comando <sp_help><tabela>)

3 – Comandos de consulta

O comando de consulta tem como objetivo minerar os dados com parâmetros objetivando a informação, podemos fazer consultar relacionando tabelas (Intersecção e união)

3.1 – Comando Select

O comando de consulta e composto da seguinte forma:

Sintaxe:

```
<SELECT CAMPO1,CAMPO2,...>
<FROM TABELA1,TABELA2,...>
[WHERE <CONDIÇÕES>]

Aonde:
<SELECT> Lista dos campos a serem listados.
<FROM> Lista de tabelas consultadas.
[WHERE] Lista de condições da consulta.
```

Exemplo:

```
Select nome,email
from cliente
where codigo=12
```

Obs: Seleciona os campos(nome e email), quando o codigo=12

ou

```
Select *
from cliente
where codigo=12
```

Obs: (*) Seleciona todos os campos, quando o codigo=12

ou

```
Select produto.codigo,fornecedor.nome
from produto, fornecedor
where produto.fornecedor = fornecedor.fornecedor
```

Obs: Neste exemplo estamos usando duas tabelas, precisamos identificar a tabela a qual campo pertence (produto.codigo).

3.2 – Alias

Podemos dar apelidos as tabelas e as colunas para deixar o comando mais simples.

Sintaxe:

```
Alias para campos
<Campo>[as <Alias>]

Aonde:
[as <Alias>] nome do apelido.

e/ou

Alias para tabelas
<tabela><Alias>

Aonde:
<Alias> nome do apelido da tabela.
```

Exemplo:

```
Select t1.codigo as c1,t2.nome as c2
from produto t1, fornecedor t2
where t1.fornecedor = t2.fornecedor
```

3.3 – Condição

O bloco de condição permite utilizar condições lógicas para fazer o relacionamento entre campos e valores.

Para realizar estas operações nos utilizamos os operadores lógicos e relacionais

Operadores Lógicos

Operador	Descrição
=	Igualdade
<>	Diferente
>	Maior
<	Menor
>=	Maior igual
<=	Menor igual

Exemplo:

```
Select * from tb_cliente where idade >= 18

Ou

Select * from tb_cliente where codigo=123
```

Operadores Relacionais
Quando tivermos mais de uma condição lógica em um comando SQL, precisaremos utilizar os operadores relacionais para compor a expressão:

Operador	Descrição
AND	E – Quando as duas condições forem verdadeiras
OR	OU – Quando uma das condições for verdadeira
NOT	Não – Inverte uma condição

Exemplo:

```
Select * from tb_cliente where idade >= 18 and
cidade='cotia'

Ou

Select * from tb_cliente where codigo=123 or
Codigo=567

Ou

Select * from tb_cliente where not codigo=123
```

3.4 – Ordenação

Para ordenar o resultado de uma consulta utilizamos a clausula order by:
Sintaxe:

```
<SELECT CAMPO1,CAMPO2,...>
<FROM TABELA1,TABELA2,...>
[ORDER BY <CAMPO,...>[DESC]]

Aonde:
<CAMPO,...> Lista dos campos que ordenaram a
consulta.
[DESC] Por padrão a ordenação e crescente, mas
com o parâmetro DESC a ordenação ficara
decrescente.
```

Exemplo:

```
Select nome,email
from cliente
where idade>=18 order by name

ou

Select nome,email
from cliente
where idade>=18 order by idade desc
```

Exercícios

1-Faça uma consulta de todos os campos na tabela cliente

2-Consulte os campos nome e idade da tabela cliente onde o sexo='F'

3-Consulte os campos da tabela de cliente quando o sexo='F' e a idade for >=18, ordenando pela idade e nome

4-consulte os campos produto,valor e quantidade da tabela produto e estoque ordenando pela quantidade em estoque decrescente

4 – Comandos de Execução

Complementado os comandos de consulta os comando de execução tem como objetivo administrar a base de dados.
Através da inclusão, alteração e exclusão de registros.

4.1 – Comando e Inclusão

A inclusão de registro e feita através do comando INSERT INTO

Sintaxe:

```
<INSERT INTO><TABELA><(campos,...)>
<VALUES(valores,...)>

Aonde:
<(CAMPOS,...)> Lista dos nomes dos campos serão incluídos, todos os campos obrigatórios deveram estar inclusos nesta lista. Caso algum campo seja auto-incremento não e necessário que este esteja listado.

<VALUES(CAMPOS,...)> Lista dos valores relacionados ao campos, quando o valores forem alfanumérico deverão estar entre aspas.
```

Exemplo:

```
Insert into tb_cliente(nome, email, cidade, idade) values('maria', 'maria@uol.com', 'cotia', 12)
```

4.2 – Comando de exclusão

A exclusão pode ser feita de um registro ou de um conjunto de registros, o comando de exclusão possui características de comandos de consulta.

Sintaxe:

```
<DELETE><TABELA>[WHERE <CONDICAO>]
```

Exemplo:

```
Delete tb_cliente where código=123
Excluindo um registro especifico

Ou

Delete tb_cliente where cidade="taboao"
Excluindo todos os registros da cidade do taboao

Ou

Delete tb_cliente
Apaga todos os registros da tabela
```

4.3 – Comando de alteração

O comando de alteração visa modificar o conteúdo de um ou mais campos, de um registros ou mais.

Sintaxe:

```
<update><TABELA><set><campos=valor,...>[WHERE <CONDICAO>]

Aonde:
<campos=valor,...> lista de campos a serem alterados com o valor a ser associado.
```

Exemplo:

```
Update tb_cliente Set idade=20, cidade='jundiai'
Altera os dois campos acima de todos os registros

Ou

Update tb_cliente Set idade=20, cidade='jundiai'
where código=1123
Altera os dois campos de um registro especifico
```

Exercícios

1- Inclua um registro na tabela cliente

2- Altere o registro incluído, o campo cidade para "são Paulo"

3- Exclua os registros da tabela cliente os quais a idade menor que 18.

5 – Agrupamento e sumarizações

Por muitas vezes em uma consulta a informação requisitada e a composição e a sumarização de registros. Através do agrupamento podemos obter totais, medias, menores valores e maiores valores de um conjunto de dados.

5.1 – Agrupando dados

Para agrupar dados utilizamos o comando **GROUP BY** este comando agrupo um conjunto de dados com uma determinada característica.

Sintaxe:

```
<GROUP BY><TABELA>[WHERE <CONDICAO>]
```

Exemplo:

Select filial,count(codigo) from pedido

```
Group by filial
```

5.2 – Comandos de sumarização

Para complementar a utilização do comando de agrupamento temos disponível os comandos de sumarização que são utilizados para a composição de relatórios e consultas

5.2.1-Count

O comando count conta a ocorrência de um campos dentro de um agrupamento.

Sintaxe:

```
COUNT(<CAMPO>)
```

Exemplo:

Select filial,count(codigo) from pedido

```
Group by vendedor
```

5.2.2-Sum

O comando SUM soma todas as ocorrências de um campos dentro de um agrupamento.

Sintaxe:

```
SUM(<CAMPO>)
```

Exemplo:

Select filial,SUM(VALOR) from pedido

```
Group by filial
```

5.2.3-Max

O comando MAX traz a maior ocorrência de um valor de campo dentro de um agrupamento.

Sintaxe:

```
MAX(<CAMPO>)
```

Exemplo:

Select filial,MAX(VALOR) from pedido

```
Group by filial
```

5.2.4-Min

O comando MIN traz a menor ocorrência de um valor de campo dentro de um agrupamento.

Sintaxe:

```
MIN(<CAMPO>)
```

Exemplo:

Select filial,MIN(VALOR) from pedido

```
Group by filial
```

5.2.5-Average

O comando AVERAGE realiza a media de um valor de campo dentro de um agrupamento.

Sintaxe:

```
MAX(<CAMPO>)
```

Exemplo:

Select filial, AVERAGEMAXSUM(VALOR) from pedido

```
Group by filial
```

Exercícios

1- Agrupe os pedidos por vendendor , mostrando o numero de pedidos , total dos pedidos e media dos pedidos.

2- Verifique no banco de dados o menor e o maior valor de pedido

6 – Funções Complementares

O Sql possui funçoes complementares que auxiliam na composição de consultas e formatação de dados.

6.1 – Funções de manipulação de texto

Os campos do tipo texto por muitas vezes precisam de uma pré-formatação para exibição correta em relatórios. Para isto utilizamos funções abaixo:

6.1.1 – Len *(Largura)*
Função que retorna o numero de caracteres de um valor de um campo.
Sintaxe:
```
Len(<campo>)
```

Exemplo:
```
Select nome, len(nome) from cliente
```

6.1.2 – substring *(Subcaracter)*
Função que retorna um pedaço do caracter de uma posição ate outra.
Sintaxe:
```
substring(<campo>,<inicio>,<fim>)
```

Exemplo:
```
Select substring("ola turma",5,10)
```

vai retornar o string ***"turma"***

6.1.3 – replace *(substituir)*
Função para substituir uma seqüência de caracteres por outra dentro de um valo de campo. E utilizado para correção de dados armazenados.
Sintaxe:
```
replace(
<campo>,
<sequencia>,
<alteração>)
```

Exemplo:
```
Select replace("Jose de silva","de","da")
```

vai retornar: ***"Jose da silva"***

6.1.4 – lower *(minuscul)*
Transforma as letras do campo para minúsculo. E utilizado para correção de dados armazenados.
Sintaxe:
```
lower(<campo>)
```

Exemplo:
```
Select lower("MARIANA Souza")
from cliente
```

vai retornar: ***"mariana Souza"***

6.1.5 – Upper *(maiúsculo)*
Transforma as letras do campo para maiúsculo. E utilizado para correção de dados armazenados.
Sintaxe:
```
upper(<campo>)
```

Exemplo:
```
Select upper("MARIANA Souza")
from cliente
```

vai retornar: ***"MARIANA SOUZA"***

6.2 – Funções de manipulação de números

Para os números o SQL dispõem de funções matemáticas que facilitam o calculo de formulas dentro da consulta.

Entre as operações básicas nos temos:
Adição (+)
```
Select valor+frete from pedido
```

Subtração (-)
```
Select (valor+frete) - desconto from pedido
```

multiplicação(*)
```
Select (valor*qtd) from item
```

divisão(/)
```
Select (valor/0.10) from item
```

6.2.2 – Power *(potencia)*

Eleva a potencia de um numero por outro.
Sintaxe:
`power(<base>,<potencia>)`

Exemplo:
```
Select power(2,3)
```

Vai retornar: ***8***

6.2.3 – SQRT *(Raiz Quadrada)*
Retorna a raiz quadrada de um numero.
Sintaxe:
`sqrt(<numero>)`

Exemplo:
```
Select sqrt(25)
```

Vai retornar: 5

6.2.4 – Round *(Arrendondar)*
Arredonda um numero decimal.
Sintaxe:
`round(<numero>,<casas>)`

Exemplo:
```
Select round(25.34534,2)
```

Vai retornar: 25,35

6.3 – Funções de manipulação de datas Exercícios

As datas são campos importantes na coleta de dados pois possibilita a geração de informações baseadas no tempo.

6.3.1 – getdate() *(Pegue a Data)*
Traz a data atual do sistema.
Sintaxe:
`getdate()`

Exemplo:
```
Select getdate()
```

Vai retornar: 2006-10-03 09:30:00.000

6.3.2 – month *(mês)*
Traz o mês de uma determinada data.
Sintaxe:
`Month(<data>)`

Exemplo:
```
Select month(getdate())
```

Vai retornar: 10

6.3.3 – Year *(ano)*
Traz o ano de uma determinada data.
Sintaxe:
`year(<data>)`

Exemplo:
```
Select year(getdate())
```

Vai retornar: 2006

6.3.4 – day *(dia)*
Traz o dia de uma determinada data.
Sintaxe:
`day(<data>)`

Exemplo:
```
Select day(getdate())
```

Vai retornar: 10

6.3.5 – dateadd *(adiciona data)*
Adiciona um intervalo a data passada como parâmetro. Recebe três argumentos, A parte da data a ser incrementada (DatePart), o incremento e a data alvo.

Sintaxe:
`dateadd(<parte>,valor,<data>)`

Aonde a <parte> pode ser:
year (Ano)
quarter (trimestre)
month (mes)
dayofyear (dia do ano)
day (dia)
week (Semana)
hour (hora)
minute (minuto)
second (segundo)
millisecond (milesegundos)

Exemplo:
```
Select dateadd(month,3,getdate())
```

Vai retornar: 2007-01-03 09:30:00.000

6.3.6 – datediff *(diferença de datas)*
Calcula a diferença entre duas datas e possível especificar qual parte da data será utilizada no calculo.

Sintaxe:
`datediff(<parte>,<data 1>,<data 2>)`

Aonde a <parte> pode ser:
year (Ano)
quarter (trimestre)
month (mes)
dayofyear (dia do ano)
day (dia)
week (Semana)
hour (hora)
minute (minuto)
second (segundo)
millisecond (milesegundos)

Exemplo:
```
Select  datediff(day,'2006-10-01','2006-10-10')
```

Vai retornar: 9 (dias)

Exercícios

1- Faça uma consulta que traga todos os pedidos dos ultimo 3 meses.

2- faça uma consulta que traga o nome dos clientes em maiúsculo e troque os espaços em branco por pontos.

7- Visualizações

Quando efetuamos uma consulta em nosso banco de dados que é utilizada repetidas vezes nos podemos criar uma **VIEW**(Visualização), que possui o objetivo de armazenar consultas complexas para serem utilizadas de forma direta.

A Visualização e atualizada automaticamente de acordo com a alteração e manipulação de registros.

Sintaxe:

```
CREATE VIEW <NOME DA VIEW> AS
<CONSULTA SQL>
```

Exemplo:

```
Create view AtendimentoVendedor AS
Select      t1.codigo,      t1.vendedor,
t2.codigo,  t2.nome,  t2.telefone  from
vendedor t1 , cliente t2
Where t2.vendedor=t1.codigo
Group by t1.codigo,t1.vendedor
```

Desta forma podemos utilizar a view acima como uma tabela da seguinte forma:

Select * from `AtendimentoVendedor`

7.1 Consultando uma view já existente

Para consultar o código pertencente ao uma view já existe utilizamos a função sp_helptext:

Sintaxe:

```
sp_helptext <nome da view>
```

Exemplo:

```
sp_helptext AtendimentoVendedor
```

7.2 Apagando uma view já existente

Para excluir uma view que não esta sendo utilizada usamos o comando:

Sintaxe:

```
Drop view <nome da view>
```

Exemplo:

```
Drop view AtendimentoVendedor
```

7.3 alterando uma view já existente

Para alterarmos uma view utilizamos o comando alter view

Sintaxe:

```
ALTER VIEW <NOME DA VIEW> AS
<CONSULTA SQL>
```

Exemplo:

```
alter view AtendimentoVendedor AS
Select      t1.codigo,      t1.vendedor,
t2.codigo,  t2.nome,  t2.telefone  from
vendedor t1 , cliente t2
Where t2.vendedor=t1.codigo
Group by t1.codigo,t1.vendedor
Order by t1.vendedor
```

Exercícios

1- Crie uma view de produtos com a sua quantidade vendida.
2- Altere a view para ordenar de forma decrescente por quantidade vendida.