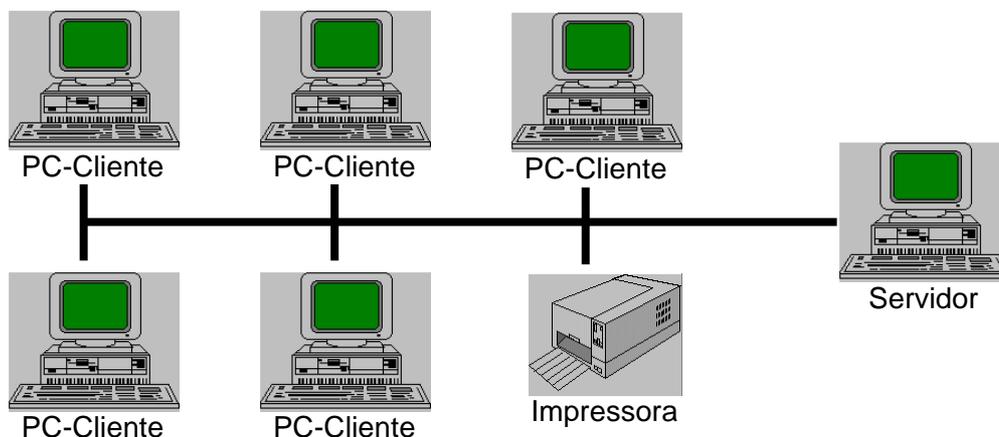


## ARQUITETURA CLIENTE/SERVIDOR



**Exemplo simples de Arquitetura Cliente/Servidor baseada em Rede**

### **BANCOS DE DADOS BASEADOS NA ARQUITETURA CLIENTE/SERVIDOR:**

Uma simples rede (LAN), como a exemplificada acima, é suficiente para comportar um banco de dados relacional (DBMS) de tecnologia Cliente/Servidor (C/S) como por exemplo o Oracle.

Em linhas gerais, esta tecnologia tem como característica principal a divisão de tarefas entre o **cliente**, a estação de trabalho que ordena através das aplicações o acesso aos bancos de dados, e o **servidor**, que executa tarefas, tais como: atualizações, deleções, procura de dados e todas as outras tarefas próprias do gerenciamento de banco de dados, porém, sob as ordens da estação de trabalho (**Cliente**).

A vantagem é evidente: dividindo o processamento em dois sistemas, temos de saída a diminuição do tráfego de dados na rede. Com isto, o desempenho aumenta pois evitaremos de processar os dados, fazendo-os transitar pela rede, entre a estação de trabalho e o servidor, pelo menos duas vezes. Ao invés disso, armazenamos os dados variáveis do processo em alguns parâmetros e os enviamos ao servidor. Estes ao chegarem são recepcionados pelo Oracle que os envia para Stored Procedure, que então inicia o processamento desejado até seu final dentro do servidor, limitando-se a avisar a estação de trabalho o término do processo, com sucesso ou não.

Porém, nem tudo são flores, existem também as desvantagens e a principal delas é o fato das estações de trabalho (Clientes) se localizarem em pontos geográficos distantes do servidor.

Embora este problema possa hoje ser minimizado pela adoção das arquiteturas de processamento distribuído, por outro lado não sem um considerável investimento em equipamentos, aplicativos auxiliares e a contratação de especialistas. Este investimento acompanhado de despesas de manutenção constantes embora não signifique propriamente um retorno aos custos de um computador de grande porte, é porém um gasto significativo e que dependendo do tamanho e da complexidade da rede podemos até alcançar os custos de um grande porte.

## - **VANTAGENS DA TECNOLOGIA CLIENTE/SERVIDOR:**

### **1- SEPARAÇÃO DAS TAREFAS DE SOLICITAÇÃO E PROCESSO.**

A primeira é efetuada pela estação de trabalho e a última é feita no servidor, ou seja: as tarefas de tratar e manipular os dados. Como já dissemos o tráfego na Rede diminui sensivelmente, pois só é entregue os dados necessários solicitados pela pesquisa do cliente, e estes depois de tratados são atualizados ao final da transação no servidor. Ao contrário dos Sistemas de Bancos de Dados sem a tecnologia Cliente/Servidor, que disponibiliza todo o banco de dados, indiferente a necessidade quando da solicitação pelo Cliente. A tecnologia Cliente/Servidor é antes de tudo uma incrementadora de performance sem igual.

### **2- INDEPENDÊNCIA DA ESTAÇÃO DE TRABALHO.**

Os usuários não ficam restritos a um tipo de sistema ou plataforma.

### **3- PRESERVAÇÃO DA INTEGRIDADE DOS DADOS**

Mesmo quando são efetuados Back-ups em tempo real ou até a encriptação dos dados. Nestes casos o DBMS, utiliza o espelhamento dos dados enquanto eles são acessados, gravando sempre a última fotografia dos dados antes da cópia de segurança.

### **4- PROCESSAMENTO DE TRANSAÇÕES.**

A grande vantagem deste método é guardar durante um certo tempo tempo as modificações efetuadas no Banco de Dados. Podendo, recuperá-las em caso de queda de energia ou mesmo quando o usuário do Banco desiste da modificação.

## - **DESVANTAGENS:**

1- A maior delas é o aumento do custo administrativo e a contratação de pessoal especializado para dar suporte e manter o Banco de Dados sempre ativo. Nasce o profissional Administrador de Bancos de Dados (DBA).

2- O aumento do custo de hardware, também é significativo, pois parte integrante desta tecnologia Cliente/Servidor, exige a distribuição do processamento, quando a rede for grande.

3- Quando da utilização do Processamento Distribuído a complexidade aumenta, o número de equipamentos diversos também aumenta, e nem sempre podemos encontrar profissionais no mercado com um conhecimento tão diversificado

## UTILITÁRIOS ORACLE:

### O AMBIENTE:

Para manipular e acessar as estruturas de Bancos de Dados Relacionais criadas no Oracle, o produto possui um ambiente completo para estas atividades.

A base para este trabalho é a linguagem SQL, que através desta sintax, cria, modifica e manipula as estruturas de dados, tendo como componentes básicos do ambiente:

**SQL (Structured Language Query):** Linguagem básica e padrão, extremamente simples e muito próxima da língua inglesa falada de maneira simples e comum, faz a maior e mais expressiva parte do trabalho de criação e manutenção de estruturas de dados. Porém sua limitação é não ser procedural, ou seja, não permite agrupar as palavras chaves sob a forma de programas executáveis. É uma linguagem puramente interativa de construção e submissão de comandos individuais, embora os comandos sejam macro comandos muito poderosos.

**SQL\*PLUS:** É o ambiente que permite tornar procedural os comandos individuais do SQL. Este ambiente é composto, como veremos a seguir de um editor de textos, que permite reunir grupos de frases SQL, que podem ser gravadas sob a forma de arquivo e então executadas.

**ORACLE NAVIGATOR:** Conjunto de recursos por intermédio telas(forms), onde o desenvolvedor do Banco de Dados, cria, altera e exclui objetos. É uma tendência no que diz respeito a manutenção de objetos e no elaborar das estruturas.

**PL/SQL:** É a linguagem procedural do SQL do ORACLE, composta essencialmente de todos os comandos SQL padrão e mais um grupo adicional que permite utilizar o SQL de forma procedural.

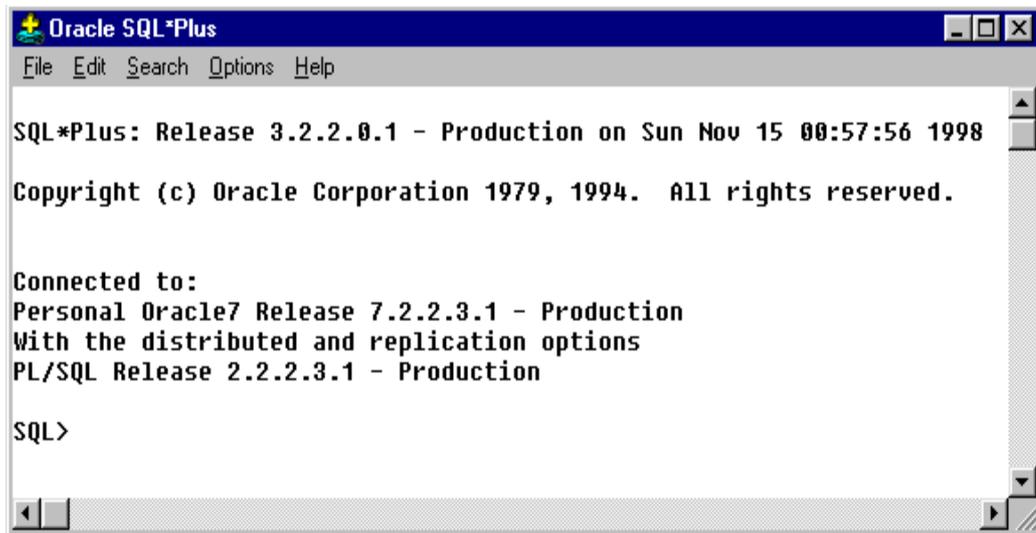


SQL Plus 3.2

Ao ser acionado com um duplo click, o editor é ativado solicitando para sua operação a identificação do usuário e seu código de acesso, seguido do “drive” lógico, que no caso do uso “Stand Alone”, foi convencionado 2: Caso estivermos trabalhando em uma rede local o drive lógico seria por exemplo o X:, mais as informações da localização física do Banco de Dados.

Log On	
User Name:	SCOTT
Password:	*****
Host String:	2:
OK Cancel	

Realizada a identificação com sucesso, o editor é liberado para uso, mostrando o prompt SQL> a espera de qualquer comando válido para a execução. A seguir mostramos a característica do editor e o ambiente que permite o trabalho com o SQL de forma procedural.



## BREVE INTROÇÃO A UTILIZAÇÃO DO SQL PLUS E A LINGUAGEM SQL

1. **Janela de Conexão** – Scott / Tiger
2. `Select * From User_Objects;`
3. `Select * From  
User_Objects` Where  
`Object_Type = 'TABLE';`
4. **Describe (ver Estrutura)** Describe DEPT
5. `Select * From DEPT;`
6. `Select * From Emp;`
7.
 

**Relacionamento**  
`Select d.dname, e.ename From dept d, emp e  
 where d. Deptno = e. Deptno;`
8.
 

**Agrupar**  
`Select deptno, min (sal), max (sal), sum (sal)  
 from emp group by deptno;`

`Select JOB, sum (sal) from emp group by JOB`
9.
 

**Insert**  
`Insert into dept values(35, 'Estoque', 'Madureira');`

10. **Update**

Update dept set loc = 'Centro' where deptno = 35;

11.

**Delete**

Delete from dept where dnome = 'Estoque'

12. Commit

13. Rollback

14. Connect (System/Manager)

15. Disconnect

## **PASSOS BÁSICOS PARA UTILIZAÇÃO DO ORACLE**

### **1. Conectar como DBA – System Manager (Criar Usuário)**

1.1. Create user AlunoX Identified by A123;

1.2. Grant connect to AlunoX;

### **2. Conectar com usuário – Scott / Tiger (Dar Privilégio)**

2.1. Grant Select on Dept to AlunoX;

2.2. Connect AlunoX / A123;

2.3. Select \* From Scott.Dept;

### **3. Conectar como System/Manager (Criar Papel e Dar Privilégio ao mesmo)**

3.1. Create Role PapelAlunoX;

3.2. Connect Scott /Tiger;

3.3. Grant Insert, Delete, Update on Dept to PapelAlunoX;

3.4. Grant Select on Emp to PapelAlunoX;

### **4. Conectar como DBA – System / Manager (Atribuir Papel ao Usuário)**

4.1. Grant PapelAlunoX to AlunoX;

### **5. Conectar como Usuário – Scott /Tiger (Criar Sinônimo)**

5.1. Create Public Synonym Empregado for Scott.Emp;

5.2. Select \* From Empregados;

### **6. Tópicos Diversos**

6.1. Ver qual usuário

conectado

Select user from

Dual;

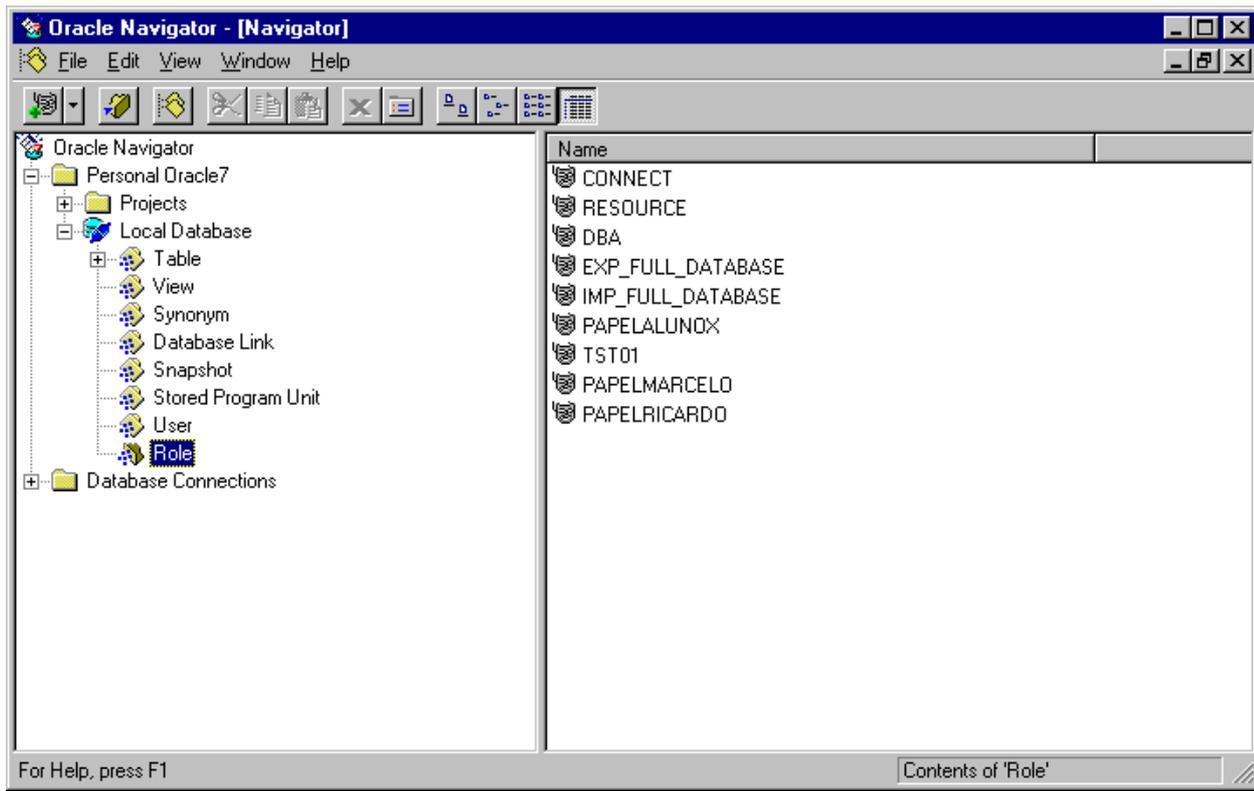
6.1.1. Dar Privilégio à Usuário de Criar Tabelas e Derivados

6.1.2. Grant Resource to AlunoX; (Conecatodo como System)

6.1.3. Connect AlunoX / A123;

# ORACLE NAVIGATOR

## IMPLEMENTAÇÃO DA ESTRUTURA BASEADO EM INTERFACE VISUAL



### -Personal Oracle

Nó que identifica o Banco de Dados Local. Suas subdivisões são: **Projects** e **Local Database**, onde Projects corresponde ao nó onde agrupamos para efeito de transferência (exportar para um Servidor Oracle) um projeto de banco de dados. E Local Database, corresponde ao nó que agrupa todos os objetos de um banco de dados. É a partir destes elementos, que teremos acesso as diversas interfaces para desenvolvimento facilitado (codificação abreviada).

### Database Connection

Agrupa os links possíveis para gerenciadores que externos ao ambiente local.

## Objeto User

**New User Properties** [?] [X]

General | Role/Privilege

 Name:   
Type: User

Password:  
New:   
Confirm:

OK Cancel

**New User Properties** [?] [X]

General | Role/Privilege

Show  
 Roles  
 Privileges

Granted:

-  CONNECT
-  RESOURCE

Remaining:

-  DBA
-  EXP\_FULL\_DATABASE
-  IMP\_FULL\_DATABASE
-  PAPELALUNOX
-  PAPELMARCELO
-  PAPELRICARDO
-  TST01

< << > >>

OK Cancel

**Nota:** Acionado a partir do click do botão direito do mouse, quando sobre este nó.

### **Aba General**

**Propriedade Name -** identificador do usuário

**Propriedade New -** password(iniciado por caracter Alpha seguido de Alpha-Numérico)

### **Aba Role/Privilege**

Define os privilégios de acesso e manipulação atribuídos ao usuário

## Objeto Roles

The image shows a 'New Role Properties' dialog box with two tabs: 'General' and 'Role/Privilege'. The 'General' tab is active. It contains a 'Name' field with the text 'PapelAluno' and a 'Type' field with the text 'Role'. There is a small icon of a person's face to the left of the 'Name' field. Below these fields is a checkbox labeled 'Password Protected' which is currently unchecked. Underneath the checkbox are two text input fields, one labeled 'New:' and one labeled 'Confirm:'. At the bottom of the dialog are two buttons: 'OK' and 'Cancelar'.

**Nota:** Acionado a partir do click do botão direito do mouse, quando sobre este nó. Este objeto, visa receber por atribuição, privilégios, criando um grupo de atribuições, que posteriormente agregará privilégios aos usuários que tiverem este papel atribuído a sua lista de privilégios..

### Aba General

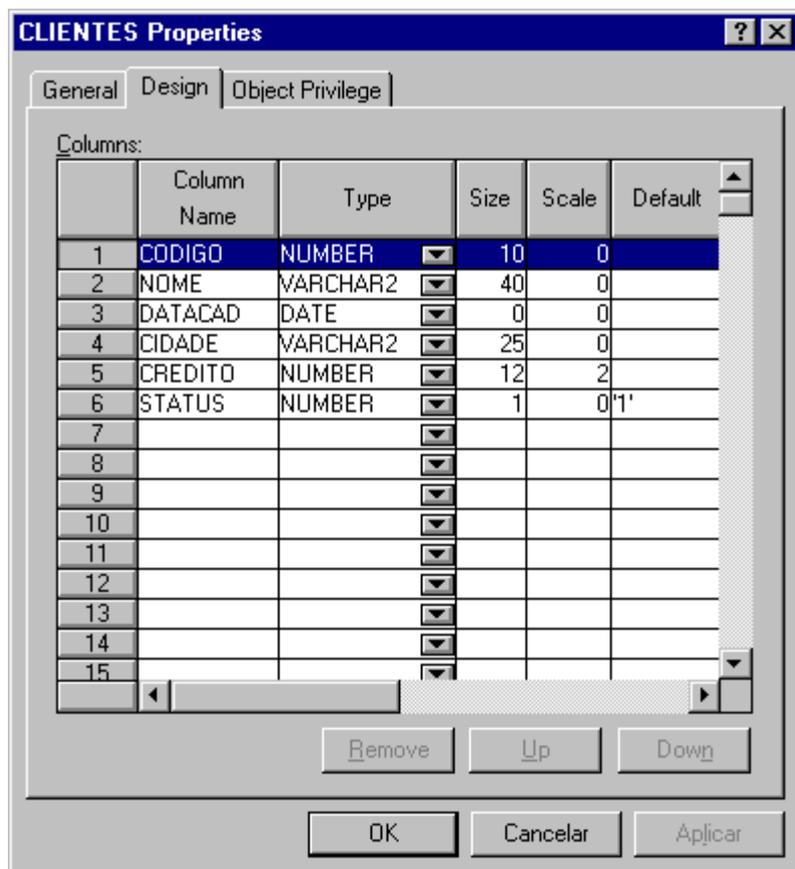
**Password protected**  
**Propriedade Name -**

Quando assinalado, habilita definição de senha(New/Confirm)  
Nome do objeto papel

### Aba Role/Privilege

Define os privilégios de acesso e manipulação atribuídos ao usuário

## Objeto Table



**Nota:** Acionado a partir do click do botão direito do mouse, quando sobre este nó. De forma bastante prática, temos uma visão de toda a estrutura da tabela.  
A sintaxe necessária para execução via script ou código entrado pelo SQL Plus, segue:

### Aba General

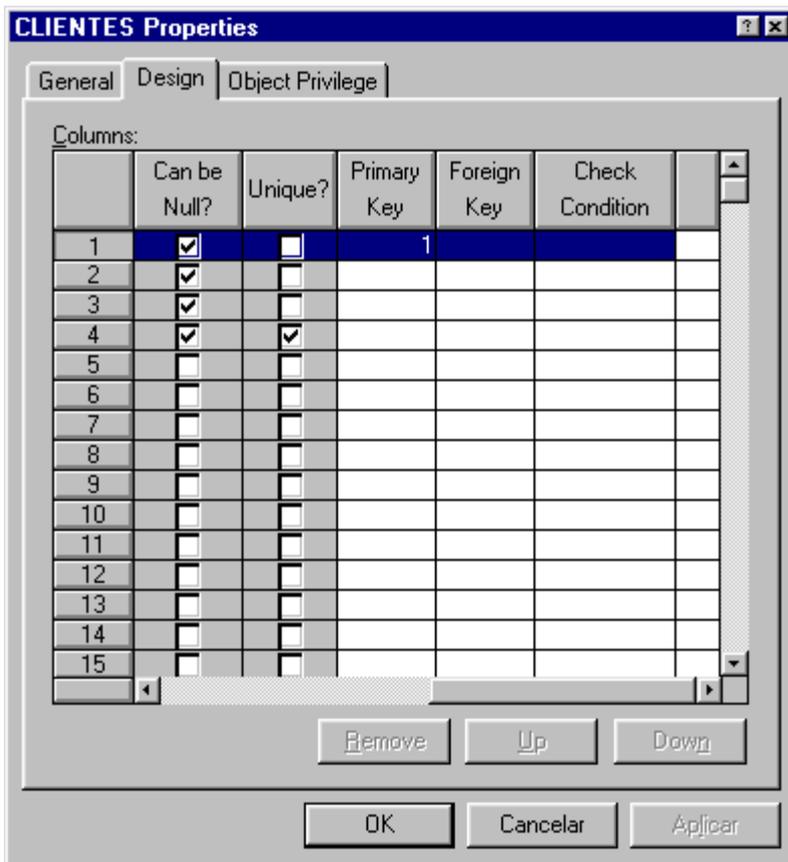
**Name** - Nome da tabela a ser criada  
**Owner** - Usuário a quem pertence

**Aba Object Privilege** - Definição dos privilégios

### Aba Design

**Column Name** - Nome do campo  
**Type** - Acionando o combo é apresentado uma lista para escolha  
**Size** - Define-se o tamanho do campo  
**Scale** - Quando tipo number, indica quantas casas decimais serão reservadas  
**Default** - Valor entrado como default quando do não preenchimento

## Aba Design(continuação)



- Can Be Null ? -** Quando assinalado, o campo aceitará valor nulo
- Unique ? -** Quando assinalado, cria restrição para valores duplicados nesta coluna
- Primary Key -** Indica que o(s) campo(s) faz(em) parte da chave primária
- Foreign Key -** Cria restrição, indicando o campo como chave estrangeira, referenciado um campo chave em outra tabela
- Check Condition -** Atribuição de uma condição para validação do valor entrado no campo

-----Script das Tabelas do Projeto-----  
 ---

```

CREATE TABLE CLIENTES (
  CODIGO      NUMBER(10, 0)          NOT NULL,
  NOME        VARCHAR2(50)         NOT NULL,
  DATA       DATE                  NOT NULL,
  STATUS      NUMBER(1, 0)          DEFAULT '1' NOT NULL,
  CREDITO     NUMBER(12, 2),
  HISTORICO   LONG,
  TIPO        VARCHAR2(1)          DEFAULT 'J' NOT NULL,
  CGC         VARCHAR2(14)         NOT NULL)

CREATE TABLE CLIENTESFOTO (
  CODIGO      NUMBER(10, 0)          NOT NULL,
  FOTO        LONG RAW)
  
```

```
CREATE TABLE PRODUTOS (  
  CODIGO          NUMBER(10, 0)    NOT NULL,  
  DESCRICAO      VARCHAR2(25)    NOT NULL,  
  SALDO          NUMBER(10, 0),  
  PRECOVENDA    NUMBER(13, 2))
```

```
CREATE TABLE ITENS (  
  NUMEROPED      NUMBER(10, 0)    NOT NULL,  
  CODPRODUTO     NUMBER(10, 0)    NOT NULL,  
  QUANTIDADE     NUMBER(10, 0)    NOT NULL,  
  PRECOVENDA    NUMBER(13, 2)    NOT NULL)
```

```
CREATE TABLE PEDIDOS (  
  NUMERO         NUMBER(10, 0)    NOT NULL,  
  CODCLIENTE    NUMBER(10, 0)    NOT NULL,  
  DATAPED       DATE            NOT NULL)
```

```
CREATE TABLE VENDASCLI_TEMP (  
  CODIGO         NUMBER(10,0),  
  NOME           VARCHAR2(50),  
  TOTVENDAS     NUMBER(13, 2))
```

### **Criar Índices/Chave Primária**

---

```
CREATE INDEX IDEXNOME ON CLIENTES (NOME)  
CREATE UNIQUE INDEX PK_CLIENTES ON CLIENTES (CODIGO)  
CREATE UNIQUE INDEX PK_CLIENTESFOTO ON CLIENTESFOTO (CODIGO)  
CREATE UNIQUE INDEX UK_CLIENTESCGC ON CLIENTES (CGC)  
CREATE UNIQUE INDEX PK_ITENS ON ITENS (NUMEROPED, CODPRODUTO)  
CREATE INDEX IDXDATACLI ON PEDIDO (DATAPED, CODCLIENTE)  
CREATE UNIQUE INDEX PK_PEDIDO ON PEDIDO (NUMERO)  
CREATE INDEX IDXDESCRICAO ON PRODUTOS (DESCRICAO)  
CREATE UNIQUE INDEX PK_PRODUTOS ON PRODUTOS (CODIGO)
```

### **Criar Sequências**

```
Create Sequence Seq_Clientes;
```

```
Create Sequence Seq_Produtos;
```

```
Create Sequence Seq_Pedidos start with 1 increment by 2;
```

## **Alteração na Estrutura de Tabelas / Constraint / Chave Estrangeira**

Alter Table Clientes Modify (Nome char (42));

Alter Table Clientes Add Constraint Ck\_Clientescredito Check (Credito <= 50000)

Alter Table Clientes Add Constraint Nn\_Clientescgc Check (Cgc Is Not Null)

Alter Table Clientes Add Constraint Uk\_Clientescgc Unique (Cgc)

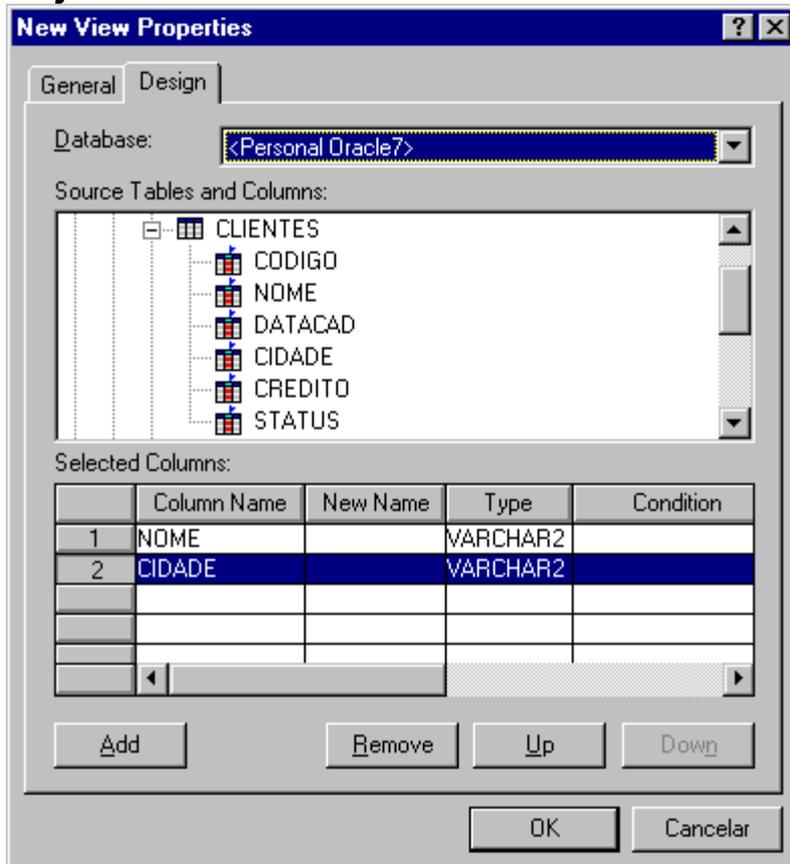
Alter Table Itens Add Constraint Sys\_C00577 Foreign Key (Numeroped)  
References Aulamanha.Pedido

Alter Table Itens Add Constraint Sys\_C00578 Foreign Key (Codproduto)  
References Aulamanha.Produtos

Alter Table.Pedido Add Constraint Sys\_C00576 Foreign Key (Codcliente)  
References Aulamanha.Clientes

Alter Table Clientes Add Constraint Clientescheckconstraint1 Check (Tipo In ('J', 'F'))

## Objeto View



**Nota:** Acionado a partir do click do botão direito do mouse, quando sobre este nó. Uma View conceitualmente se aplica para minimizar o acesso aos dados de uma tabela ou a composição de um join.

### Create View View\_Clientesfoto As

```
Select Clientes.Nome, Clientes.Foto  
From Clientes Where Clientes.Status = 1
```

### Create View View\_Itensdescricao As

```
Select Produtos.Descricao,Itens.Quantidade,Itens.Precovenda  
From Itens,Produtos  
Where Itens.Codproduto = Produtos.Codigo
```

### Create View View\_Pedidosdata As

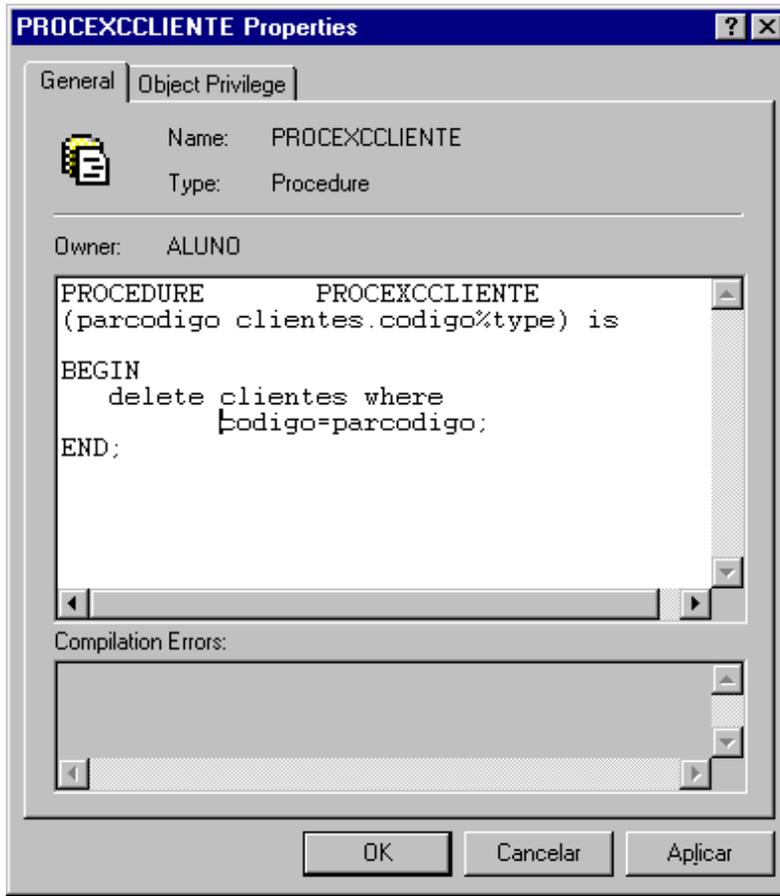
```
Select Clientes.Nome, Pedido.Dataped, Pedido.Numero  
From Aulamanha.Clientes, Aulamanha.Pedido  
Where Clientes.Codigo = Pedido.Codcliente And Pedido.Dataped >= Sysdate -
```

150

### Create View View\_VendaMesAno As

```
Select Trunc(Pedidos.DataPed, 'Month') As MesAno,  
Sum(Itens.Quantidade * Itens.Precovenda) As Total  
From Itens, Pedidos  
Where Pedidos.Numero = Itens.NumeroPed  
Group By Trunc(Pedidos.DataPed, 'Month')
```

## Objeto Procedure



### CREATE PROCEDURE PROCINCCLIENTE (

```
Parnome Clientes.Nome%Type,  
Pardata Clientes.Data%Type,  
Parstatus Clientes.Status%Type,  
Parcredito Clientes.Credito%Type,  
Parhistorico Clientes.Historico%Type,  
Parfoto Clientes.Foto%Type,  
Partipo Clientes.Tipo%Type,  
Parcgc Clientes.Cgc%Type) Is
```

BEGIN

```
Insert Into Clientes Values ( Seq_Clientes.Nextval,  
Pardata,  
Parnome,  
Parstatus,  
Parcredito,  
Parhistorico,  
Parfoto,  
Partipo,  
Parcgc );
```

END;

```
CREATE PROCEDURE PROCALTCLIENTE (  
    Parcodigo    Clientes.Codigo%Type,  
    Parnome      Clientes.Nome%Type,  
    Pardata      Clientes.Data%Type,  
    Parstatus    Clientes.Status%Type,  
    Parcredito   Clientes.Credito%Type,  
    Parhistorico Clientes.Historico%Type,  
    Parfoto      Clientes.Foto%Type,  
    Partipo      Clientes.Tipo%Type,  
    Parcgc       Clientes.Cgc%Type ) Is
```

```
BEGIN
```

```
    Update Clientes Set
```

```
        Clientes.Nome = Parnome,  
        Clientes.Data = Pardata,  
        Clientes.Status = Status,  
        Clientes.Credito = Parcredito,  
        Clientes.Historico = Parhistorico,  
        Clientes.Foto = Parfoto,  
        Clientes.Tipo = Partipo,  
        Clientes.Cgc = Parcgc
```

```
    Where Clientes.Codigo = Parcodigo;
```

```
END;
```

```
CREATE PROCEDURE PROCEXCCLIENTE (  
    Parcodigo Clientes.Codigo%Type) IS
```

```
BEGIN
```

```
    Delete Clientes
```

```
    Where Codigo = Parcodigo;
```

```
END;
```

## Criar Procedure com Cursor

```
CREATE PROCEDURE Proccursorregistro(Pardata In Pedidos.Dataped%Type) As
  Registro          Pedidos%Rowtype;
  Cursor           Cursorregistros Is
                    Select * From Pedidos
                    Where Pedidos.Dataped >= Pardata;

Begin
  Open Cursorregistro;
  Loop
    Fetch Cursorregistro Into Registro;
    Exit When Cursorregistro%Notfound;
  End Loop;
  Close Cursorregistro;
End;
```

```
CREATE PROCEDURE PROCVENDASCLI (
  Pardata Pedido.Dataped%Type,
  Parvalor Itens.Precovenda%Type) As

  Cursor Cursorvendascli Is
    Select Clientes.Nome, Sum(Itens.Quantidade * Itens.Precovenda) As Total
    From Clientes,Itens,Pedido
    Where Clientes.Codigo = Pedido.Codcliente And
           Pedido.Numero = Itens.Numeroped And
           Pedido.Dataped >= Pardata

    Group By Clientes.Nome
    Having Sum(Itens.Quantidade * Itens.Precovenda) >= Parvalor
    Order By Clientes.Nome;

  Varnome Clientes.Nome%Type;
  Vartotal Itens.Precovenda%Type;
BEGIN
  Delete Vendascli;
  Open Cursorvendascli;
  If Sql%Notfound Then
    Raise_Application_Error(-20000,'Não Há Vendas Neste Período.');
```

```
  End If;
  Loop
    Fetch Cursorvendascli Into Varnome,Vartotal;
    Exit When Cursorvendascli%Notfound;
    Insert Into Vendascli Values (Varnome, Vartotal);
  End Loop;
  Close Cursorvendascli;
End;
```

### Testando no SQL-Plus

```
SQL> Set ServerOutput on;
SQL> Execute ProcCursor1(1);
```

## Criar Função

```
CREATE FUNCTION Func_Totcompracli
(
    Parcodcli In Clientes.Codigo%Type,
    Pardata1 In Pedido.Dataped%Type,
    Pardata2 In Pedido.Dataped%Type
)
Return Number As
Vartotal Number (13,2);
Varcodigo Number (10);

BEGIN
Select
    Sum(Itens.Quantidade * Itens.Precovenda)
    Into Vartotal
From
    Itens,Pedido
Where
    Itens.Numeroped = Pedido.Numero And
    Pedido.Codcliente = Parcodcli And
    Pedido.Dataped Between Pardata1 And Pardata2
Return Vartotal;
END;

Testando no SQL Plus:    Select FuncTotRegistros(1) from Dual;
```

## Objeto Trigger

**TRIGGER\_TESTAVAL Properties**

General | Body

Name: TRIGGER\_TESTAVAL  
Type: Trigger

Owner: ALUNO  
Table: ALUNO.PRODUTOS

Triggering:  Before  After

Statements:  Insert  Delete  Update

CODIGO  
 DESCRICAO  
 SALDO  
 VALORVENDA

For Each Row

Referencing: OLD As: ANTERIOR NEW As: NOVO

When:

Enable?

OK Cancelar Aplicar

**CREATE TRIGGER TRIG\_AJUSTAPRECOVENDA  
BEFORE INSERT OR UPDATE ON ITENS FOR EACH ROW  
DECLARE**

```
Varprecocompra  
Produtos.Precocompra%Type;
```

**BEGIN**

```
Select Produtos.Precocompra Into Varprecocompra  
From Produtos Where Produtos.Codigo = :Novo.Codproduto;  
If :Novo.Precovenda < (Varprecocompra * 1.17) Then  
:Novo.Precovenda := (Varprecocompra * 1.17);  
End If;
```

**END;**

**TRIG\_INCITEM Properties**

General | Body

Name: TRIG\_INCITEM  
Type: Trigger

---

Owner: AULAMANHA  
Table: AULAMANHA.ITENS

---

Triggering:  Before  After  
Statements:  Insert  Delete  Update

For Each Row

Referencing: OLD As: ANTERIOR NEW As: NOVO  
When:

Enable?

OK Cancelar Aplicar

**CREATE TRIGGER TRIG\_INCITEM  
BEFORE INSERT ON ITENS FOR EACH ROW  
DECLARE**

Varsaldo produtos.saldo%type;  
VarDescricao produtos.descricao%type;

**BEGIN**

--Comentário

Select Produtos.Saldo,Produtos.Descricao Into Varsaldo,Vardescricao  
From Produtos Where Produtos.Codigo = :Novo.Codproduto;

If Varsaldo >= :Novo.Quantidade Then

Update Produtos Set

Produtos.Saldo = Produtos.Saldo - :Novo.Quantidade

Where Produtos.Codigo = :Novo.Codproduto;

Else

Raise\_Application\_Error( -20000, 'Ini O Saldo Do Produto ' ||  
Vardescricao || ' É De: ' ||  
To\_Char(Varsaldo) || ' Não  
Atendendo O Solicitado. Fim' );

End If;

**END;**

**TRIGGER\_ALTITEM Properties**

General | Body

Name: TRIGGER\_ALTITEM  
Type: Trigger

Owner: ALUNO  
Table: ALUNO.ITENS

Triggering:  
 Before  
 After

Statements:  
 Insert  
 Delete  
 Update

For Each Row

Referencing: OLD As: ANTERIOR NEW As: NOVO

When:

Enable?

OK Cancelar Aplicar

**CREATE TRIGGER TRIG\_ALTITEM  
BEFORE UPDATE ON ITENS FOR EACH ROW  
DECLARE**

```
Varsaldo produtos.saldo%type;  
VarDescricao produtos.descricao%type;
```

**BEGIN**

```
--Comentário  
Update Produtos Set  
Produtos.Saldo = Produtos.Saldo + :Anterior.Quantidade  
Where Produtos.Codigo = :Anterior.Codproduto;  
Select Produtos.Saldo,Produtos.Descricao Into Varsaldo,Vardescricao  
From Produtos Where Produtos.Codigo = :Novo.Codproduto;  
If Varsaldo >= :Novo.Quantidade Then  
    Update Produtos Set  
    Produtos.Saldo = Produtos.Saldo - :Novo.Quantidade  
    Where Produtos.Codigo = :Novo.Codproduto;  
Else  
    Raise_Application_Error(-20000, 'Ini O Saldo Do Produto ' ||  
    Vardescricao || ' É De: ' ||  
    To_Char(Varsaldo) || ' Não  
    Atendendo O Solicitado. Fim');
```

```
End If;
```

**END;**

**TRIGGER\_EXCITEM Properties** [?] [X]

General | **Body**

 Name: TRIGGER\_EXCITEM  
Type: Trigger

---

Owner: ALUNO  
Table: ALUNO.ITENS

---

Triggering:      Statements:

Before       Insert  
 **After**       **Delete**  
       Update

For Each Row

Referencing: OLD As: ANTERIOR    NEW As: NOVO

When:

Enable?

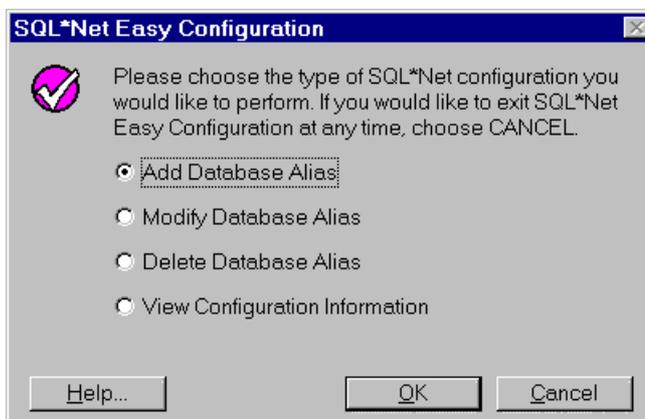
OK    Cancelar    Aplicar

**CREATE TRIGGER TRIG\_EXCITEM  
BEFORE DELETE ON ITENS FOR EACH ROW  
BEGIN**

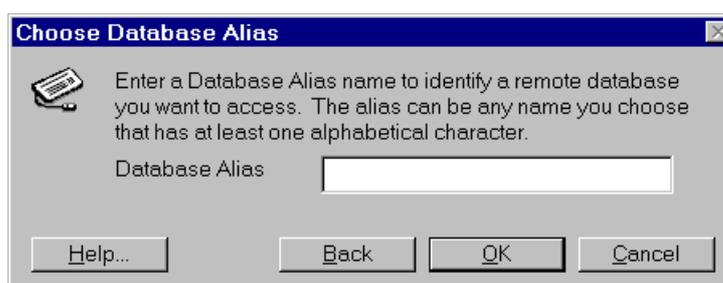
    Update Produtos Set  
    Produtos.Saldo=Produtos.Saldo + :Anterior.Quantidade  
    Where Produtos.Codigo = :Anterior.Codproduto;

**END;**

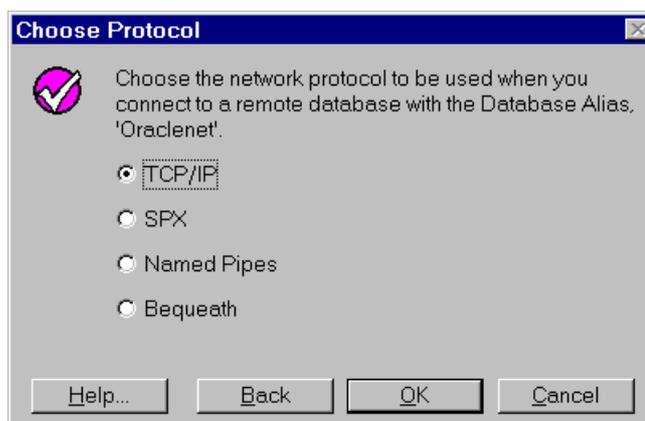
## Criar Conexão (Alias) no SQL Net e Drive ODBC32



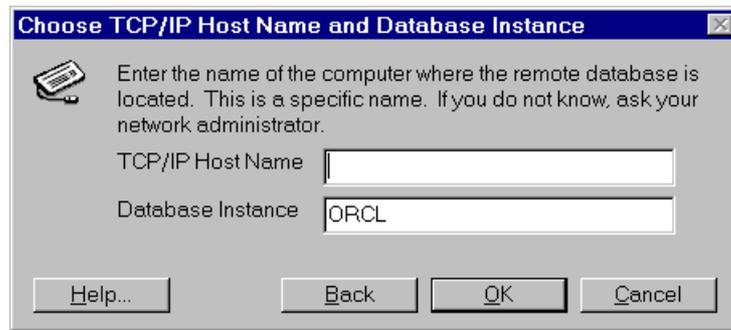
1. Indicar Add Database Alias, clicando no botão OK.



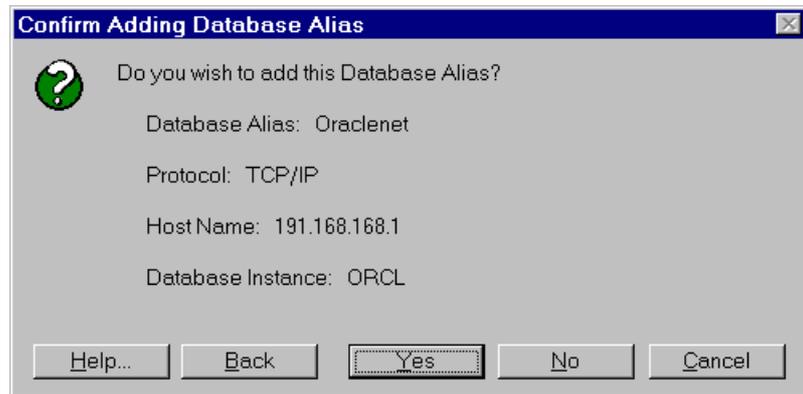
2. Atribuir um nome para o Alias(Database Alias) clicando em OK.



3. Indicar tipo de conexão (TCP/IP), clicando em OK.

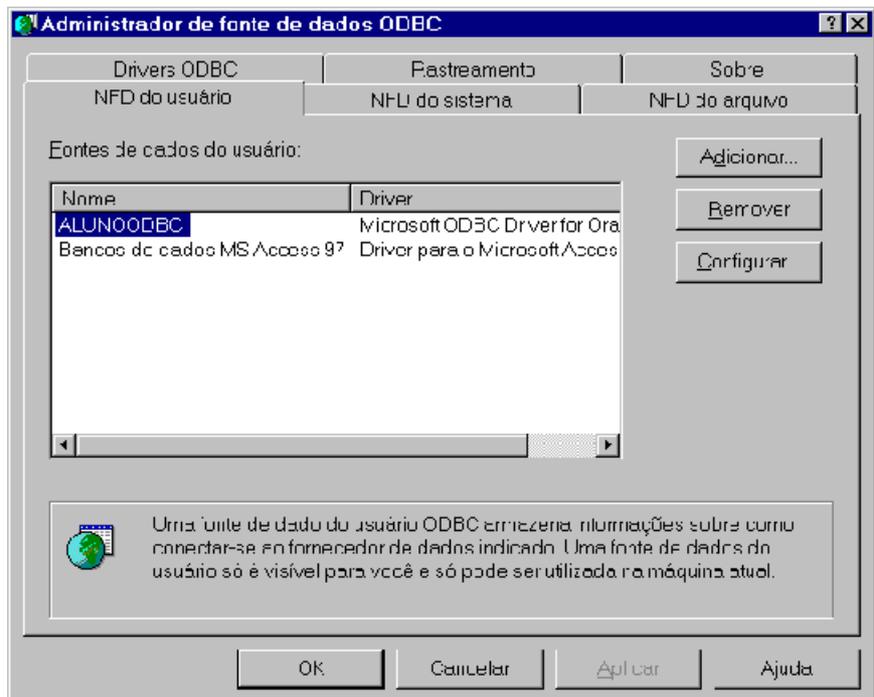


4. Indicar em TCP/IP Host Name, o nº do servidor, clicando em OK.



5. Ao final será apresentado um resumo dos parâmetros.

## Criar Conexão (Alias) no ODBC 32



6.

Acessar ODBC 32 e clicar em Add



7.

Indicar o Drive específico (indicado na imagem), clicando em Concluir.

8.

Description:		Cancel
User Name:	AlunoX	Help
Connect String:	2:	Options >>

8. Finalizando, indique Data Source Name, Description, User Name e Connect String. Indicando neste último parâmetro, 2:(para Banco Dados Local) ou o nome do Alias criado no SQLNet(AlunoX\_SQLNet) para uma conexão com o Servidor Oracle.
9. Criar um Objeto Projeto, anexar os objetos ao mesmo e exportá-lo.
10. No lado do servidor Importar o Projeto.
11. Recriar as Sequências.



## OUTROS SÍMBOLOS

( )	Separadores de lista	Ex.: AND MODELO IN('SANTA CECILIA','SÃO PAULO','SP')
;	Final de declaração	Ex.: COMMIT WORK;
.	Separador de item	Ex.: CLIENTES.CODIGO
'	Engloba uma string	Ex.: 'DIGIDATA'
:=	Atribuição	Ex.: NOME := 'DIGIDATA'
	Concatenação	Ex.: 'Codigo Cliente: '    CLIENTES.CODIGO
--	Comentário na mesma linha	Ex.: Begin -- Início da execução
/* e */	Delimitadores de comentários abrangendo várias linhas ( início e fim de comentário ).	

## VARIÁVEIS

- Seus nomes devem iniciar sempre com uma letra de ( A – Z )
- Podem ser seguidas de uma ou mais letras, números ou caracteres dos especiais \$, # ou \_
- Devem Ter no máximo 30 caracteres
- Não podem conter espaços em branco
- Não podem ser palavras reservadas do PL/SQL, como commit, if, raise, etc...

## TIPOS DE DADOS

<b>CHAR</b>	-	Tipos de dados alfanuméricos
<b>VARCHAR2</b>	-	Tipo alfanumérico com comprimento de – 32.768 a 32.767 bytes
<b>NUMBER</b>	-	Precisão de até 38 caracteres e ponto flutuante
<b>DATE</b>	-	Armazena valores de data de comprimento fixo
<b>BOOLEAN</b>	-	Contém Status TRUE e FALSE
<b>LONG</b>	-	Longo inteiro
<b>ROWID</b>	-	Identificador de linha

## COMPONENTES DA LINGUAGEM

### ESTRUTURAS DE CONTROLE

#### IF

```
If Var1 > 10 then
    Var2 = Var1 + 20 ;
End If ;
```

```
If Not ( Var1 <= 10 ) then
    Var2 = Var1 + 20 ;
End If ;
```

```
If Var1 > 10 then  
    If Var2 < Var1 then  
        Var2 = Var1 + 20 ;  
    End If ;  
End If ;
```

```
If Var1 > 10 then  
    Var2 = Var1 + 20 ;  
Else  
    Var2 = Var1 * Var1 ;  
End If ;
```

```
If Var1 > 10 then  
    Var2 = Var1 + 20 ;  
Else  
    If Var1 between 7 and 8 then  
        Var2 = 2 * Var1 ;  
    Else  
        Var2 = Var1 * Var1 ;  
    End If ;  
End If ;
```

```
If Var1 > 10 then  
    Var2 = Var1 + 20 ;  
Elseif Var1 between 7 and 8 then  
    Var2 = 2 * Var1 ;  
Else  
    Var2 = Var1 * Var1 ;  
End If ;
```

## LOOP

```
Contador := 1 ;
```

```
Loop  
    Contador := Contador + 1 ;  
    If Contador > 100 then  
        Exit ;  
    End If ;  
End Loop ;
```

```
Contador := 1 ;
```

```
Loop  
    Contador := Contador + 1 ;  
    Exit when Contador > 100 ;  
End Loop ;
```

```
Contador := 1 ;
```

```
While Contador < = 100 Loop  
    Contador := Contador + 1 ;  
End Loop ;
```

**FOR**

```
For Contador in 1 . . 3 Loop  
    Insert into tabela values ( 'Ainda em Loop ', Contador ) ;  
End Loop ;
```

```
If Contador > = 90 then  
    Null ; -- Construção conhecida como “Nula” ou “Ausência de Valor”  
    Else  
        Insert into tabela values ( 'Ainda em Loop ', Contador ) ;  
End If ;
```

## GERANDO VALORES DE CHAVES PRIMÁRIAS

O Oracle pode gerar automaticamente um novo valor para uma chave primária de um campo com SEQUÊNCIA ( **Sequence** ). Um Sequence é um objeto Oracle com estrutura de dados independente.

Ou seja, ele constrói o próximo valor da chave primária referente a uma seqüência ( Sequence ), através do comando **INSERT**, acompanhado das expressões **NEXTVAL** e **CURRVAL**.

Seqüência . **NEXTVAL** – Gera o próximo valor disponível dentro da seqüência.

Seqüência . **CURRVAL** – Faz referência ao mais recente valor gerado na seqüência dentro da mesma sessão.

Nota: Antes de gerar a chave primária verifique se o objeto foi criado como Sequence. Lembre-se também que para usar CURRVAL, é necessário Ter usado antes o NEXTVAL.

### Exemplo:

Para criar o Departamento Treinamento automaticamente nas regiões 1 e 2 o procedimento abaixo exemplifica:

```
SQL> INSERT INTO DEPTO ( COD, NOME, COD_REGIAO )  
      VALUES ( COD.NEXTVAL, 'TREINAMENTO', 1 ) ;
```

```
SQL> INSERT INTO S_DEPT ( ID, NAME, REGION_ID)  
      VALUES ( COD.NEXTVAL, 'TREINAMENTO', 2 ) ;
```

## COMANDOS ÚTEIS

```
SQL > DESCRIBE USER_OBJECTS ;
```

Name	Null?	Type
OBJECT_NAME	---	VARCHAR2(28)
OBJECT_ID		NUMBER
OBJECT_TYPE		VARCHAR2(13)
CREATED		DATE
LAST_DDL_TIME		DATE
TIMESTAMP		VARCHAR2(75)
STATUS		VARCHAR2(7)

Apresentar os nomes das tabelas para um determinado "OWNER".

```
SQL > SELECT Object_Name FROM User_Objects WHERE Object_Type = 'TABLE'
```

Digite a seguinte linha :

```
SQL > SELECT * FROM User_Objects WHERE Object_Type = 'SEQUENCE';
```

## ESTRUTURA DE UM BLOCO PL/SQL.

DECLARE

.

.

.

BEGIN

.

.

.

EXCEPTION

.

.

.

END;

### SEÇÃO

### DESCRIÇÃO

**Declare**

Contém a declaração de todas as variáveis, constantes, cursores, etc.

**Begin**

Contém os comandos PL/SQL

**Exception**

Contém as ações que serão tomadas se ocorrer algum erro na execução.

**Ex.:**

Create or replace procedure PROC (parametro1 in varchar2, parametro2 in varchar2 ) as

**Begin**

**Declare**

Varnum1 number ;

Varnum2 number ;

Data\_do\_Dia date := sysdate ;

Varalf1 varchar2(10) ;

Varalf2 varchar2(1) ;

Cursor MEU\_CURSOR is

Select \* From Dept where campo1 = parametro1 and campo2 = parametro2 ;

**Begin**

/\* Comandos

Diversos \*/

**End ;**

**Excption**

/\*Comandos para tratamento do erro\*/

**End ;**

## CURSORES

Cursores são trechos alocados de memória destinados a processar as declarações **SELECT**. Podem ser definidos pelo próprio PL/SQL, chamados de **Cursores Implícitos**, ou podem ser definidos manualmente, são os chamados de **Cursores Explícitos**.

### Cursores Explícitos

Esta técnica exige que o cursor seja declarado manualmente na seção **DECLARE**, e deve conter somente declarações **SELECT**. Para a sua utilização os seguintes passos devem ser observados:

- Declarar o Cursor
- Abrir o Cursor ( Preparar para sua utilização ) – **Open**
- Buscar os dados – **Fetch**
- Liberar memória após o uso ( Fechar o Cursor ) – **Close**

**Nota:** O Cursor é sempre definido na Seção **DECLARE** e é aberto após o **BEGIN**. O número de variáveis que recebem dados na declaração **FETCH** deve ser igual ao número de campos definidos na declaração **SELECT**, quando da declaração do cursor. O sucesso ou insucesso do cursor é determinado pelo teste **%FOUND%** ou **%NOTFOUND%**. Este teste deve ser feito antes que o cursor seja fechado.

ATRIBUTO	DESCRIÇÃO
<b>%ISOPEN</b>	Atributo do tipo Boolean, (True/False), sendo True quando o cursor está aberto
<b>%NOTFOUND</b>	Atributo do tipo Boolean, (True/False), sendo True quando o último FETCH não retornou linha.
<b>%FOUND</b>	Atributo do tipo Boolean, (True/False), sendo True quando o último FETCH retorna linha.
<b>%ROWCOUNT</b>	Atributo numérico, que retorna o número da linha retornada.

### CURSORES IMPLÍCITOS

Manipulado pelo próprio PL/SQL, sem a declaração na seção **DECLARE**. Caracterizado pelo uso da declaração **SELECT** a qual o PL/SQL manipula a definição do cursor implicitamente e os seguintes aspectos devem ser levados em consideração:

- Deve haver um **INTO** com cada cursor implícito.
- Assim como os cursores explícitos, as variáveis recebem dados com a palavra chave **INTO**
- Os cursores implícitos esperam que apenas uma linha seja retornada. Você deve examinar algumas exceções mais comuns, discutidas na tabela abaixo.

<b>NO_DATA_FOUND</b>	Nenhuma linha satisfaz os critérios de seleção
<b>TOO_MANY_ROWS</b>	Detecta a existência de mais de uma linha.
<b>DUP_VAL_ON_INDEX</b>	Detecta uma tentativa de criar uma entrada em um índice cujos valores de coluna chave já existem
<b>VALUE_ERROR</b>	O campo de destino não é suficientemente grande para conter o valor que está sendo colocado.

<b>ATRIBUTO</b>	<b>DESCRIÇÃO</b>
-----------------	------------------

<b>SQL%ISOPEN</b>	Atributo do tipo Boolean sendo sempre FALSE
<b>SQL%NOTFOUND</b>	Atributo do tipo Boolean, sendo TRUE quando o último comando não atualizou nenhuma linha.
<b>SQL%FOUND</b>	Atributo do tipo Boolean, sendo TRUE quando o último comando atualizou alguma linha.
<b>SQL%ROWCOUNT</b>	Atributo numérico, que retorna o número de linhas atualizadas.

## PACKAGES

Package é um objeto do banco de dados capaz de armazenar **procedures** e **functions** integradas, que podem ser executadas separadamente como se fossem parte de uma biblioteca ou à partir de uma execução provocar várias execuções encadeadas. A package é dividida em duas partes: a parte pública e a parte privada.

### Construção Pública

**Descrição** Pode ser referenciada de qualquer ambiente no Oracle.

**Localização** Declarar dentro da package specification e definir dentro da package body

### Construção Privada

**Descrição** Pode ser referenciada somente dentro da mesma package

**Localização** Declarar e definir dentro da package body

- Digitar o código fonte através de um editor de texto, criando-se primeiro a package specification e depois a package body.
- Executar o **Script** a partir do **SQL\*PLUS**, através do comando **Start / @**.
- Utilizar o comando **execute** para executar a Procedure / Function de dentro da package.

```
Create or Replace PACKAGE ATUALIZA_SALARIO is
  G_Salario number := 2500 ;
  Procedure RECEBE_DEPT ( Dept in number, sal number );
End ;
/
```

```

Create or Replace PACKAGE BODY ATUALIZA_SALARIO is
PROCEDURE RECEBE_DEPT ( pdept in number, sal number ) IS
  Salario_valido boolean ;
  Begin
    Salario_valido := ver_salario( pdept, sal ) ;

    If Salario_valido = TRUE then
      Update FUNCIONARIO set SALARIO = SALARIO * ( 1 + (sal/salario))
      Where coddept = pdept ;
    End If ;
  End;
End;

```

Function Ver\_Salario( Deptno in number, vsal number ) return boolean is

```

Begin
  SELECT MAX(SALARIO) INTO G_Salario FROM Funcionario
  Where coddept = deptno ;
  If G_Salario > vsal then
    Return ( TRUE ) ;
  Else
    Return ( FALSE ) ;
  End If ;
End ; /

```

## TRIGGER

**TRIGGER(GATILHO)** – Procedure executada ( disparada ) conforme o acontecimento o acontecimento de um evento, conforme a lista abaixo relacionada.

<b>BEFORE INSERT</b>	Dispara uma vez antes de uma transação INSERT
<b>BEFORE INSERT FOR EACH ROW</b>	Dispara antes de cada novo registro criado.
<b>AFTER INSERT</b>	Dispara uma vez depois de uma transação INSERT
<b>AFTER INSERT FOR EACH ROW</b>	Dispara depois de cada novo registro criado.
<b>BEFORE UPDATE</b>	Dispara uma vez antes de uma transação UPDATE
<b>BEFORE UPDATE FOR EACH ROW</b>	Dispara antes de cada novo registro alterado.
<b>AFTER UPDATE</b>	Dispara uma vez depois de uma transação UPDATE
<b>AFTER UPDATE FOR EACH ROW</b>	Dispara depois de cada novo registro alterado.
<b>BEFORE DELETE</b>	Dispara uma vez antes de uma transação DELETE
<b>BEFORE DELETE FOR EACH ROW</b>	Dispara antes de cada novo registro apagado.
<b>AFTER DELETE</b>	Dispara uma vez depois de uma transação DELETE
<b>AFTER DELETE FOR EACH ROW</b>	Dispara depois de cada novo registro apagado.

## SET SERVEROUTPUTON

Este comando habilita o Package **DBMS\_OUTPUT**, que através da Procedure **PUT\_LINE**, enviar as mensagens após uma Transação efetuada com sucesso. Este Package, bem como as Procedures, são do próprio Oracle, criados quando da sua instalação.

# Linguagem SQL

## Instrução SELECT

Instrui o programa principal do banco de dados para retornar a informação como um conjunto de registros.

### Sintaxe

```
SELECT      [predicado { * | tabela.* | [tabela.]campo1 [AS alias1]
            [, [tabela.]campo2 [AS alias2] [, ...]]}
FROM        expressãotabela [, ...] [IN bancodedadosexterno]
[WHERE... ]
[GROUP BY... ]
[HAVING... ]
[ORDER BY... ]
```

A instrução SELECT tem as partes abaixo:

Parte	Descrição
predicado	Um dos seguintes predicados: ALL, DISTINCT, DISTINCTROW ou TOP. Você usa o predicado para restringir o número de registros que retornam. Se nenhum for especificado, o padrão será ALL.
*	Especifica que todos os campos da tabela ou tabelas especificadas são selecionados.
tabela	O nome da tabela que contém os campos dos quais os registros são selecionados.
campo1, campo2 você incluir	Os nomes dos campos dos quais os dados serão recuperados. Se mais de um campo, eles serão recuperados na ordem listada.
alias1, alias2	Os nomes que serão usados como títulos de colunas em vez dos nomes originais das colunas na tabela.
expressãotabela recuperar.	O nome da tabela ou tabelas contendo os dados que você quer recuperar.
bancodedadosexterno expressãotabela se não	O Nome do banco de dados que contém as tabelas em expressãotabela se não estiver no banco de dados atual.

### Comentários

Para executar esta operação, o programa principal de banco de dados procura a tabela ou tabelas especificadas, extrai as colunas escolhidas, seleciona as linhas que satisfazem o critério e classifica ou agrupa as linhas resultantes na ordem especificada.

A instrução SELECT **não muda** os dados no banco de dados.

SELECT é normalmente a primeira palavra em uma instrução SQL. A maior parte das instruções SQL são instruções SELECT.

A sintaxe mínima da instrução SELECT é:

```
SELECT campos FROM tabela
```

Você pode usar um asterisco (\*) para selecionar todos os campos na tabela. O exemplo abaixo seleciona todos os campos na tabela Funcionários:

```
SELECT * FROM Funcionários;
```

Se o nome de um campo estiver incluído em mais de uma tabela na cláusula FROM, preceda-o com o nome da tabela e o operador . (ponto). No exemplo abaixo, o campo Departamento está nas tabelas Funcionários e Supervisores. A instrução SQL seleciona Departamento da tabela Funcionários e NomeSupv da tabela Supervisores:

```
SELECT Funcionários.Departamento, Supervisores.NomeSupv  
FROM Funcionários , Supervisores  
WHERE Funcionários.Departamento = Supervisores.Departamento;
```

Ao criar um objeto Recordset, o programa principal de banco de dados do Jet usa o nome do campo da tabela como o nome do objeto Field no objeto Recordset. Se você quiser um nome de campo diferente ou um nome que não esteja implícito na expressão usada para gerar o campo, use a palavra reservada AS. O exemplo abaixo usa o título Nasc para nomear o objeto Field retornado no objeto Recordset resultante:

```
SELECT DataNasc AS Nasc FROM Funcionários;
```

Sempre que você usar funções aggregate ou consultas que retornem nomes de objetos Field ambíguos ou duplicados, você precisará usar a cláusula AS para fornecer um nome alternativo para o objeto Field. O exemplo abaixo usa o título Contagem para nomear o objeto Field retornado no objeto Recordset resultante:

```
SELECT COUNT(FuncionárioID) AS Contagem FROM Funcionários;
```

Você pode usar outras cláusulas na instrução SELECT para restringir e organizar posteriormente os seus dados retornados.

## **Cláusula GROUP BY**

GROUP BY é opcional. Valores de resumo são omitidos se não houver qualquer função aggregate SQL na instrução SELECT. Os valores Null nos campos GROUP BY são agrupados e não omitidos. No entanto, os valores Null não são avaliados em qualquer função aggregate SQL. Use a cláusula WHERE para excluir linhas que você não quer agrupadas e use a cláusula HAVING para filtrar os registros após eles terem sido agrupados.

A não ser que contenha dados Memo ou OLE Object, um campo na lista de campos GROUP BY pode fazer referência a qualquer campo em qualquer tabela listada na cláusula FROM. Mesmo que o campo não esteja incluído na instrução SELECT, fornecida a instrução SELECT, inclua pelo menos uma função SQL. O programa principal de banco de dados do Jet não pode agrupar campos Memo ou OLE Objects.

Todos os campos na lista de campos SELECT devem ser incluídos na cláusula GROUP BY ou incluídos como argumentos em uma função aggregate SQL.

## Cláusula HAVING

HAVING é opcional. HAVING é semelhante a WHERE, que determina quais registros são selecionados. Depois que os registros são agrupados com GROUP BY, HAVING determina quais registros são exibidos:

```
SELECT CategoriaID, Sum(UnidadesNoEstoque) FROM Produtos
GROUP BY CategoriaID
HAVING Sum(UnidadesNoEstoque) > 100 AND LIKE "BOS*";
```

Uma cláusula HAVING pode conter até 40 expressões vinculadas por operadores lógicos, como And ou Or.

## Cláusula ORDER BY

ORDER BY é opcional. Entretanto, se você quiser exibir seus dados na ordem classificada, você deve utilizar ORDER BY. O padrão ordem de classificação é ascendente (A a Z, 0 a 9). Os dois exemplos abaixo classificam os nomes dos funcionários pelo sobrenome.

```
SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome;
```

```
SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome ASC;
```

Para classificar em ordem descendente (Z a A, 9 a 0), adicione a palavra reservada DESC ao final de cada campo que você quiser classificar em ordem descendente. O exemplo abaixo seleciona salários e os classifica em ordem descendente

```
SELECT Sobrenome, Salário FROM Funcionários ORDER BY Salário DESC, Sobrenome;
```

Se você especificar um campo que contém dados Memo ou OLE Objects na cláusula ORDER BY, um erro ocorrerá. O programa principal de banco de dados do Jet não classifica campos deste tipo. ORDER BY é normalmente o último item em uma instrução SQL.

Você pode incluir campos adicionais na cláusula ORDER BY. Os registros são classificados primeiro pelo primeiro campo listado depois de ORDER BY. Os registros que tiverem valores iguais naquele campo são classificados pelo valor no segundo campo listado e assim por diante.

## Exemplo da instrução SELECT, cláusula FROM

Esse exemplo seleciona os campos "Sobrenome" e "Nome" de todos os registros da tabela "Funcionários".

```
SELECT Sobrenome, Nome FROM Funcionários
```

Esse exemplo seleciona todos os campos da tabela "Funcionários".

```
SELECT Funcionários.* FROM Funcionários;
```

e Esse exemplo conta o número de registros que têm uma entrada no campo "CódigoPostal" nomeia o campo retornado como "Tcp".

```
SELECT Count(CódigoPostal) AS Tcp FROM Clientes;
```

Esse exemplo mostra qual seria o salário se cada funcionário recebesse um aumento de 10 por cento. Não altera o valor original dos salários.

```
SELECT Sobrenome, Salário AS Atual, Salário * 1.1 AS Proposto FROM Funcionários;
```

exibido Esse exemplo coloca o título Nome no topo da coluna "Sobrenome". O título Salário é no topo da coluna "Salário".

```
SELECT Sobrenome AS Nome, Salário FROM Funcionários;
```

Esse exemplo mostra o número de funcionários e os salários médio e máximo.

```
SELECT Count(*) AS [Total de Funcionários], Avg(Salário)
AS [Salário Médio], Max(Salário) AS [Salário Máximo]
FROM Funcionários;
```

de Para cada registro, mostra Sobrenome e Salário no primeiro e último campos. A seqüência de caracteres "tem um salário de" é retornada como o campo do meio de cada registro.

```
SELECT Sobrenome, 'tem um salário de', Salário FROM Funcionários;
```

### **Exemplo de cláusula GROUP BY**

funcionários Esse exemplo cria uma lista de nomes de departamentos únicos e o número de em cada um destes departamentos.

```
SELECT Departamento, Count([Departamento]) AS Tbc FROM Funcionários
GROUP BY Departamento;
```

Para cada título de função único, calcula o número de funcionários do departamento de Vendas que têm este título.

```
SELECT Título, Count(Título) AS Tbc FROM Funcionários
WHERE Departamento = 'Vendas' GROUP BY Título;
```

cor Esse exemplo calcula o número de itens em estoque para cada combinação de número e do item.

```
SELECT Item, Sum(Unidades) AS Tbc FROM ItensEmEstoque
GROUP BY Item, Cor;
```

## Exemplo de cláusula HAVING

Esse exemplo seleciona os títulos de cargos do departamento de Produção atribuídos a mais de 50 funcionários.

```
SELECT Título, Count(Título)
FROM Funcionários
WHERE Departamento = 'Produção'
GROUP BY Título HAVING Count(Título) > 50;
```

Esse exemplo seleciona os departamentos que tenham mais de 100 funcionários.

```
SELECT Departamento, Count([Departamento])
FROM Funcionários
GROUP BY Departamento HAVING Count(Departamento) > 100;
```

## Exemplo de cláusula ORDER BY

As instruções SQL mostradas abaixo usam a cláusula ORDER BY para classificar os registros em ordem alfabética e depois por categoria.

Esse exemplo ordena os registros pelo sobrenome, em ordem descendente (Z-A).

```
SELECT Sobrenome, Nome FROM Funcionários ORDER BY Sobrenome DESC;
```

Esse exemplo ordena, primeiro, por categoria ID e depois por nome do produto.

```
SELECT CategoricalID, ProdutoNome, PreçoUnit FROM Produtos
ORDER BY CategoricalID, NomeProduto;
```

## Instrução INSERT INTO

Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.

### Sintaxe

```
INSERT INTO destino [IN bancodedadosexterno] [(campo1[, campo2[, ...]])]  
SELECT [origem.]campo1[, campo2[, ...]]  
FROM expressãodetabela
```

Consulta anexação de um único registro:

```
INSERT INTO destino [(campo1[, campo2[, ...]])]  
VALUES (valor1[, valor2[, ...]])
```

A instrução INSERT INTO tem as partes abaixo:

Parte	Descrição
<b>destino</b>	O nome da tabela ou consulta em que os registros devem ser anexados.
<b>bancodedadosexterno</b>	O caminho para um banco de dados externo. Para uma descrição do caminho, consulte a cláusula IN.
<b>origem</b>	O nome da tabela ou consulta de onde os dados devem ser copiados.
<b>campo1, campo2</b>	Os nomes dos campos aos quais os dados devem ser anexados, se estiverem após um argumento destino ou os nomes dos campos dos quais se deve obter os dados, se estiverem após um argumento origem.
<b>expressãodetabela</b>	O nome da tabela ou tabelas das quais registros são inseridos. Este argumento pode ser um único nome de tabela ou uma combinação resultante de uma operação INNER JOIN, LEFT JOIN ou RIGHT JOIN ou de uma consulta gravada.
<b>valor1, valor2</b>	Os valores para inserir em campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante. Você deve separar os valores com uma vírgula e colocar os campos de textos entre aspas (" ").

### Comentários

Você pode usar a instrução INSERT INTO para adicionar um único registro a uma tabela usando a sintaxe de consulta anexação de um único registro como mostrado acima. Neste caso, seu código especifica o nome e o valor de cada campo do registro. Você precisa especificar cada um dos campos do registro para os quais um valor deve ser designado e um valor para este campo. Quando você não especifica cada campo, o valor padrão ou Null é inserido nas colunas omitidas. Os registros são adicionados no final da tabela.

Você também pode usar INSERT INTO para anexar um conjunto de registros de outra tabela ou consulta usando a cláusula SELECT ... FROM como é mostrado acima na sintaxe consulta anexação de vários registros. Neste caso, a cláusula SELECT especifica os campos para acrescentar à tabela destino especificada.

A tabela de origem ou de destino pode especificar uma tabela ou uma consulta. Se uma consulta for especificada, o programa principal de banco de dados do Microsoft anexa a qualquer e a todas as tabelas especificadas pela consulta.

INSERT INTO é opcional, mas quando incluída, precede a instrução SELECT.

Se sua tabela de destino contém uma chave primária, você deve acrescentar valores únicos, não Null ao campo ou campos da chave primária. Caso contrário, o programa principal de banco de dados do Jet não anexará os registros.

Se você anexar registros a uma tabela com um campo Counter e quiser numerar novamente os registros anexados, não inclua o campo Counter em sua consulta. Inclua o campo Counter na consulta se quiser manter os valores originais do campo.

Use a cláusula IN para anexar registros a uma tabela de outro banco de dados. Para achar quais registros serão anexados, antes de você executar a consulta anexação, primeiro execute e veja os resultados de uma consulta seleção que use o mesmo critério de seleção.

Uma operação de consulta anexação copia os registros de uma ou mais tabelas em outra. As tabelas que contêm os registros que você anexa não são afetadas pela operação de consulta anexação.

Em lugar de acrescentar registros existentes de outra tabela, você pode especificar o valor de cada campo em um único registro novo usando a cláusula VALUES. Se você omitir a lista de campo, a cláusula VALUES deve incluir um valor para cada campo na tabela; caso contrário, um erro ocorrerá em INSERT. Use uma instrução adicional INSERT INTO com uma cláusula VALUES para cada registro adicional que você quiser criar.

### Exemplo de instrução INSERT INTO

Esse exemplo seleciona todos os registros de uma tabela hipotética "Novos Clientes" e os adiciona à tabela "Clientes" (quando não são designadas colunas individuais, os nomes das colunas das tabelas SELECT devem corresponder exatamente aos da tabela INSERT INTO).

```
INSERT INTO Clientes SELECT [Novos Clientes].*  
FROM [Novos Clientes];
```

Esse exemplo cria um novo registro na tabela "Funcionários"

```
INSERT INTO Funcionários (Nome,Sobrenome, Título)  
VALUES ("André", "Pereira", "Estagiário");
```

Esse exemplo seleciona todos os estagiários de uma tabela hipotética "Estagiários" que foram contratados há mais de 30 dias e adiciona seus registros à tabela "Funcionários".

```
INSERT INTO Funcionários SELECT Estagiários.*  
FROM Estagiários WHERE DataContrato < Now() - 30;
```

## Declaração UPDATE

Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.

### Sintaxe

```
UPDATE tabela  
SET valornovo  
WHERE critério;
```

A instrução UPDATE tem as partes abaixo:

Parte	Descrição
tabela	O nome da tabela cujos os dados você quer modificar.
valornovo	Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.
critério registros	Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizados.

### Comentários

UPDATE é especialmente útil quando você quer alterar muitos registros ou quando os registros que você quer alterar estão em várias tabelas. Você pode alterar vários campos ao mesmo tempo. O exemplo abaixo aumenta o Valor do Pedido em 10 por cento e o valor do Frete em 3 por cento para embarques do Reino Unido:

```
UPDATE Pedidos SET ValorPedido = ValorPedido * 1.1, Frete = Frete * 1.03  
WHERE PaísEmbarque = 'RU';
```

UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

### Exemplo de instrução UPDATE

Esse exemplo muda os valores no campo "RelatórioPara" para 5 para todos os registros de funcionários que atualmente têm valores de RelatórioPara de 2.

```
UPDATE Funcionários SET RelatórioPara = 5 WHERE RelatórioPara = 2;
```

Esse exemplo aumenta o "PreçoUnit" de todos os produtos não suspensos do fornecedor 8 em 10 por cento.

```
UPDATE Produtos SET PreçoUnit = PreçoUnit * 1.1  
WHERE FornecedorID = 8 AND Suspenso = No;
```

## Instrução DELETE

Cria uma consulta exclusão que remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE.

### Sintaxe

```
DELETE [tabela.*]  
FROM tabela  
WHERE critério
```

A instrução DELETE tem as partes abaixo:

Parte	Descrição
tabela.*	O nome opcional da tabela da qual os registros são excluídos.
tabela	O nome da tabela da qual os registros são excluídos.
critério	Uma expressão que determina qual registro deve ser excluído.

### Comentários

DELETE é especialmente útil quando você quer excluir muitos registros. Para eliminar uma tabela inteira do banco de dados, você pode usar o método Execute com uma instrução DROP.

Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa DELETE, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Você pode usar DELETE para remover registros de tabelas que estão em uma relação um por vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado "vários" da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta. Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

Uma consulta de exclusão exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para Null.

### Importante

Após remover os registros usando uma consulta exclusão, você não poderá desfazer a operação. Se quiser saber quais arquivos foram excluídos, primeiro examine os resultados de uma consulta seleção que use o mesmo critério e então, execute a consulta exclusão. Mantenha os backups de seus dados. Se você excluir os registros errados, poderá recuperá-los a partir dos seus backups.

## Exemplo de instrução DELETE

Esse exemplo exclui todos os registros de funcionários cujo título seja Estagiário. Quando a cláusula FROM inclui apenas uma tabela, não é necessário indicar o nome da tabela na instrução DELETE.

```
DELETE *FROM Funcionários WHERE Título = 'Estagiário';
```

## Subconsultas SQL

Uma subconsulta é uma instrução **SELECT** aninhada dentro de uma instrução **SELECT**, **INSERT**, **DELETE** ou **UPDATE** ou dentro de uma outra subconsulta.

### Sintaxe

Você pode usar três formas de sintaxe para criar uma subconsulta:

comparação [ANY | ALL | SOME] (instruçõesql)  
expressão [NOT] IN (instruçõesql)  
[NOT] EXISTS (instruçõesql)

Uma subconsulta tem as partes abaixo:

Parte	Descrição
comparação	Uma expressão e um operador de comparação que compara a expressão com o resultado da subconsulta.
expressão	Uma expressão para a qual o resultado definido da subconsulta é procurado.
instruçõesqlt outra	Uma instrução SELECT de acordo com as mesmas regras e formato de qualquer outra instrução SELECT. Ela deve estar entre parênteses.

### Comentários

Você pode usar uma subconsulta em vez de uma expressão na lista de campo de uma instrução SELECT ou em uma cláusula WHERE ou HAVING. Em uma subconsulta, você usa uma instrução SELECT para fornecer um conjunto de um ou mais valores específicos para avaliar as expressões das cláusulas WHERE ou HAVING.

Use o predicado ANY ou SOME, que são sinônimos, para recuperar registros na consulta principal que satisfaçam a comparação com quaisquer registros recuperados na subconsulta. O exemplo abaixo retorna todos os produtos cujo preço unitário é maior que o preço de qualquer produto vendido com um desconto de 25 por cento ou mais:

```
SELECT * FROM Produtos WHERE PreçoUnit > ANY  
(SELECT PreçoUnit FROM PedidoDetalhes WHERE Desconto >= .25);
```

Use o predicado ALL para recuperar apenas os registros na consulta principal que satisfaçam a comparação com todos os registros recuperados na subconsulta. Se você mudou ANY para ALL no exemplo acima, a consulta retornaria apenas os produtos cujo preço unitário fosse maior que o de todos os produtos vendidos com um desconto de 25 por cento ou mais. Isto é muito mais restritivo.

Use o predicado IN para recuperar apenas os registros na consulta principal para os quais alguns registros na subconsulta contêm um valor igual. O exemplo abaixo retorna todos os produtos com um desconto de 25 por cento ou mais:

```
SELECT * FROM Produtos WHERE ProdutoID IN
(SELECT ProdutoID FROM PedidoDetalhes WHERE Desconto >= .25);
```

De maneira contrária, você pode usar NOT IN para recuperar apenas os registros na consulta principal para os quais não existam registros com valores iguais na subconsulta. Utilize o predicado EXISTS (com a palavra reservada NOT opcionalmente) em comparações true/false para determinar se a subconsulta retorna algum registro.

Você também pode usar aliases de nomes de tabelas em uma subconsulta para fazer referência a tabelas listadas em uma cláusula FROM fora da subconsulta. O exemplo abaixo retorna os nomes dos funcionários cujos salários sejam iguais ou superiores à média de salários de todos os funcionários na mesma função. Para a tabela Funcionários é dada o alias "T1":

```
SELECT Sobrenome, Nome, Título, Salário FROM Funcionários AS T1
WHERE Salário >= (SELECT Avg(Salário)
FROM Funcionários WHERE T1. T1.Título = Funcionários.Título) Order by Title;
```

No exemplo acima, a palavra reservada AS é opcional. Algumas subconsultas são aceitas em consultas de tabela cruzada especialmente como predicados (as da cláusula WHERE). Subconsultas como saída (as da lista SELECT) não são aceitas em tabelas de referência cruzada.

## Exemplos de subconsultas SQL

Esse exemplo lista o nome, título e salário de todos os representantes de vendas cujos salários sejam superiores aos de todos os gerentes e diretores.

```
SELECT Sobrenome, Nome, Título, Salário FROM Funcionários
WHERE Título LIKE "**Repr Vendas*" AND Salário > ALL
(SELECT Salário FROM Funcionários WHERE (Título LIKE "**Gerente*"
OR (Título LIKE "**Diretor*")));
```

Esse exemplo lista o nome e preço unitário de todos os produtos cujo preço unitário seja igual ao do Licor de Cacau.

```
SELECT NomeProduto, PreçoUnit FROM Produtos
WHERE PreçoUnit = (SELECT PreçoUnit FROM [Produtos]
WHERE NomeProduto = "Licor de Cacau");
```

Esse exemplo lista a empresa e o contato de cada empresa de todos os clientes que fizeram pedidos no segundo trimestre de 1995.

```
SELECT NomeContato, NomeEmpresa, ContatoTítulo, Fone FROM Clientes
WHERE ClienteID IN (SELECT ClienteID FROM Pedidos
WHERE DataPedido BETWEEN '1/04/95' AND '1/07/95');
```

Esse exemplo lista os funcionários cujo salário seja maior que a média dos salários de todos os funcionários.

```
SELECT Sobrenome, Nome, Título, Salário FROM Funcionários T1
WHERE Salário >= (SELECT AVG(Salário) FROM Funcionários
WHERE Funcionários.Título = T1.Título) ORDER BY Título;
```

Esse exemplo seleciona o nome de todos os funcionários que tenham registrado pelo menos um pedido. Isto também poderia ser feito com INNER JOIN.

```
SELECT Nome, Sobrenome FROM Funcionários AS E
WHERE EXISTS (SELECT * FROM Pedidos AS O
WHERE O.FuncionárioID = E.FuncionárioID);
```

Altera o campo Efetuado do arquivo de serviços para 2 caso o mesmo tenha parecer técnico da entidade encaminhamento diferente de nulo.

```
UPDATE servico SET efetuado = 2
WHERE numero_servico = ANY (SELECT servico.numero_servico
FROM servico INNER JOIN encaminhamento
ON (servico.numero_servico = encaminhamento.numero_servico)
AND (servico.ano_servico = encaminhamento.ano_servico)
WHERE (((servico.efetuado) Is Null) AND ((encaminhamento.parecer_tecnico) Is Not Null))
GROUP BY servico.numero_servico ORDER BY servico.numero_servico);
```