

Banco de Dados II

Cristiane Tuji da Silva

Objetivos

- ⇒ Introdução
 - ⇒ MS-SQL Server - Overview
 - ⇒ Modelos de Implementação de Banco de Dados
 - ⇒ Restrição de Integridade
 - ⇒ Segurança de Acesso a Banco de Dados
 - ⇒ Arquitetura Cliente Servidor
 - ⇒ Arquitetura do MS-SQL Server e BD Físicos
 - ⇒ Tipo de Consultas ao Banco de Dados
 - ⇒ Tipo de Dados
 - ⇒ Comandos SQL:
 - ⇒ DDL - Data Definition Language
 - ⇒ DCL - Data Control Language
 - ⇒ DML - Data Manipulation Language
-

Introdução: Arquivos x Banco de Dados

⇒ Dados armazenados em arquivos:

- ⇒ dados não centralizados
 - ⇒ mudança de esquema, ou formato dos dados sem independência física aos programas
 - ⇒ acesso concorrente prejudicada
 - ⇒ compartilhamento de dados ineficiente
 - ⇒ recuperação automatizada não existe
 - ⇒ segurança de acesso - Não há plano de lock
 - ⇒ tratamento de falhas - ineficientes
 - ⇒ em resumo: sua manutenção exige muito esforço
-

Introdução: Arquivos x Banco de Dados

- ⇒ Dados armazenados em Banco de Dados
 - ⇒ Gerenciador de Banco de Dados (SGBDR)
 - ⇒ gravação e recuperação seguras e eficientes
 - ⇒ mudanças de esquema - Dicionário de Dados
 - ⇒ segurança de acesso
 - ⇒ recuperação automática
 - ⇒ concorrência
 - ⇒ acesso através de interface padrão (linguagem de consulta)
 - ⇒ independência física
-

Projeto de um Banco de Dados

- ⇒ identificar e organizar os dados relevantes a serem armazenados
 - ⇒ identificar o atributo determinante
 - ⇒ identificar os relacionamentos entre esses dados
 - ⇒ documentar os dados de forma inteligível e cristalina
 - ⇒ produzir um Modelo de Implementação
 - ⇒ buscar flexibilidade, escalabilidade, estabilidade, integridade e desempenho
 - ⇒ implementar o Banco de Dados
 - ⇒ apoiar aos usuários “How to use”
 - ⇒ manter a disponibilidade do Banco de Dados
 - ⇒ contingência do Banco de Dados
-

[illegible]

Restrição de Integridade Referencial

⇒ Impõe que linhas ligadas sempre estejam associadas a uma linha válida

⇒ **Exclusão**

⇒ **Cascata**

⇒ **Anular**

⇒ **Impedir**

⇒ **Alteração**

⇒ **Cascata**

⇒ **Impedir**

Restrição de Integridade - domínio

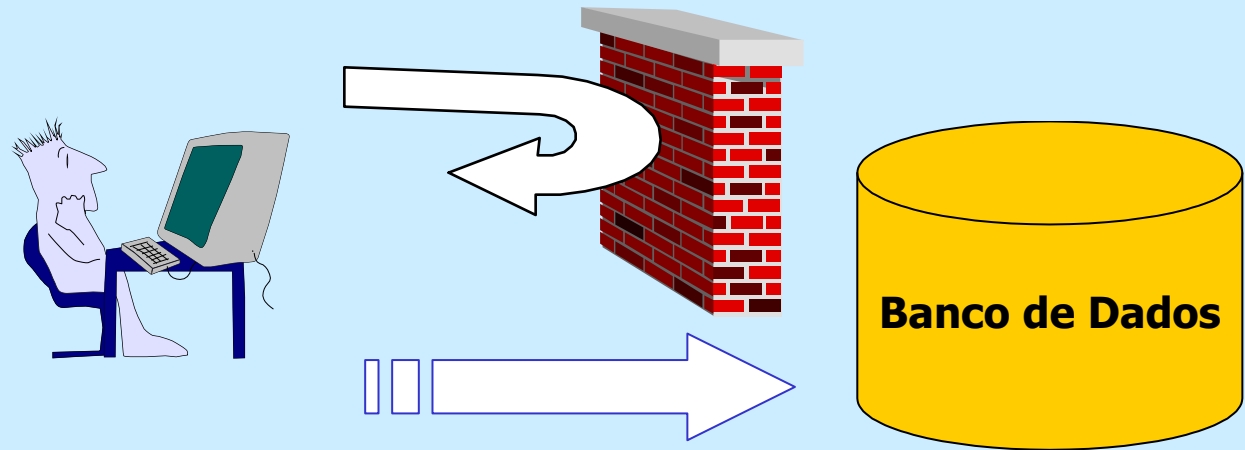
- ⇒ Mantém a validade das informações dos campos com valores sempre pertencentes ao domínio (tipo de dados)
 - ⇒ São implementadas pelo SGBD
 - ⇒ Pode ser necessário implementar algumas validações por programação através de Stored Procedure ou Trigger.
-

Segurança de Acesso ao BD

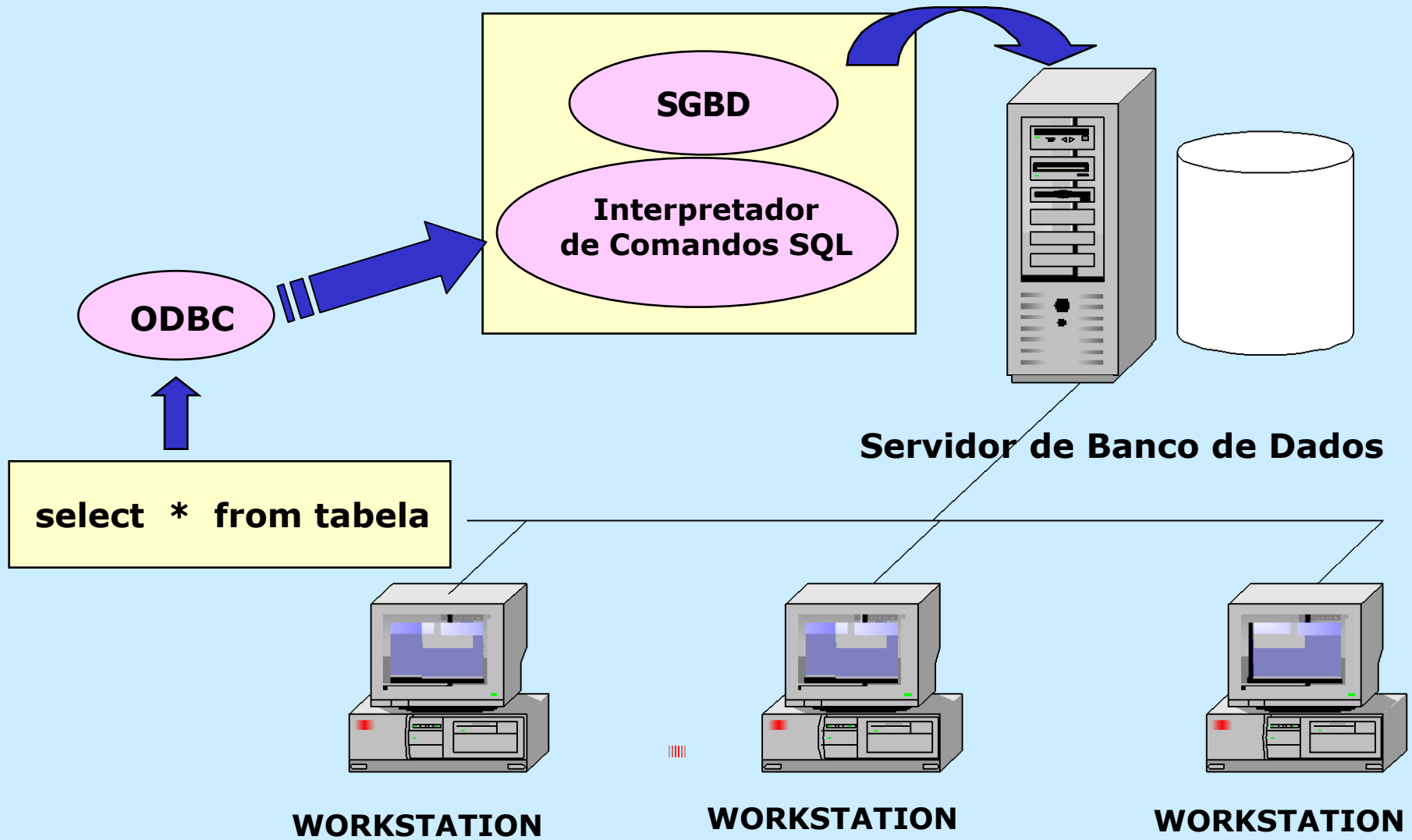
- ⇒ É controlada pelo SGBD ou pelo Sistema Operacional
- ⇒ O acesso ao Banco de Dados, pode ser restrito através de permissões e senhas
- ⇒ Há comandos SQL para controlar a segurança.

Revoke

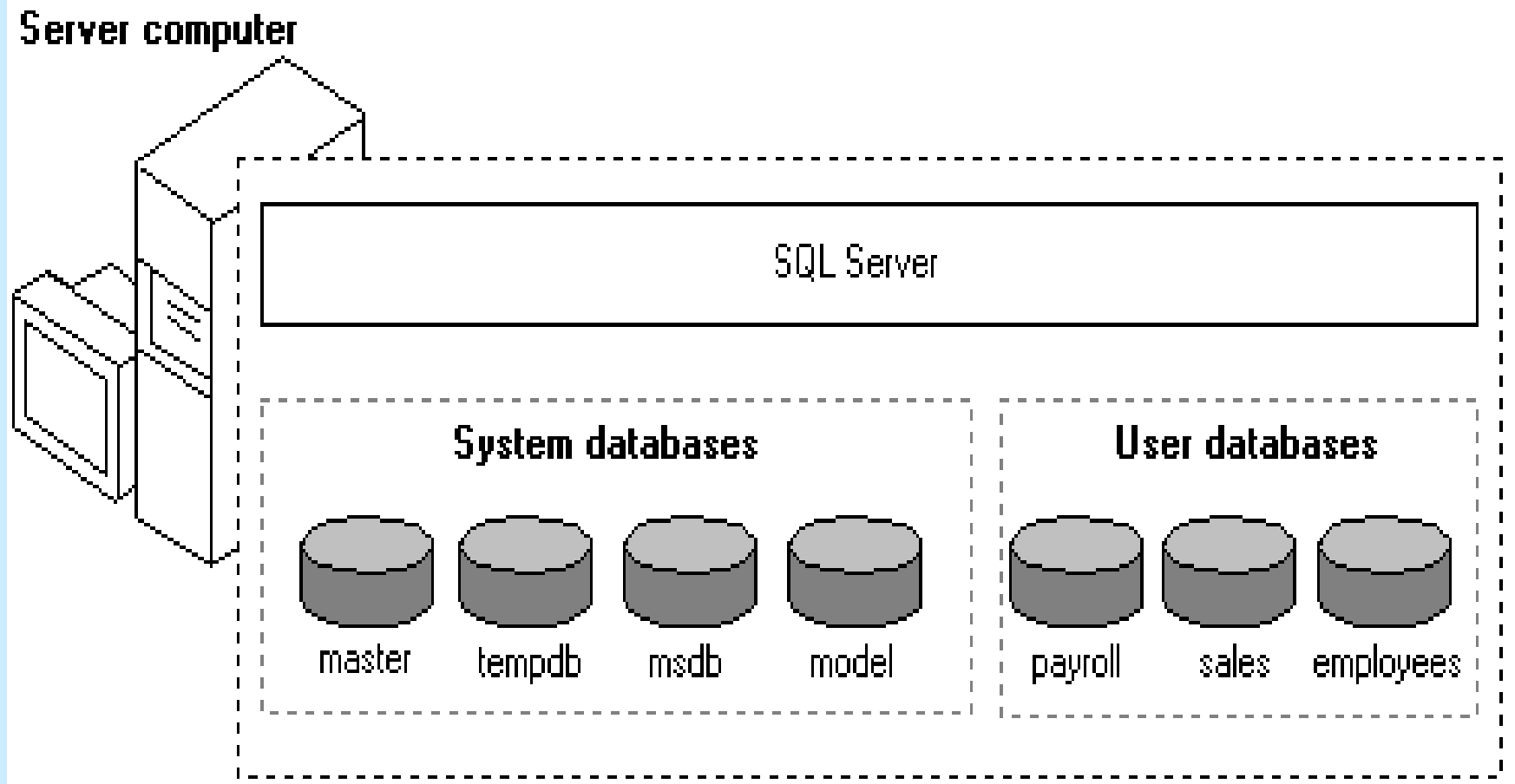
Grant



Arquitetura Cliente / Servidor



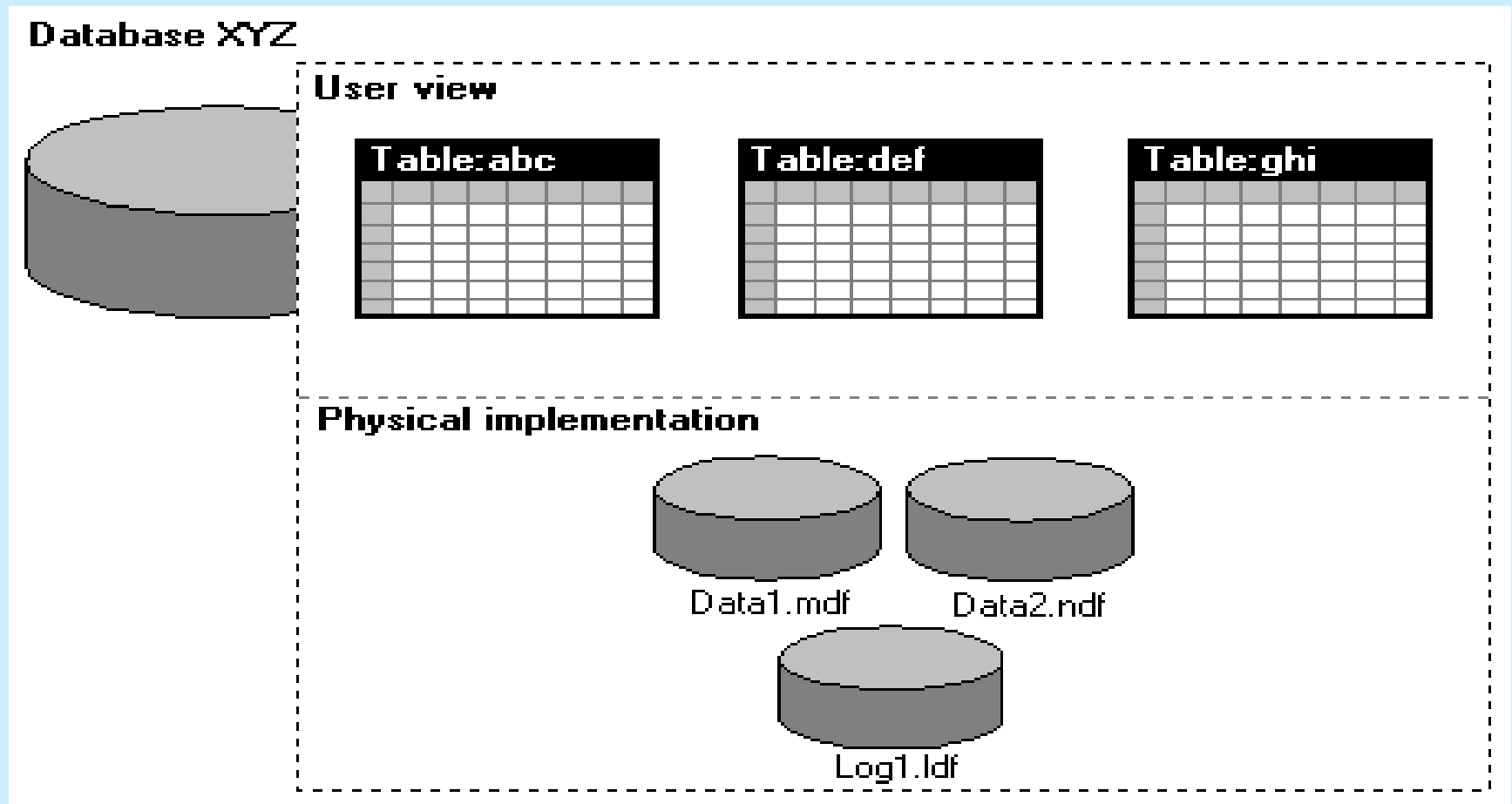
Arquitetura do MS SQL Server



Introdução: Arquitetura do MS SQL Server

- ⇒ **MASTER:** contem os dados operacionais do SQL-Server e dos usuários.
 - ⇒ Contas de login, Msgs erros, databases devices, locks ativos, processos ativos,
 - ⇒ Catálogo do Sistema e Dicionário de Dados
 - ⇒ **MODEL:** protótipo para um novo BD (templates)
 - ⇒ user datatype, regras, defaults, STP, usuários
 - ⇒ **MSDB:** suporte ao SQL Executive Service
 - ⇒ schedule de tarefas, replicações, ger. Alertas
 - ⇒ **TEMPDB:** ações temporárias ou intermediárias
 - ⇒ Group by, Order by, Distinct, Cursores
-

Introdução: Banco de Dados - Físico



Consulta a Dados no Banco de Dados

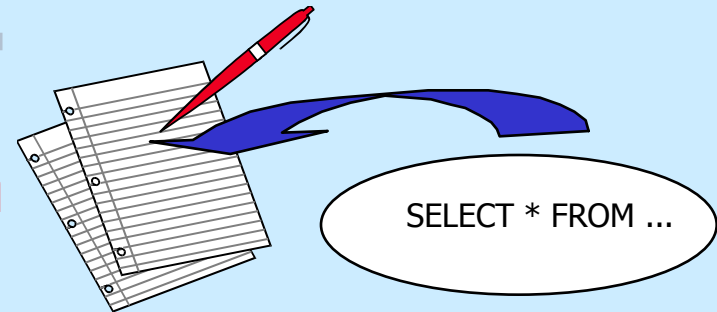
- ⇒ Linguagem mais usada: SQL
- ⇒ Todos os comandos de manipulação podem ser executados através de comandos SQL

⇒ **Interactive SQL**

- ISQLW
- Query Analyser



⇒ **Intercalados no programa**



Tipo de Dados do SQL SERVER

- **Bit** 0 ou 1
 - **Integer** 4 bytes binários: +/-2.147.483.648
 - **Smallint** 2 bytes binários: +/- 32.768
 - **Tinyint** 1 byte binário: 0 a 255
 - **Decimal(p,d)** precisão até 15 dígitos. (P - d = inteiros , d= decimais)
 - **Numeric(p,d)**
 - **Money** 8 bytes, +/- 922.337.203.685.477,5808
 - **Smallmoney** 4 bytes, +/- 214.748,3648
 - **Float (p)** valores em ponto flutuante: +/- 1.79E+308
 - **Real** valores em ponto flutuante: +/- 3,40E+38
 - **Datetime** data e hora em 4 bytes cada: dd/mm/aaaa hh:mm:ss,cc
 - **Smalldatetime** data e hora em 2 bytes cada: precisão de minutos
 - **Timestamp** hora em centena de milésimo de segundos. Garantir concorrência
 - **Char(n)** string com tamanho fixo de até 255 caracteres a 8000 caract.
 - **Varchar(n)** string com tamanho variável até 8000 caracteres
 - **Text(n)** string de tamanho fixo de até 2 Gb (2.147.483.647)
 - **Nchar(n)** string unicode fixo de até 4000 caracteres
 - **Binary (n)** string de bits de comprimento fixo até 8000 bytes
 - **Image(n)** string de bits de comprimento fixo até 2 Gb
-

Comandos: Structured Query Language

⇒ Pode ser:

⇒ DDL - Data Definition Language

Serve para definir as estruturas de dados.

⇒ DCL - Data Control Language

Serve para controle do Banco de Dados

⇒ DML - Data Manipulation Language

Serve para acesso e manipulação dos dados

Comandos: Data Definition Language

- ⇒ Servem para definição da estrutura de dados.
 - ⇒ **create database:**
criar Banco de Dados
 - ⇒ **create table:**
criar tabelas de dados
 - ⇒ **crate index:**
criar índices das tabelas
 - ⇒ **create view:**
criar visões de dados
-

DDL - Create Database

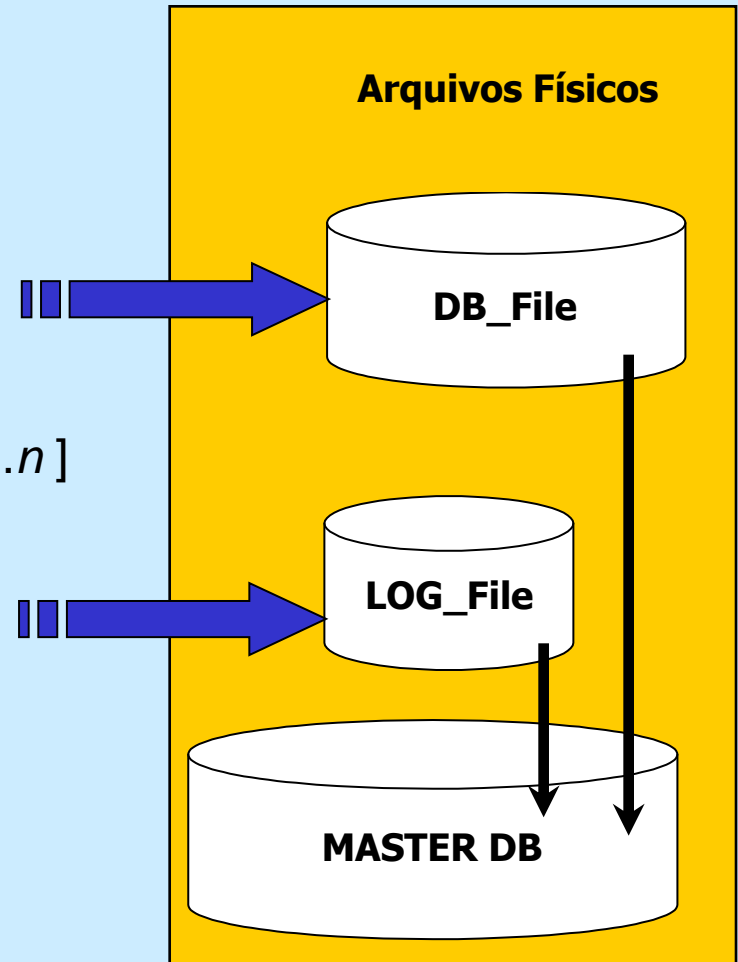
CREATE DATABASE *database_name*

ON

([NAME = *logical_file_name* ,]
FILENAME = '*os_file_name*'
[, SIZE = *size*]
[, MAXSIZE = { *max_size* | UNLIMITED }]
[, FILEGROWTH = *growth_increment*])
FILEGROUP *filegroup_name* < filespec > [,...*n*]

LOG ON

([NAME = *logical_file_name* ,]
FILENAME = '*os_file_name*'
[, SIZE = *size*]
[, MAXSIZE = { *max_size* | UNLIMITED }]
[, FILEGROWTH = *growth_increment*])



DDL - create database Exemplo

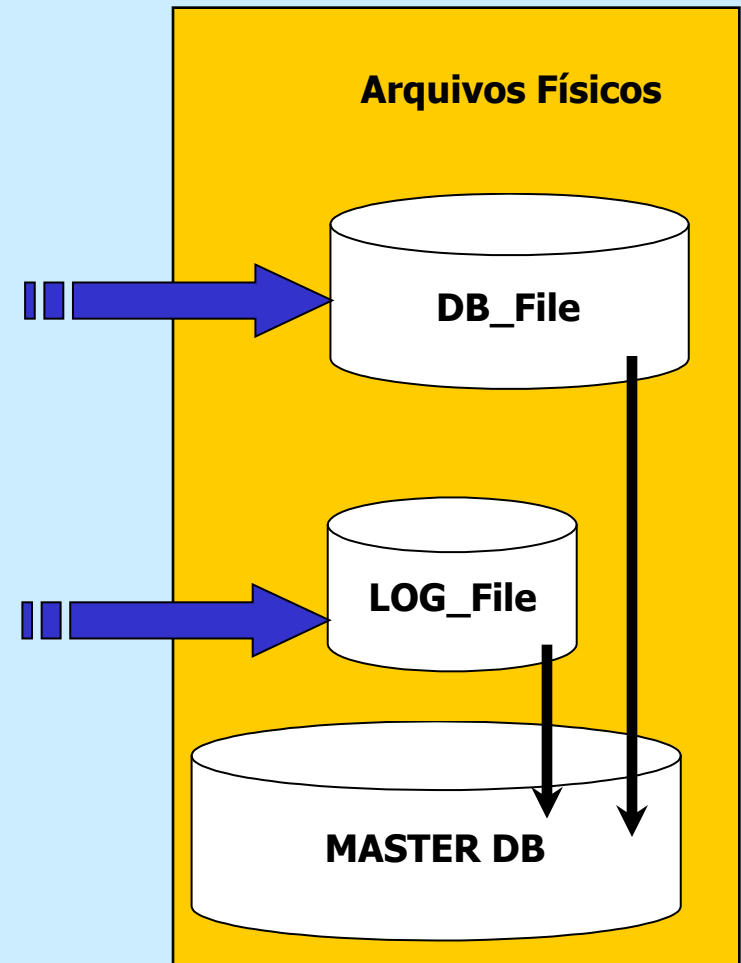
```
CREATE DATABASE db_curso
```

```
ON
```

```
( NAME = db_curso_data1,  
  FILENAME = 'c:\curso\db_curso.mdf',  
  SIZE = 2 Mb,  
  MAXSIZE = 10 Mb,  
  FILEGROWTH = 1 Mb )
```

```
LOG ON
```

```
( NAME = db_curso_log1,  
  FILENAME = 'c:\curso\db_curso.ldf',  
  SIZE = 1 Mb,  
  MAXSIZE = 5 Mb,  
  FILEGROWTH = 1 Mb )
```



DDL - create table

CREATE TABLE

```
[ database_name . [ owner ] . table_name
( { column_name1    data_type
    | column_name AS computed_column_expression
    [ DEFAULT constant_expression ] |
    [ IDENTITY ( seed , increment ) ]
    [ CONSTRAINT constraint_name ]
    { [ NULL | NOT NULL ] | [ PRIMARY KEY | UNIQUE ] }
    [ CLUSTERED | NONCLUSTERED ]
    [ WITH FILLFACTOR = fillfactor ]
    [ ON filegroup | DEFAULT ] ]
} | [ { PRIMARY KEY | UNIQUE } ]
[ column_name2,...n ] )

| [ FOREIGN KEY ] REFERENCES ref_table [ ( ref_column ) ]
    [ ON DELETE { CASCADE | NO ACTION } ]
    [ ON UPDATE { CASCADE | NO ACTION } ]
    [ NOT FOR REPLICATION ]
```

DDL - create table - Exemplo

```
USE db_curso  
go
```

```
CREATE TABLE empresa  
( cd_empresa      INTEGER      NOT NULL UNIQUE,  
  razao_social    VARCHAR(30)  NOT NULL,  
  cnpj_empr      NUMERIC(11 )  NULL,  
  endereco       VARCHAR(40)  NULL,  
  PRIMARY KEY (cd_empresa) )  
go
```

```
CREATE TABLE departamento  
( cd_depto      INTEGER      NOT NULL UNIQUE,  
  cd_empresa    INTEGER      NULL,  
  nm_depto      VARCHAR(40)  NOT NULL,  
  missao        VARCHAR(50)  NULL,  
  PRIMARY KEY (cd_depto),  
  FOREIGN KEY (cd_empresa) REFERENCES empresa )  
go
```

DDL - create index

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]

INDEX *index_name*

ON *nome_da_tabela*

(*coluna_1* [, *coluna_2* ,...*n*])

[**WITH** < **index_option** > [,...*n*]]

[**ON** *filegroup*]

index_option

{ FILLFACTOR = *fillfactor* |

IGNORE_DUP_KEY |

DROP_EXISTING |

SORT_IN_TEMPDB }

DDL - create index - Exemplo

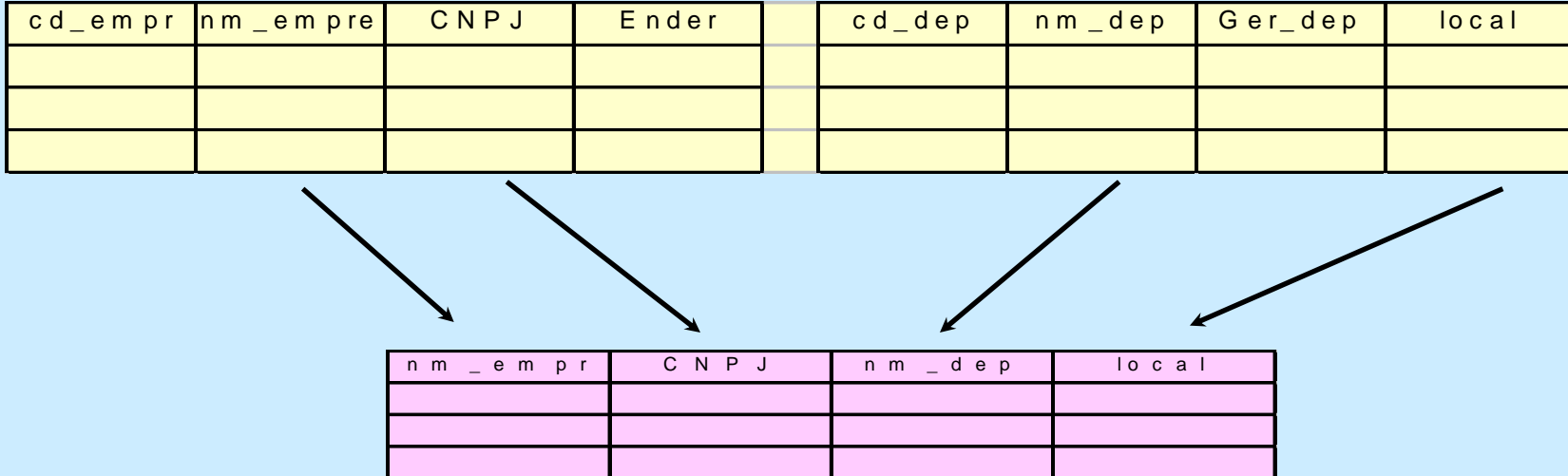
```
CREATE CLUSTERED INDEX XIF1empresa ON empresa  
( cd_empresa )
```

```
CREATE UNIQUE INDEX XIF2departamento ON departamento  
( cd_depto )
```

```
CREATE INDEX XIF3departamento ON departamento  
( cd_empresa, cd_depto)
```

DDL - View

CREATE VIEW [*database_name . owner . view_name*]
[(*column* [,...*n*])]
AS
select_statement



DDL - View - Exemplo


tb_empresa

cd_empr	nm_empre	CNPJ	Ender

tb_departamento

cd_dep	nm_dep	Ger_dep	local

vw_emp_dep



nm_empr	CNPJ	nm_dep	local

```
Create view vw_emp_dep  
(nm_empr, CNPJ, nm_dep, local)  
as select a.nm_empre, a.CNPJ, b.nm_dep, b.local  
from tb_empresa a, tb_departamento b  
where b.cd_empr = a.cd_empr
```

DDL - DROP

⇒ para eliminar um objeto do Banco de Dados.

⇒ Sintaxe:

```
drop database db_name
```

```
drop table tbl-name
```

```
drop index idx_name
```

```
drop view vw_name
```

DDL - ALTER

⇒ para alterar um objeto do Banco de Dados.

⇒ Sintaxe:

alter table bd_nome.owner.tabele_nome

alter column nome_coluna

novο_tipo_dados

add nome_coluna **dados_coluna**

Ex.: Alter table cliente

add ender varchar(40) null

go

alter column cidade varchar(30)

DCL - Data Control Language

⇒ **COMMIT** para efetivar as operações de atualização no Banco de Dados.

As atualizações são executadas na memória, para garantir a integridade de dados no Banco de Dados. Após o comando, as atualizações são gravadas no Banco de Dados

ROLLBACK para desfazer as operações de atualizações, que ainda não foram efetivadas

DCL - Data Control Language

⇒ serve para conceder permissão aos objetos do Banco de Dados

⇒ sintaxe: **GRANT** { **ALL** | comando [,...n] }

TO *security_account* [,...n]

Ex.: grant create table to usuário
grant all to public
grant select, update, delete to Maria

DCL - Data Control Language

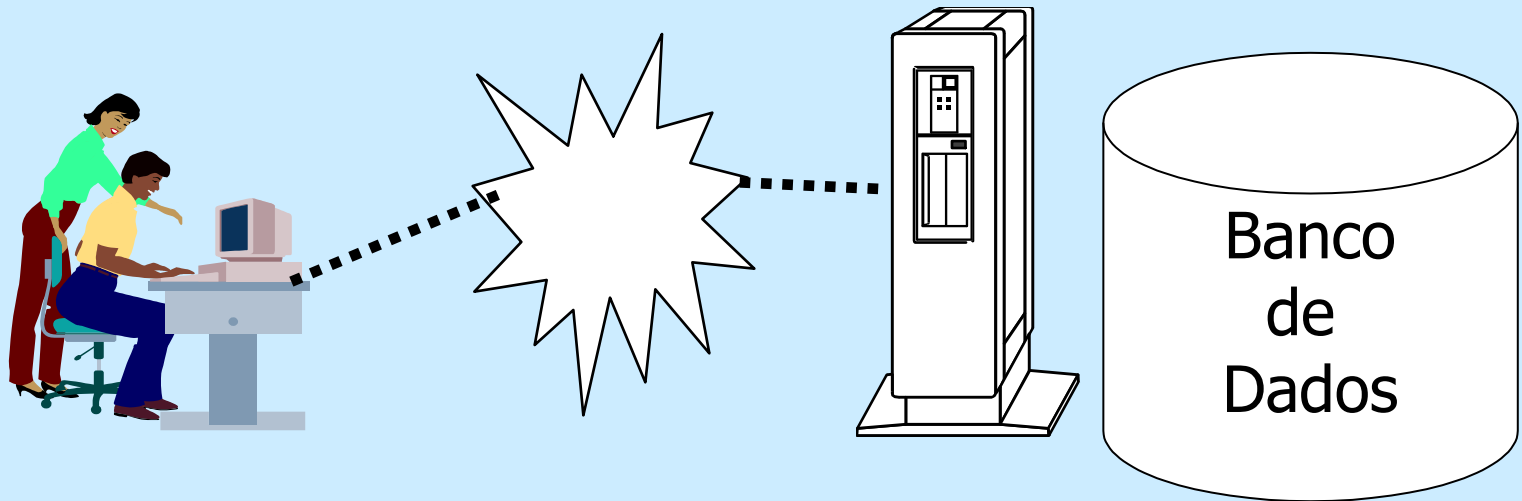
- ⇒ serve para revogar permissões para os objetos do Banco de Dados
- ⇒ sintaxe: **REVOKE** { **ALL** | *comando* [,...*n*] }

FROM *security_account* [,...*n*]

Ex.: revoke all from public
revoke create table from usuário
revoke create database from public

DML - Data Manipulation Language

⇒ Serve para acessar e manipular os dados do Banco de Dados



Operadores

⇒ **Aritméticos:**

adição:	+
subtração:	-
multiplicação:	*
divisão:	/

⇒ **Lógicos:**

e	AND
ou	OR
não	NOT

Comando select

⇒ Comando para consulta a dados

⇒ sintaxe:

```
select <coluna1>, <coluna2>, ..., <colunan>  
  from <tabela1>, <tabela2>, ... <tabelam>  
  [where <condição>]
```

select

- ⇒ No **select**, escolhemos quais colunas compõem o resultado
- ⇒ As colunas devem pertencer as tabelas identificadas no "**from**"
- ⇒ Quando há colunas com o mesmo nome, deve qualificá-las colocando o nome da tabela antes da coluna, separado por um ponto.

Ex.: **tb_empresa** . cd_empresa,
tb_departamento . cd_empresa

select

⇒ Quando desejamos escolher todas as colunas de uma tabela, usamos o asterisco (*)

```
select * from funcionários
```

```
select * from alunos
```

select

⇒ Podemos colocar constantes, e operações básicas (+, -, *, /) entre colunas

```
select salário * 10  
      from funcionários
```

```
select nome, 1999      from alunos
```

select

- ⇒ Para eliminar duplicidades devemos usar o comando **distinct**
- ⇒ O comando **all** faz com que as duplicidades sejam mantidas (default)

```
select distinct nome from Alunos
```

Obs.: lista apenas os nomes distintos

Cláusula where

- ⇒ Define as condições de filtragem do conjunto resultante do select
 - ⇒ Podem incluir comparações entre colunas e entre colunas e constantes
 - ⇒ Todas as colunas participantes devem pertencer à tabelas do comando from
-

Exemplos

```
Select * from funcionários  
      where salário > 1200,00
```

```
select código, nome from alunos  
      where curso = "computação"
```

where

- ⇒ Condições múltiplas podem ser formadas com conectivos and e or
 - ⇒ O uso de parênteses é permitido (e em alguns casos, recomendado)
 - ⇒ Podem ser usados os operadores <, >, <=, >=, <> (diferente de), = e like
-

exemplos

```
select nome, salário
      from funcionários
     where salário > 1000
    and   depto      =      1
```

```
select      *      from alunos
     where datanasc < "01/01/90"
    and   datanasc > "31/12/79"
```

between

⇒ Para definir intervalos, pode ser usado o comando “between” “and”

```
select *  
  from funcionários  
 where  
       salário between 1000 and 2000
```

Not between

⇒ Para valores fora de um intervalo, pode-se usar o not between and

```
select * from alunos
      where ingresso not between
              01/01/90 and 31/12/99
```

Comparador like

- ⇒ O comparador “like” pode ser usado em consultas em que não temos o valor exato de uma cadeia de caracteres para encontrar
 - ⇒ “%” significa qualquer substring
 - ⇒ “_” significa qualquer caractere posicional
-

exemplos

Select * from funcionários

where nome like "José%"

(inicia com Jose ...

Ex.: Jose de Souza)

select * from clientes

where nome like "%Souza"

(cliente que termina com Souza

Ex.: Jose de Souza)

exemplos

Select * from alunos

where nome like " _ _ _ _ Paula%"

(aluna Ana Paula da Silva.

5a. Posição em diante = Paula)

select * from clientes

where nome like "%Pedro%"

(seleciona João Pedro da Silva)

Order by

- ⇒ As linhas aparecem ordenadas pela chave primária, se esta estiver definida.
- ⇒ Pode-se ordenar por qualquer coluna através do comando order by

```
Ex.: select cd_aluno, nome, curso  
        from alunos  
        order by nome
```

Order by

- ⇒ O default do order by é a ordenação crescente (default)
- ⇒ Pode-se inverter a ordem através do "desc" (descendente)

```
select cd_aluno, nome, curso  
      from alunos  
      order by nome desc
```

Order by

⇒ Em caso de “igualdade”, a ordem é arbitrária, a não ser que sejam definidas mais colunas para o order by

```
select nome, data_nasc  
      from pessoas  
      order by nome, data_nasc
```

Renomear

⇒ Podemos renomear colunas ou tabelas no comando SQL, somente para o comando em execução, através do as

```
select nome as "Nome Aluno"  
from alunos
```

(o cabeçalho da coluna será Nome Aluno)

Cláusula from

- ⇒ Várias tabelas podem ser colocadas no from
- ⇒ É feito um produto cartesiano entre as linhas das tabelas do from

```
select *  
  from funcionários, departamentos
```

Junção

⇒ Para conseguir uma consulta que faz junção de duas tabelas ligadas, é preciso colocar a declaração de junção no where

```
select *  
  from funcionário f, departamento d  
 where d.cod_dep = f.cod_dep
```

Junções

⇒ Outras condições podem ser feitas

```
select *  
  from funcionários, departamentos  
 where cod_dep >= codigo
```

Junções e condições

⇒ Deve especificar as condições acrescentando-se as junções, através do "and"

```
select *  
  from funcionário f, departamento d  
 where f.cod_dep = d.codigo  
 and    f.salário > 1000
```

União

- ⇒ Podemos unir o resultado de duas consultas com o comando “**union**” as colunas do **select** devem ser compatíveis (mesmo tipo) nas consultas participantes
 - ⇒ Retorna uma tabela com as linhas das duas tabelas. Repetições são excluídas.
-

União - exemplo

```
select nome, datanasc  
from alunos  
where curso = "computação"  
union  
select nome, datanasc  
from funcionários  
where salário < 1000
```

Union all

⇒ Se colocado, faz com que sejam exibidas todas as repetições

```
select nome, datanasc
from alunos
where curso = "computação"
union all
select nome, datanasc
from funcionários
where salário < 1000
```

Funções de agregação

- ⇒ Min - o menor valor
 - ⇒ max - o maior valor
 - ⇒ avg - a média dos valores
 - ⇒ sum - a soma dos valores
 - ⇒ count - contador das rows com valores \neq Null
 - ⇒ count(*) - contador das rows, inclusive Null
-

exemplos

```
select sum(salário) from funcionários
```

```
select count(*) from alunos
```

```
select min(data_ingresso) from alunos  
       where curso = "computação"
```

```
select avg(mensalidade) from cursos
```

agrupamentos

- ⇒ Quando não se quer agregar um valor total, mas sim agrupar as agregações de acordo com um valor. Ex: Salário médio por departamento. Compras totais por cliente. Número de clientes por estado, etc.
 - ⇒ Comando group by após o comando SQL
-

exemplos

```
select CPF, avg(saldo) from contas  
group by CPF
```

```
select cd_produto, nome, avg(preço)  
from Produtos  
group by cd_produto
```

having

- ⇒ Usamos a cláusula “**having**” quando desejamos uma condição sobre a função de agregação.
 - ⇒ Para esse fim, não podemos colocar essa condição no where
 - ⇒ O having vem depois do group by
-

exemplos

```
select CPF, avg(saldo) from contas  
group by CPF  
having avg(saldo) > 5000
```

seleciona as contas agrupadas por CPF cuja média dos saldos seja maior que 5000

```
select Dep, sum(salário) from funcionários  
group by Dep  
having sum(salário) < 20000
```

seleciona soma dos salários de funcionários agrupados por departamentos e menores que 20000

Subselect

- ⇒ Consultas dentro da condição where, usadas como conjuntos, e em outras situações
 - ⇒ inclusão em consulta:
 - ⇒ in
 - ⇒ not in
 - ⇒ colocados no where, verifica se a coluna está ou não no resultado da consulta
-

Exemplo

**select * from conta_correntes
where cpf in (select cpf from poupanca)**

O conjunto resultante de cpfs de poupança servirá como dados de seleção para conta_correntes, ou seja, conta_correntes que tem poupança.

**select * from conta_correntes
where cpf not in (select cpf from poupanca)**

Selecionar as conta-correntes que não tem poupança, ou seja, o conjunto resultante de cpfs de poupança servirá como dados de seleção para conta_correntes

Mais exemplos

```
select nome from conta-Correntes  
where (cpf, nome, num_ag) in  
(select cpf, nome, num_ag from poupança)
```

```
select nome from alunos  
where nome not in ("Jones", "James")
```

Alterações nos dados

- ⇒ Exclusão: comando delete
 - ⇒ Alteração: comando update
 - ⇒ Inclusão: comando insert
-

Comando delete

⇒ Exclui dados de uma tabela (somente uma)

⇒ sintaxe

delete

from <tabela>

[where <condição>]

delete

⇒ Quando sem condição where, apaga todas as linhas da tabela

`delete from funcionários`

`delete from produtos`

`delete from alunos`

`delete from fornecedores`

delete

- ⇒ Quando usado com o where, pode especificar quais linhas apagar
 - ⇒ O where do delete comporta quase todas as condições possíveis do select, desde que seja somente referente a uma tabela
 - ⇒ subconsultas são permitidas
-

delete - exemplos

delete from funcionarios

where chapa = 111231

delete from funcionarios

where salário between 1000 and 2000

delete from funcionarios

where funcionarios.cd_chapa

in (select cd_chapa from dependentes

where data_nasc < 01/01/70)

insert

⇒ Para adicionar linhas na tabela, utiliza-se o comando insert

⇒ Para inserir uma linha:

insert into funcionários

values(8, “Milton”, 2000, 2)

⇒ Neste caso, os valores correspondem à ordem na definição da tabela

insert - especificando campos

⇒ Podemos definir que campos estamos inserindo:

```
insert into funcionarios( nome, chapa, salário, coddep)  
values( “Francisco”, 9, 3000, 2 )
```

⇒ Os campos não identificados no comando ficarão com valor nulo (campos definidos como null)

insert com select

⇒ Podemos incluir várias linhas na tabela com o uso do comando select:

insert into poupança

select numag, numconta, 200

from correntes

where numag = 1191

Atualizações

⇒ As atualizações das tabelas são feitas pelo comando update

⇒ sintaxe:

```
update <tabela>  
    set    <coluna>    =  <novo valor>  
    where  <condição>
```

update

- ⇒ O where é opcional. Na falta, atualiza todas as linhas da tabela.
 - ⇒ A expressão de atualização pode envolver a própria coluna.
 - ⇒ O where pode ter qualquer condição válida, inclusive subconsultas. O where é avaliado antes das atualizações, uma vez que pode envolver valores sendo alterados.
-

Exemplos de update

⇒ Atualizar todos os salários, aumentando 2%

```
update funcionários
```

```
    set salário = salário * 1.02
```

Poderíamos colocar um valor fixo:

```
update funcionários
```

```
    set salário = 10000
```

Update com where

⇒ No exemplo a seguir, damos aumento de 5% aos funcionários que recebem menos que 1000:

```
update funcionários  
    set salário=salário * 1,05  
    where salário < 1000
```

view

- ⇒ Nem sempre todos podem ter acesso a todo o esquema do banco de dados.
 - ⇒ Com as visões, podemos definir partes ou tabelas juntas, que o usuário vê como se fosse uma tabela normal
 - ⇒ Podemos consultar e atualizar dados através de visões, com algumas restrições.
-

view em SQL

⇒ Uma visão é definida a partir de uma consulta em SQL.

⇒ Sintaxe:

`create view visao as <consulta>`

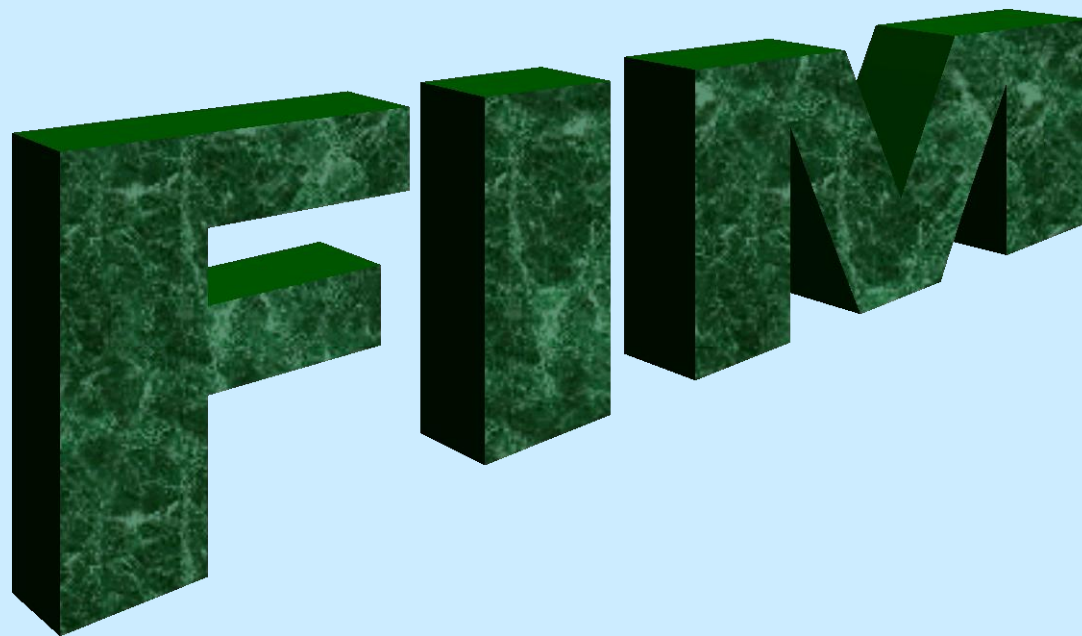
⇒ Usando visões podemos restringir acesso a determinadas colunas.

Exemplos

```
create view alunos_website as
(select *   from alunos
 where disciplina = "website")
```

```
create view clientes as
(select CPF, nome   from clientes)
```

```
create view salario_total( nomedep, total )
as ( select nomedep, sum (salário)
      from (funcionários inner join
            departamentos on coddep = codigo)
      group by nomedep )
```



Cristiane Tuji da Silva
cris_tuji@yahoo.com.br
