BANCO DE DADOS

SQL

Prof. Marcos Alexandruk

SUMÁRIO

1. SQL: HISTÓRICO E NOÇÕES GERAIS

BREVE HISTÓRIA DA LINGUAGEM SQL

Pesquisadores da IBM desenvolveram, no início da década de 1970, a SEQUEL (Structured English Query Language), primeira linguagem de acesso a SGBDR (Sistemas Gerenciadores de Banco de Dados Relacionais). Alguns anos depois a linguagem passou a ser denominada simplesmente SQL (Structured Query Language).
Com o surgimento de um número cada vez maior de SGBDRs (Sistemas de Gerenciamento de Bancos de Dados Relacionais), fez-se necessário especificar um padrão para a linguagem de acesso.
Portanto, em 1986, surgiu o SQL-86, a primeira versão da linguagem SQL (Structured Query Language), em um trabalho conjunto da ISO (International Organization for Standardization) e da ANSI (American National Standards Institute).
A linguagem passou por aperfeiçoamentos e em 1992 foi lançada a SQL-92 ou SQL-2.
Em 1999, foi lançado um novo padrão chamado SQL-99 ou SQL-3. Este padrão permitiu a utilização de tipos de dados complexos (exemplo: BLOB - Binary Large Object) e incorporou características de orientação a objetos.
A mais nova versão do padrão SQL é a SQL:2003. Fez-se uma revisão do padrão SQL-3 e acrescentou-se uma nova parte que contempla o tratamento de XML.

PRINCIPAIS SISTEMAS DE BANCO DE DADOS QUE UTILIZAM SQL

- DB2
- Ingres
- MySQL
- Oracle Database
- PostgreSQL
- Microsoft SQL Server
- SQLite
- Sybase
- Informix
- Firebird

ORACLE: HISTÓRICO

1977 Larry Ellison, Bob Miner e Ed Oates fundam a SDL (Software Level Laboratories).
1978 O nome da empresa é mudado para RSI (Rational Software Inc.). O Oracle 1.0 é escrito em assembly (utilizando no máximo 128 KB de memória).
1979 A RSI lança o primeiro produto comercial de banco de dados relacional utilizando a linguagem SQL.
1980 O nome da RSI é mudado para Oracle System Corporation. É lançado o Oracle 2.0.
1983 Lançado o Oracle 3 (escrito em C), o primeiro a rodar em mainframes e em minicomputadores.
1984 Desenvolvida a versão para PC do Oracle. Lançada a versão 4 do Oracle.
1986 O Oracle 5 é lançado com capacidades distribuídas (SQL*Star). É possível reunir informações de bancos de dados localizados em sites (locais) diferentes.
1988 A Oracle muda sua sede para Redwood City.
1989 Lançado o Oracle 6.2.
1993 Lançado o Oracle 7 para UNIX.
1994 Lançado o Oracle 7 para PC.
1999 Lançado o Oracle 8i, o primeiro banco de dados com suporte nativo para XML.
2000 Lançado o Oracle 9i, primeiro banco de dados com Real Application Clusters (RAC).
2003 Lançado o Oracle 10g.
2007 Lançado o Oracle 11g.

NOÇÕES GERAIS DA LINGUAGEM SQL

Os comandos SQL estão divididos em categorias de acordo com sua funcionalidade:

DDL (DATA DEFINITION LANGUAGE)

Linguagem de Definição de Dados: usada para definir dados e objetos de um banco de dados.

COMANDO	FUNÇÃO
CREATE TABLE	Cria uma tabela
ALTER TABLE	Altera uma tabela
DROP TABLE	Elimina uma tabela
CREATE VIEW	Cria uma visão
ALTER VIEW	Altera uma visão
DROP VIEW	Elimina uma visão
CREATE INDEX	Cria um índice
ALTER INDEX	Altera um índice
DROP INDEX	Elimina um índice
TRUNCATE	Remove todas as linhas da tabela (não pode ser desfeito)

DML (DATA MANIPULATION LANGUAGE)

Linguagem de Manipulação de Dados: usada para manipular dados.

COMANDO	FUNÇÃO
INSERT	Insere as linhas na tabela
DELETE	Exclui as linhas da tabela
UPDATE	Altera os dados (conteúdos) da tabela

DQL (DATA QUERY LANGUAGE)

Linguagem de Consulta de Dados: usada para recuperar dados.

COMANDO	FUNÇÃO
SELECT	Seleciona (recupera) dados de uma tabela ou visão (view)

Observação: O SELECT também é considerado um comando DML.

DCL (DATA CONTROL LANGUAGE)

Linguagem de Controle de Dados: usada para conceder ou remover direitos de acesso aos usuários do banco de dados.

COMANDO	FUNÇÃO
GRANT	Concede privilégios de acesso para um usuário
REVOKE	Revoga privilégios de acesso para um usuário

DTL (DATA TRANSACTION LANGUAGE)

Linguagem de Transação de Dados: usada para controlar as transações do banco de dados.

COMANDO	FUNÇÃO	
BEGIN WORK	START TRANSACTION	Marca o começo de uma transação
COMMIT	Envia todos os dados das mudanças permanentemente	
ROLLBACK	Descarta as mudanças nos dados existentes desde o último COMMIT ou ROLLBACK	

NOMES DE TABELAS E COLUNAS

- Não utilizar caracteres especiais (exceto o underscore "_");
- Começar com uma letra e não com um número;
- Evitar acentuação e "ç";
- Não utilizar espaços.

PRINCIPAIS TIPOS DE DADOS

DATATYPE	DESCRIÇÃO
CHAR	Alfanumérico de tamanho fixo. Tamanho máximo 2.000 caracteres. Valor default: 1.
VARCHAR2	Alfanumérico de tamanho variável. Tamanho máximo 4.000 caracteres.
NUMBER	Valores numéricos. Ex: NUMBER (5,2) armazena -999,99 a +999,99. Máximo: NUMBER (38)
DATE	Armazena data e hora (inclusive minuto e segundo). De 01/01/4712 AC até 31/12/9999 DC
TIMESTAMP	Armazena data e hora (minuto, segundo e milésimo de segundo).
CLOB	Character Long Object. Tamanho máximo 4 GB.
BLOB	Binary Long Object. Tamanho máximo 4 GB.
BFILE	Binary File. Referência a um arquivo externo ao banco.

ALGUNS COMANDOS ÚTEIS

COMANDO	DESCRIÇÃO	
PASSWORD nome_usuario;	Altera senha do usuário	
PASSWORD	Altera senha do usuário atual	
SELECT * FROM ALL_USERS;	Lista todos os usuários	
SELECT TABLE_NAME FROM USER_TABLES;	Lista todas as tabelas	
SELECT * FROM CAT;	Lista todas as tabelas	
SHOW USER	Mostra usuário atual	
SELECT * FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'nome_da_tabela';	Apresenta informações sobre as constraints da tabela:	
	OWNER	Usuário que criou a constraint
	CONSTRAINT_NAME	Nome da constraint
	CONSTRAINT_TYPE	Tipo da constraint: P = Primary Key R = Foreign Key C = Check U = Unique
	TABLE NAME	Nome da Tabela
	SEARCH_CONDITION	Domínio permitido (CHECK)
	R_CONSTRAINT_NAME	Utilizada em constraints FK, indica a constraint PK na tabela referenciada
	STATUS	Indica se a constraint está habilitada

Observação: Alguns comandos requerem privilégio de administrador.

WEBSITES RECOMENTADOS

http://www.oracle.com/technology/documentation/index.html

http://www.ss64.com/ora/

2. CRIAÇÃO DE TABELAS E CONSTRAINTS

CRIANDO UMA TABELA

Para criar uma tabela utilize o comando CREATE TABLE:

```
CREATE TABLE NOME_DA_TABELA (
NOME_DA_COLUNA_1 TIPO_DE_DADO(TAMANHO),
NOME_DA_COLUNA_2 TIPO_DE_DADO(TAMANHO));
```

O exemplo a seguir apresenta a criação de uma tabela denominada **CLIENTE** com a seguinte estrutura:

NOME DA COLUNA	TIPO DE DADO	TAMANHO
CODCLI	NUMBER	4
NOME	VARCHAR2	50
UF	CHAR	2
CPF	CHAR	11

```
CREATE TABLE CLIENTE (
CODCLI NUMBER(4),
NOME VARCHAR2(50),
UF CHAR(2),
CPF CHAR(11));
```

VERIFICANDO A ESTRUTURA DE UMA TABELA

Para verificar a estrutura da tabela criada utilize o comando DESCRIBE ou a forma abreviada DESC:

```
DESCRIBE CLIENTE ou DESC CLIENTE
```

CONSTRAINTS

As restrições (constraints) estabelecem as relações entre as várias tabelas em um banco de dados e realizam três tarefas fundamentais:

- Mantêm a integridade dos dados;
- Não permitem a inclusão de valores "indesejáveis";
- Impedem a exclusão de dados se existirem dependências entre tabelas.

CHAVE PRIMÁRIA (PK – PRIMARY KEY)

Coluna ou grupo de colunas que permite identificar uma única linha (tupla) da tabela.

Para chave primária **simples** utilize:

```
CREATE TABLE NOME_DA_TABELA (
NOME_DA_COLUNA_1 TIPO_DE_DADO(TAMANHO),
NOME_DA_COLUNA_2 TIPO_DE_DADO(TAMANHO),
CONSTRAINT NOME_DA_CONSTRAINT PRIMARY_KEY(NOME_DA_COLUNA_1));
```

Para chave primária **composta** utilize:

```
CREATE TABLE NOME_DA_TABELA (
NOME_DA_COLUNA_1 TIPO_DE_DADO(TAMANHO),
NOME_DA_COLUNA_2 TIPO_DE_DADO(TAMANHO),
CONSTRAINT NOME_DA_CONSTRAINT PRIMARY_KEY(NOME_DA_COLUNA_1,NOME_DA_COLUNA_2);
```

O exemplo a seguir apresenta a criação **CLIENTE** com a coluna CODCLI como chave primária:

```
CREATE TABLE CLIENTE (
CODCLI NUMBER(4),
NOME VARCHAR2(50),
UF CHAR(2),
CPF CHAR(11),
CONSTRAINT CLIENTE_PK PRIMARY KEY(CODCLI));
```

CHAVE ESTRANGEIRA (FK – FOREIGN KEY)

Coluna que estabelece o relacionamento entre duas tabelas. Corresponde à chave primária da tabela-pai.

```
CREATE TABLE NOME_DA_TABELA_1 (
NOME_DA_COLUNA_1_1 TIPO_DE_DADO(TAMANHO),
NOME_DA_COLUNA_1_2 TIPO_DE_DADO(TAMANHO),
CONSTRAINT NOME_DA_CONSTRAINT PRIMARY_KEY(NOME_DA_COLUNA_1_1));

CREATE TABLE NOME_DA_TABELA_2 (
NOME_DA_COLUNA_2_1 TIPO_DE_DADO(TAMANHO),
NOME_DA_COLUNA_2_2 TIPO_DE_DADO(TAMANHO),
CONSTRAINT NOME_DA_CONSTRAINT PRIMARY_KEY(NOME_DA_COLUNA_2_1),
CONSTRAINT NOME_DA_CONSTRAINT FOREIGN_KEY(NOME_DA_COLUNA_2_2
REFERENCES NOME_DA_TABELA_1(NOME_DA_COLUNA_1_1));
```

O exemplo a seguir apresenta coluna CODCLI da tabela PEDIDO como chave estrangeira relacionando-a com a coluna CODCLI da tabela CLIENTE (anteriormente criada):

```
CREATE TABLE PEDIDO (
NRPED NUMBER(5),
DATA_EMISSAO DATE,
CODCLI NUMBER(4),
CONSTRAINT PEDIDO_PK PRIMARY KEY(NRPED),
CONSTRAINT PEDIDO_CLIENTE_FK FOREIGN KEY(CODCLI) REFERENCES CLIENTE(CODCLI));
```

Observe que o tipo de dado e o tamanho da coluna CODCLI da tabela PEDIDO deve ser o mesmo que o apresentado na tabela CLIENTE, isto é, NUMBER(4).

A chave estrangeira tem importância fundamental para garantir a INTEGRIDADE referencial dos dados. Portanto, nos exemplos apresentados, se o usuário tentar inserir na tabela PEDIDO um código de cliente (CODCLI) que não conste na tabela CLIENTE este será rejeitado pelo banco de dados. Por outro lado, se o usuário tentar excluir da tabela CLIENTE um código que conste na tabela PEDIDO, isto também não será possível. No entanto, as seguintes cláusulas podem ser acrescentadas à chave estrangeira alterando o seu funcionamento:

CLÁUSULA	FUNÇÃO
ON DELETE SET NULL	Quando for removido um valor da tabela-pai o banco define os valores correspondentes da tabela-filho como NULL.
ON DELETE CASCADE	Quando for removido um valor da tabela-pai o banco remove os valores correspondentes da tabela-filho.

Exemplo:

```
...
CONSTRAINT PEDIDO_CLIENTE_FK FOREIGN KEY(CODCLI)
REFERENCES CLIENTE(CODCLI) ON DELETE SET NULL);
```

NOT NULL

Indica que o conteúdo de uma coluna não poderá ser nulo (vazio).

```
CREATE TABLE CLIENTE (
CODCLI NUMBER(4),
NOME VARCHAR2(50) NOT NULL,
UF CHAR(2),
CPF CHAR(11),
CONSTRAINT CLIENTE_PK PRIMARY KEY(CODCLI));
```

UNIQUE

Indica que não poderá haver repetição no conteúdo da coluna.

```
CREATE TABLE CLIENTE (
CODCLI NUMBER(4),
NOME VARCHAR2(50),
UF CHAR(2),
CPF CHAR(11) UNIQUE,
CONSTRAINT CLIENTE_PK PRIMARY KEY(CODCLI));
```

ou:

```
CREATE TABLE CLIENTE (
CODCLI NUMBER(4),
NOME VARCHAR2(50),
UF CHAR(2),
CPF CHAR(11),
CONSTRAINT CLIENTE_PK PRIMARY KEY(CODCLI),
CONSTRAINT CLIENTE_UK UNIQUE(CPF));
```

Observe a seguir algumas diferenças entre as constraints PRIMARY KEY e UNIQUE:

PRIMARY KEY			UNIQUE	
Permite repetição de valores?	NÃO		Permite repetição de valores?	NÃO
Permite valores nulos (NULL)?	NÃO		Permite valores nulos (NULL)?	SIM

CHECK

Define um domínio de valores para o conteúdo da coluna.

```
CREATE TABLE TABELA_1 (
COLUNA_1 NUMBER(5),
COLUNA_2 CHAR(1),
CONSTRAINT TABELA_1_CK CHECK (COLUNA_2 = 'M' OR COLUNA_2 = 'F'));
```

```
CREATE TABLE TABELA_2 (
COLUNA_1 NUMBER(5),
COLUNA_2 CHAR(2),
CONSTRAINT TABELA_2_CK CHECK (COLUNA_2 IN ('SP','RJ','MG')));
```

```
CREATE TABLE TABELA_3 (
COLUNA_1 NUMBER(5),
COLUNA_2 NUMBER(5),
CONSTRAINT TABELA_3_CK CHECK (COLUNA_2 < 1000));
```

DESABILITANDO CONSTRAINTS

Para desabilitar uma constraint utilize:

```
ALTER TABLE NOME_DA_TABELA DISABLE CONSTRAINT NOME_DA_CONSTRAINT;
```

Exemplo:

```
ALTER TABLE TABELA_1 DISABLE CONSTRAINT TABELA_1_PK;
```

HABILITANDO CONSTRAINTS

Para habilitar uma constraint utilize:

```
ALTER TABLE NOME_DA_TABELA ENABLE CONSTRAINT NOME_DA_CONSTRAINT;
```

Exemplo:

```
ALTER TABLE TABELA_1 ENABLE CONSTRAINT TABELA_1_PK;
```

Observação: Desativar uma CONSTRAINT, inserir valores que violem a restrição de integridade e tentar ativar a restrição posteriormente com o comando ENABLE provocará um erro.

Até a versão atual do Oracle não há possibilidade de incluir a cláusula ON UPDATE CASCADE em uma restrição do tipo Foreign Key. Portanto, para alterar os valores em colunas Primary Key que contém referencias em tabelas-filho é preciso desativar a restrição Foreign Key na tabela-filho, alterar os valores na coluna Primary Key na tabela-pai, alterar (se necessário) os valores na coluna Foreign Key na tabela-filho e ativar novamente a restrição Foreign Key.

REMOVENDO CONSTRAINTS

Para remover uma constraint utilize:

```
ALTER TABLE NOME_DA_TABELA
DROP CONSTRAINT NOME_DA_CONSTRAINT;
```

Exemplo:

```
ALTER TABLE TABELA_1 DROP CONSTRAINT TABELA_1_PK;
```

ADICIONANDO CONSTRAINTS

Para adicionar uma constraint utilize:

```
ALTER TABLE NOME_DA_TABELA
ADD CONSTRAINT NOME_DA_CONSTRAINT TIPO_DA_CONSTRAINT(NOME_DA_COLUNA);
```

Exemplo:

```
ALTER TABLE TABELA_1 ADD CONSTRAINT TABELA_1_PK PRIMARY KEY(COLUNA_1);
```

3. EXCLUSÃO E ALTERAÇÃO DE TABELAS

EXCLUINDO UMA TABELA

Para excluir uma tabela utilize o comando DROP TABLE:

```
DROP TABLE NOME_DA_TABELA;
```

Para excluir uma tabela e suas respectivas constraints utilize o comando DROP TABLE com a opção CASCADE CONSTRAINTS:

```
DROP TABLE NOME_DA_TABELA CASCADE CONSTRAINTS;
```

A partir da versão 10g, quando uma tabela é "dropada", o Oracle não remove automaticamente o espaço dela da tablespace, a não ser que o parâmetro PURGE seja especificado no comando DROP. Os metadados da tabela, dos índices e das constraints associadas a ela são renomeados e enviados a uma tabela conhecida como RECYCLE BIN (lixeira). Desta forma, caso a tabela tenha sido excluída erroneamente é possível recuperá-la posteriormente através do comando FLASHBACK TABLE.

Para saber quais objetos estão na lixeira utilize:

```
SELECT * FROM RECYCLEBIN;
```

RECUPERANDO UMA TABELA DA LIXEIRA

Para recuperar uma tabela da lixeira utilize.

```
FLASHBACK TABLE NOME_DA_TABELA TO BEFORE DROP;
```

Exemplo:

```
CREATE TABLE TESTE (
CODIGO NUMBER(4));
DROP TABLE TESTE CASCADE CONSTRAINTS;
SELECT * FROM CAT;
FLASHBACK TABLE TESTE TO BEFORE DROP;
SELECT * FROM CAT;
```

A cláusula RENAME TO permite que a tabela receba um novo nome na recuperação. É uma cláusula opcional porém necessária caso exista uma tabela com o mesmo nome da original no esquema do usuário.

```
FLASHBACK TABLE NOME_DA_TABELA TO BEFORE DROP RENAME TO NOVO_NOME_DA_TABELA;
```

PURGE

Utilize o comando PURGE para eliminar da lixeira uma tabela que não seja mais necessária:

```
PURGE "NOME_DA_TABELA";
```

Para limpar toda a lixeira do usuário utilize:

```
PURGE RECYCLEBIN;
```

Para limpar a lixeira de todos os esquemas do banco de dados (de preferência após um backup completo) utilize o comando abaixo (conectado como sys as sysdba):

```
PURGE DBA_RECYCLEBIN;
```

Para a remoção física da tabela e dos objetos relacionados a ela sem transferi-los para a lixeira utilize:

```
DROP TABLE NOME_DA_TABELA CASCADE CONSTRAINTS PURGE;
```

ALTERANDO UMA TABELA

Para alterar as definições de uma tabela utilize o comando ALTER TABLE.

ADICIONANDO COLUNAS

Para adicionar uma coluna utilize:

```
ALTER TABLE NOME_DA_TABELA ADD NOME_DA_COLUNA TIPO_DE_DADO(TAMANHO);
```

Exemplo:

```
ALTER TABLE CLIENTE ADD E_MAIL VARCHAR2(40);
```

ALTERANDO A LARGURA DE UMA COLUNA

Para alterar a largura de uma coluna utilize:

```
ALTER TABLE NOME_DA_TABELA MODIFY NOME_DA_COLUNA TIPO_DE_DADO(NOVO_TAMANHO);
```

Exemplo:

```
ALTER TABLE CLIENTE MODIFY E_MAIL VARCHAR2(50);
```

ALTERANDO O NOME DE UMA COLUNA

Para alterar o nome de uma coluna utilize:

```
ALTER TABLE NOME_DA_TABELA
RENAME COLUMN NOME_ANTIGO_DA_COLUNA TO NOME_NOVO_DA_COLUNA;
```

Exemplo:

```
ALTER TABLE CLIENTE RENAME COLUMN E_MAIL TO EMAIL;
```

EXCLUINDO COLUNAS

Para excluir uma coluna de uma coluna utilize:

```
ALTER TABLE NOME_DA_TABELA DROP COLUMN NOME_DA_COLUNA;
```

Exemplo:

```
ALTER TABLE CLIENTE DROP COLUMN EMAIL;
```

Para excluir uma coluna com constraints utilize:

```
ALTER TABLE NOME_DA_TABELA DROP COLUMN NOME_DA_COLUNA CASCADE CONSTRAINTS;
```

RENOMEANDO UMA TABELA

Para alterar o nome de uma tabela utilize:

```
RENAME NOME_ANTIGO_DA_TABELA TO NOME_NOVO_DA_TABELA;
```

CRIANDO UMA TABELA COM BASE EM OUTRA

Para criar uma tabela com todas as linhas e colunas de outra utilize:

```
CREATE TABLE NOME_DA_NOVA_TABELA AS SELECT * FROM NOME_DA_TABELA_BASE;
```

Exemplo:

```
CREATE TABLE CLIENTE_2 AS SELECT * FROM CLIENTE;
```

Para criar uma tabela com todas as linhas e algumas colunas selecionadas de outra:

```
CREATE TABLE NOME_DA_NOVA_TABELA (NOME_DA_COLUNA_1,NOME_DA_COLUNA_2)
AS SELECT NOME_DA_COLUNA_1,NOME_DA_COLUNA_2 FROM NOME_DA_TABELA_BASE;
```

Exemplo:

```
CREATE TABLE CLIENTE_3 (CODCLI,NOME) AS SELECT CODCLI,NOME FROM CLIENTE;
```

TRUNCATE

Comando DDL utilizado para 'cortar' uma tabela. Todas as linhas da tabela são eliminadas.

Para 'cortar' uma tabela utilize:

```
TRUNCATE TABLE NOME_DA_TABELA;
```

Exemplo:

```
TRUNCATE TABLE CLIENTE;
```

Algumas considerações importantes sobre o comando TRUNCATE:

- Mais rápido do que executar um DELETE sem a cláusula WHERE.
- Os dados não poderão ser recuperados (a não ser através do backup).

4. INSERT, UPDATE E DELETE

INSERT

Para incluir dados em uma tabela utilize o comando INSERT:

```
INSERT INTO NOME_DA_TABELA (NOME_DA_COLUNA_1,NOME_DA_COLUNA_2...)
VALUES (VALOR_DA_COLUNA_1, VALOR_DA_COLUNA_2...);
```

Exemplo:

```
INSERT INTO CLIENTE (CODCLI,NOME,UF,CPF)
VALUES (1001,'Antonio Alves','SP''11122233344',);
```

Observação: Valores cujo tipo de dado é NUMBER não precisam ser envolvidos por ' (aspas simples). Outros valores com tipos de dados VARCHAR2, CHAR e DATE **devem** ser envolvidos por ' (aspas simpes).

UPDATE

Para modificar (atualizar) os dados inseridos em uma tabela utilize o comando UPDATE:

```
UPDATE NOME_DA_TABELA
SET NOME_DA_COLUNA_1 = NOVO_VALOR_1, NOME_DA_COLUNA_2 = NOVO_VALOR_2
WHERE ...;
```

Exemplo (atualizando um valor: NOME):

```
UPDATE CLIENTE
SET NOME = 'Antonio Alvares'
WHERE CODCLI = 1001;
```

Exemplo (atualizando dois valores: NOME e UF):

```
UPDATE CLIENTE
SET NOME = 'Antonio Alvares', UF = 'RJ'
WHERE CODCLI = 1001;
```

Observação: Caso não seja utilizada a cláusula WHERE, indicando a linha em que a alteração deve ser realizada, TODAS as linhas da tabela serão alteradas. No exemplo acima, a não utilização da cláusula WHERE provocaria a alteração dos nomes de TODOS os clientes para **Antonio Alvares**.

DELETE

Para excluir linhas de uma tabela utilize o comando DELETE:

```
DELETE NOME_DA_TABELA          ou          DELETE FROM NOME_DA_TABELA
WHERE ...;                                 WHERE ...;
```

Exemplo:

```
DELETE FROM CLIENTE
WHERE CODCLI = 1001;
```

Observação: Caso não seja utilizada a cláusula WHERE, indicando a linha em que a exclusão deve ser realizada, TODAS as linhas da tabela serão excluídas.

5. TRANSAÇÕES

Uma transação representa um conjunto que comandos cujo resultado deverá ser gravado de uma vez no banco de dados.

Uma transação pode terminar das seguintes maneiras:

COMMIT

Grava definitivamente no banco os efeitos dos comandos da transação (INSERT, UPDATE, DELETE).

Observações:

- Encerramento normal da seção: Provoca um COMMIT implícito;
- Utilização de comandos DDL ou DCL: Provoca um COMMIT implícito.

ROLLBACK

Descarta os efeitos dos comandos da transação.

Observação:

- Encerramento anormal da seção: Provoca um ROLLBACK implícito;

AUTOCOMMIT

Confirma imediatamente cada instrução DML (INSERT, UPDATE e DELETE). Neste caso, a instrução ROLLBACK não terá nenhum efeito.

Para **ativar** o AUTOCOMMIT no SQL*Plus utilize:

```
SET AUTOCOMMIT ON
```

Para **desativar** o AUTOCOMMIT no SQL*Plus utilize:

```
SET AUTOCOMMIT OFF
```

6. CLÁUSULA WHERE

Permite que sejam especificadas linhas sobre as quais será aplicada determinada instrução. É sempre seguida por uma expressão lógica que pode conter os seguintes operadores:

DE COMPARAÇÃO	>, <, =, >=, <=, <>
LÓGICOS	AND, OR, NOT
DA LINGUAGEM SQL	IS NULL, IS NOT NULL, LIKE, NOT LIKE, IN, NOT IN, BETWEEN, NOT BETWEEN, EXISTS, NOT EXISTS

OPERADORES SQL		
IS NULL	Verifica se o conteúdo da coluna é NULL (vazio)	
IS NOT NULL	Negação do operador IS NULL	
LIKE	Compara cadeia de caracteres utilizando padrões de comparação: % substitui zero, um ou mais caracteres _ substitui um caractere	
	LIKE 'A%'	inicia com a letra A
	LIKE '%A'	termina com a letra A
	LIKE '%A%'	tem a letra A em qualquer posição
	LIKE 'A_'	string de dois caracteres, inicia com a letra A
	LIKE '_A'	string de dois caracteres, termina com a letra A
	LIKE '_A_'	string de três caracteres, letra A na segunda posição
	LIKE '%A_'	tem a letra A na penúltima posição
	LIKE '_A%'	tem a letra A na segunda posição
NOT LIKE	Negação do operador LIKE	
IN	Testa se um valor pertence a um conjunto de valores	
NOT IN	Negação do operador IN	
BETWEEN	Determina um intervalo de busca: BETWENN 'valor1' AND 'valor2'	
NOT BETWEEN	Negação do operador BETWEEN	
EXISTS	Verifica se um valor existe em um conjunto, levando em conta os valores nulos. Fato não considerado pelo operador IN	
NOT EXISTS	Negação do operador EXISTS	

7. SELECT

```
SELECT [DISTINCT | ALL] NOME_DA_COLUNA,NOME_DA_COLUNA
FROM NOME_DA_TABELA
[WHERE condição]
[GROUP BY NOME_DA_COLUNA
 HAVING codição]
[ORDER BY NOME_DA_COLUNA,NOME_DA_COLUNA];
```

Selecionar todos os campos da tabela:

```
SELECT * FROM CLIENTE;
```

Selecionar um campo da tabela:

```
SELECT NOME FROM CLIENTE;
```

Selecionar mais de um campo da tabela:

```
SELECT CODCLI, NOME FROM CLIENTE;
```

ALIAS: Atribui um apelido ao nome da coluna (somente na consulta):

```
SELECT NOME NOMECLI FROM CLIENTE;
```

DISTINCT: não exibe dados duplicados

```
SELECT DISTINCT UF FROM CLIENTE;
```

ORDER BY: Classifica os resultados da pesquisa (ASC: ordem ascendente, DESC: ordem descendente)

```
SELECT * FROM CLIENTE ORDER BY NOME ASC;
ou
SELECT * FROM CLIENTE ORDER BY NOME;
```

```
SELECT * FROM CLIENTE ORDER BY NOME DESC;
```

WHERE: Seleciona linhas que satisfazem determinado critério

```
SELECT * FROM CLIENTE WHERE UF = 'SP';
SELECT * FROM CLIENTE WHERE UF = 'SP' AND CODCLI > 1003;
```

GROUP BY: Agrupa as linhas selecionadas e retorna uma única linha de informação para cada grupo

```
SELECT COUNT(NOME), UF FROM CLIENTE GROUP BY UF;
```

Observação: A consulta acima retornará as quantidades de clientes (agrupados) por UF (estado).

HAVING: Filtra as linhas retornadas exibindo apenas aquelas cuja expressão seja TRUE

```
SELECT COUNT(NOME), UF FROM CLIENTE GROUP BY UF HAVING UF <> 'SP';
```

Observação: A consulta acima retornará as quantidades de clientes (agrupados) por UF (estado) para UF diferente de 'SP'.

SELECT FOR UPDATE

Bloqueia todas as linhas selecionadas. Nenhuma alteração poderá ser realizada em sessões diferentes daquela que emitiu o comando. O bloqueio será mantido até que a sessão que emitiu o comando realize o COMMIT ou o ROLLBACK.

```
SELECT * FROM NOME DA TABELA FOR UPDATE;
```

Exemplo:

```
SELECT * FROM CLIENTE WHERE CODCLI BETWEEN 1 AND 10 FOR UPDATE;
```

Observação: As linhas com CODCLI entre 1 e 10 estarão bloqueadas para as outras sessões até que seja emitido o COMMIT ou o ROLLBACK.

CASE

Permite a utilização da estrutura IF...THEN...ELSE em uma consulta:

Exemplo:

```
SELECT NOME,
CASE UF
WHEN 'SP' THEN 'SÃO PAULO'
WHEN 'RJ' THEN 'RIO DE JANEIRO'
ELSE 'OUTRO ESTADO'
END "ESTADO"
FROM CLIENTE;
```

8. FUNÇÕES DE LINHA

UPPER

Retorna todas as letras de todas as palavras que compõem o argumento em letras maiúsculas.

```
SELECT UPPER(NOME) FROM CLIENTE;
```

Observação: Para localizar um determinado nome e não tiver certeza de como este nome foi inserido na tabela (todos caracteres em maiúsculas, somente o primeiro caractere em maiúscula, etc.), utilize:

```
SELECT * FROM CLIENTE WHERE UPPER(NOME) = 'ANTONIO ALVARES';
```

LOWER

Retorna todas as letras de todas as palavras que compõem o argumento em letras minúsculas.

```
SELECT LOWER(NOME) FROM CLIENTE;
```

INITCAP

Retorna a letra inicial de cada palavra em maiúscula e as demais em minúsculas.

```
SELECT INITCAP(NOME) FROM CLIENTE;
```

LPAD

Preenchimento à esquerda.

```
SELECT LPAD (NOME_DA_COLUNA,tamanho,caracter_de_preenchimento) FROM NOME_DA_TABELA;
```

Exemplo 1: Preenchimento com '*' à esquerda

```
SELECT LPAD(UF,10,'*') ESTADO FROM CLIENTE;
```

Observação: A palavra ESTADO acima é utilizada como 'rótulo' para a coluna UF.

Exemplo 2: Preenchimento com espaços em branco à esquerda

```
SELECT LPAD(UF,10) ESTADO FROM CLIENTE;
```

RPAD

Preenchimento à direita.

```
SELECT LPAD (NOME_DA_COLUNA,tamanho,caracter_de_preenchimento) FROM NOME_DA_TABELA;
```

Exemplo: Preenchimento com '*' à direita.

```
SELECT LPAD(UF,10,'*') ESTADO FROM CLIENTE;
```

SUBSTR

Retorna uma string (extraída de uma coluna, expressão ou constante) cujo comprimento será determinado pelo parâmetro tamanho.

```
SELECT SUBSTR(NOME_DA_COLUNA,posicao_inicial,tamanho) FROM NOME_DA_TABELA;
```

Exemplo 1: Retorna quatro caracteres a partir da posição um contidos na coluna NOME da tabela CLIENTE

```
SELECT SUBSTR(NOME,1,4) NOME FROM CLIENTE;
```

Exemplo 2: Retorna os quatro últimos caracteres contidos na coluna NOME da tabela CLIENTE

```
SELECT SUBSTR(NOME,4) NOME FROM CLIENTE;
```

9. FUNÇÕES DE GRUPO

PROJETO		
CODPROJ	**VALOR**	**CODDEPT**
1001	250.000,00	D1
1002	300.000,00	D2
1003	500.000,00	D1
1004	600.000,00	D2

AVG

Retorna a **média** aritmética de uma coluna ou expressão para o agrupamento.

Exemplo: Retorna o valor médio dos projetos agrupados por departamento

```
SELECT CODDEPT, AVG(VALOR) FROM PROJETO GROUP BY CODDEPT;
```

MAX

Retorna o **maior** valor de uma coluna ou expressão para o agrupamento.

Exemplo: Retorna o maior valor dos projetos de cada departamento

```
SELECT CODDEPT, MAX(VALOR) FROM PROJETO GROUP BY CODDEPT;
```

MIN

Retorna o **menor** valor de uma coluna ou expressão para o agrupamento.

Exemplo: Retorna o menor valor dos projetos de cada departamento

```
SELECT CODDEPT, MIN(VALOR) FROM PROJETO GROUP BY CODDEPT;
```

COUNT

Conta o número de vezes que o argumento mudou de valor para o agrupamento.

Se o argumento for *, conta as linhas selecionadas incluindo as de valor NULL.

Exemplo: Retorna a quantidade projetos de cada departamento

```
SELECT CODDEPT, COUNT(CODPROJ) FROM PROJETO GROUP BY CODDEPT;
```

SUM

Retorna a **somatória** de uma coluna ou expressão de agrupamento.

Se o argumento for *, conta as linhas selecionadas incluindo as de valor NULL.

Exemplo: Retorna o valor total dos projetos de cada departamento

```
SELECT CODDEPT, SUM(VALOR) FROM PROJETO GROUP BY CODDEPT;
```

10. FUNÇÕES NUMÉRICAS

ABS

Retorna o valor absoluto do número passado como parâmetro.

```
SELECT ABS (-10) FROM DUAL;
```

```
  ABS(-10)
----------
        10
```

CEIL

Retorna número inteiro maior ou igual ao número do argumento.

```
SELECT CEIL (10.1) FROM DUAL;
```

```
CEIL(10.1)
----------
        11
```

FLOOR

Retorna o maior número inteiro menor ou igual ao argumento.

```
SELECT FLOOR (10.1) FROM DUAL;
```

```
FLOOR(10.1)
-----------
         10
```

ROUND

Retorna o número arredondado em n casas decimais.

```
SELECT ROUND (10.8475,2) FROM DUAL;
```

```
ROUND(10.8475,2)
----------------
           10,85
```

TRUNC

Retorna o número truncado em n casas decimais.

```
SELECT TRUNC (10.8475,2) FROM DUAL;
```

```
TRUNC(10.8475,2)
----------------
           10,84
```

MOD

Retorna o resto da divisão do dividendo pelo divisor.

```
SELECT MOD (5,2) FROM DUAL;
```

```
  MOD(5,2)
----------
         1
```

POWER

Retorna o valor da potenciação.

```
SELECT POWER (4,2) FROM DUAL;
```

```
POWER(4,2)
----------
        16
```

SQRT

Retorna a raiz quadrada do número.

```
SELECT SQRT (16) FROM DUAL;
```

```
  SQRT(16)
----------
         4
```

SIGN

Retorna 1 para valores maiores que zero, 0 quando o valor for zero e -1 quando o valor for menor que zero.

```
SELECT SIGN (10) FROM DUAL;
```

```
  SIGN(10)
----------
         1
```

```
SELECT SIGN (0) FROM DUAL;
```

```
   SIGN(0)
----------
         0
```

```
SELECT SIGN (-10) FROM DUAL;
```

```
SIGN(-10)
----------
        -1
```

11. FUNÇÕES DE CONVERSÃO

TO_CHAR

Converte um valor tipo DATE ou NUMBER para um valor CHAR.

Observação: Para o próximo exemplo altere no SQL*Plus a largura das colunas DIA e MES para 3. Utilize para isso os seguintes comandos:

```
COL DIA FORMAT A3
COL MES FORMAT A3
```

```
SELECT TO_CHAR (SYSDATE,'DD') DIA,
TO_CHAR (SYSDATE,'MM') MES,
TO_CHAR (SYSDATE,'YYYY') ANO
FROM DUAL;
```

```
DIA MES ANO
--- --- ----
12  10  2011
```

```
SELECT TO_CHAR (SYSDATE,'DD "DE" MONTH "DE" YYYY') "DATA ATUAL" FROM DUAL;
```

```
DATA ATUAL
--------------------
12 DE OUTUBRO DE 2011
```

```
SELECT TO_CHAR(1250,'999,999.99') FROM DUAL;
```

```
1,250.00
```

TO_DATE

Converte uma expressão tipo CHAR para DATE.

```
INSERT INTO TESTE (DATA_NASC)
VALUES (TO_DATE('10/20/1980','MM/DD/YYYY'));
```

Observação: Utilize a função TO_DATE para inserir datas em tabelas quando não souber o formato utilizado no servidor (DD/MM/YYYY ou MM/DD/YYYY).

TO_NUMBER

Converte uma expressão (válida) do tipo CHAR para NUMBER.

```
SELECT TO_NUMBER('$1,250.00','$999,999.99') FROM DUAL;
```

```
1250
```

12. OUTRAS FUNÇÕES

NVL

Retorna um valor não nulo quando o termo nulo for encontrado.

Para retornar 0 (zero) quando o termo nulo for encontrado:

```
SELECT NVL(NOME_DA_COLUNA,0) FROM NOME_DA_TABELA;
```

NULLIF

Retorna NULL se os dois valores forem iguais, senão retorna o primeiro valor.

```
SELECT NULLIF(15,15) FROM DUAL;  --  RETORNA NULL
```

```
SELECT NULLIF(15,20) FROM DUAL;  --  RETORNA 15
```

DECODE

Implementa a lógica condicional if-then-else.

```
SELECT DECODE(15,15,'IGUAIS','DIFERENTES') FROM DUAL;  --  RETORNA IGUAIS
```

```
SELECT DECODE(15,20,'IGUAIS','DIFERENTES') FROM DUAL;  --  RETORNA DIFERENTES
```

13. SUBQUERY

Consulta dentro de um comando SQL (SELECT, UPDATE, DELETE ou INSERT).

CLIENTE	
CODCLI	NOME
1001	Antonio Alvares
1002	Beatriz Bernardes
1003	Claudio Cardoso
1004	Daniela Dantas

PEDIDO	
NRPED	CODCLI
1	1002
2	1003
3	1002
4	

Qual o nome do cliente que efetuou o pedido número 1?

```
SELECT NOME FROM CLIENTE WHERE CODCLI =
(SELECT CODCLI FROM PEDIDO WHERE NRPED = 1);
```

Observação: A query `SELECT CODCLI FROM PEDIDO WHERE NRPED = 1` é executada **primeiro** e apresenta como resultado **1002**. A query `SELECT NOME FROM CLIENTE WHERE CODCLI = 1002` apresenta o nome do cliente: **Beatriz Bernardes**.

Qual o código do cliente que efetuou o último pedido?

```
SELECT CODCLI FROM PEDIDO WHERE NRPED =
(SELECT MAX(NRPED) FROM PEDIDO);
```

Quais os nomes clientes que efetuaram pelo menos um pedido?

```
SELECT NOME FROM CLIENTE WHERE CODCLI IN
(SELECT CODCLI FROM PEDIDO);
```

Os seguintes operadores podem ser utilizados:

OPERADOR	FUNÇÃO
=, <>, <, <=, >, >=	Relacionais usados em subqueries do tipo uma linha.
IN	Testa se um valor pertence a um conjunto de valores.
NOT IN	Testa se um valor não pertence a um conjunto de valores.
ANY	Verifica se um determinado argumento casa com qualquer membro de um conjunto.
ALL	Verifica se um determinado argumento casa com todos os membros de um conjunto.
EXISTS	Verifica se um valor existe em um conjunto, levando em conta os valores nulos. Fato não considerado pelo operador IN.
NOT EXISTS	Negação do operador EXISTS.

14. JOINS – JUNÇÕES DE TABELAS

EQUI JOIN

Reúne campos iguais de tabelas diferentes.

```
SELECT C.NOME, P.NRPED
FROM CLIENTE C
INNER JOIN PEDIDO P
ON C.CODCLI = P.CODCLI;
```

```
NOME                 NRPED
-------------------- ----------
Beatriz Bernardes     1
Claudio Cardoso       2
Beatriz Bernardes     3
```

Observação: **C** e **P** foram os alias (apelidos) utilizados respectivamente para as tabelas **CLIENTE** e **PEDIDO**.

NATURAL JOIN (JUNÇÃO NATURAL)

Quando as tabelas sobre as quais queremos realizar a junção apresentam colunas com o mesmo nome e para que, nesses casos, não seja necessário explicitar o nome das colunas, utilizamos a JUNÇÃO NATURAL.

```
SELECT C.NOME, P.NRPED
FROM CLIENTE C
NATURAL INNER JOIN PEDIDO P;
```

```
NOME                 NRPED
-------------------- ----------
Beatriz Bernardes     1
Claudio Cardoso       2
Beatriz Bernardes     3
```

JUNÇÃO BASEADA EM NOMES DE COLUNAS

Na JUNÇÃO NATURAL todas as colunas de mesmo nome nas tabelas são utilizadas para realizar a junção. Porém, na JUNÇÃO BASEADA EM NOMES DE COLUNAS somente são utilizadas as colunas listadas na cláusula USING.

```
SELECT C.NOME, P.NRPED
FROM CLIENTE C
INNER JOIN PEDIDO P
USING (CODCLI);
```

```
NOME                 NRPED
-------------------- ----------
Beatriz Bernardes     1
Claudio Cardoso       2
Beatriz Bernardes     3
```

OUTER JOIN (JUNÇÃO EXTERNA)

Além de mostrar registros cujos campos em comum estejam presentes nas duas tabelas, ainda mostra os que faltam.

LEFT OUTER JOIN (JUNÇÃO EXTERNA À ESQUERDA)

Apresentar os nomes de TODOS os clientes e os números dos pedidos:

```
SELECT C.NOME, P.NRPED
FROM CLIENTE C
LEFT OUTER JOIN PEDIDO P
ON C.CODCLI = P.CODCLI;
```

```
NOME                 NRPED
-------------------- ----------
Beatriz Bernardes     1
Claudio Cardoso       2
Beatriz Bernardes     3
Antonio Alvares
Daniela Dantas
```

Observação: A **primeira tabela** da consulta é considerada **esquerda**.

RIGHT OUTER JOIN (JUNÇÃO EXTERNA À DIREITA)

Apresentar os nomes dos clientes e os números de TODOS os pedidos:

```
SELECT C.NOME, P.NRPED
FROM CLIENTE C
RIGHT OUTER JOIN PEDIDO P
ON C.CODCLI = P.CODCLI;
```

```
NOME                 NRPED
-------------------- ----------
Beatriz Bernardes     1
Claudio Cardoso       2
Beatriz Bernardes     3
                      4
```

LEFT OUTER JOIN (JUNÇÃO EXTERNA À ESQUERDA)

Apresentar os nomes de TODOS os clientes e os números de TODOS os pedidos:

```
SELECT C.NOME, P.NRPED
FROM CLIENTE C
FULL OUTER JOIN PEDIDO P
ON C.CODCLI = P.CODCLI;
```

```
NOME                 NRPED
-------------------- ----------
Beatriz Bernardes     1
Claudio Cardoso       2
Beatriz Bernardes     3
Antonio Alvares
Daniela Dantas
                      4
```

NON EQUI JOIN

Reúne campos de tabelas que não têm valores em comum.

PRODUTO	
CODPROD	VALOR
101	75.00
102	120.00
103	30.00

CLASSE		
FAIXA	VALORMIN	VALORMAX
A	0.00	50.00
B	50.01	100.00
C	100.01	150.00

```
SELECT P.CODPROD, P.VALOR, C.FAIXA
FROM PRODUTO P
INNER JOIN CLASSE C
ON P.VALOR BETWEEN C.VALORMIN AND C.VALORMAX;
```

```
CODPROD  VALOR   FAIXA
-------  ------  -----
    101   75.00  B
    102  120.00  C
    103   30.00  A
```

Observação: No exemplo acima foi utilizado o operador relacional BETWEEN. Pode-se utilizar em JUNÇÕES do tipo NON EQUI JOIN qualquer operador relacional, exceto = (igual).

SELF JOIN

Relaciona dois campos de uma mesma tabela que sejam do mesmo tipo.

CURSO		
CODCURSO	NOMECURSO	CODPREREQ
11	WINDOWS BÁSICO	
12	WINDOWS AVANÇADO	11
13	WORD BÁSICO	
14	WORD AVANÇADO	13
15	EXCEL BÁSICO	
16	EXCEL AVANÇADO	15

```
SELECT C.NOMECURSO, P.NOMECURSO "PRÉ-REQUISITO"
FROM CURSO C
INNER JOIN CURSO P
ON C.CODPREREQ = P.CODCURSO;
```

```
NM_CURSO                 PRÉ-REQUISITO
------------------------ ----------------------
WINDOWS AVANÇADO         WINDOWS BÁSICO
WORD AVANÇADO            WORD BÁSICO
EXCEL AVANÇADO           EXCEL BÁSICO
```

CROSS JOIN (JUNÇÃO CRUZADA)

Apresenta todas as combinações possíveis entre elementos de duas tabelas.

TIME_SP	
CODTIME	NOMETIME
101	CORINTHIANS
102	PALMEIRAS
103	SANTOS
104	SÃO PAULO

TIME_RJ	
CODTIME	NOMETIME
201	FLAMENGO
202	VASCO
203	FLUMINENSE
204	VASCO

```
SELECT SP.NOMETIME "TIME SP",RJ.NOMETIME "TIME RJ"
FROM TIME_SP SP
CROSS JOIN TIME_RJ RJ;
```

```
TIME SP         TIME RJ
--------------- ---------------
CORINTHIANS     BOTAFOGO
CORINTHIANS     FLAMENGO
CORINTHIANS     FLUMINENSE
CORINTHIANS     VASCO
PALMEIRAS       BOTAFOGO
PALMEIRAS       FLAMENGO
PALMEIRAS       FLUMINENSE
PALMEIRAS       VASCO
SANTOS          BOTAFOGO
SANTOS          FLAMENGO
SANTOS          FLUMINENSE
SANTOS          VASCO
SÃO PAULO       BOTAFOGO
SÃO PAULO       FLAMENGO
SÃO PAULO       FLUMINENSE
SÃO PAULO       VASCO
```

15. OPERAÇÕES DE CONJUNTO

UNION (UNIÃO)

Apresenta como resultado a **união** de todas as linhas que foram recuperadas por dois comandos SQL realizados em separado.

- Os comandos devem retornar o mesmo número de colunas;
- As colunas correspondentes em cada comando devem possuir os mesmos tipos de dados.

```
SELECT NOME_DA_COLUNA_1, NOME_DA_COLUNA_2
FROM NOME_DA_TABELA_1
UNION [ALL]
SELECT NOME_DA_COLUNA_3, NOME_DA_COLUNA_4
FROM NOME_DA_TABELA_2;
```

O predicado ALL fará com que apareçam linhas repetidas (se for o caso).

INTERSECT (INTERSEÇÃO)

Apresenta como resultado a **interseção** de todas as linhas que foram recuperadas por dois comandos SQL realizados em separado.

- Os comandos devem retornar o mesmo número de colunas;
- As colunas correspondentes em cada comando devem possuir os mesmos tipos de dados.

```
SELECT NOME_DA_COLUNA_1, NOME_DA_COLUNA_2
FROM NOME_DA_TABELA_1
INTERSECT
SELECT NOME_DA_COLUNA_3, NOME_DA_COLUNA_4
FROM NOME_DA_TABELA_2;
```

MINUS (DIFERENÇA)

Apresenta como resultado a **diferença** entre as linhas que foram recuperadas por dois comandos SQL realizados em separado.

- Os comandos devem retornar o mesmo número de colunas
- As colunas correspondentes em cada comando devem possuir os mesmos tipos de dados
- A ordem de declaração das consultas com relação ao predicado MINUS altera o resultado final.

```
SELECT NOME_DA_COLUNA_1, NOME_DA_COLUNA_2
FROM NOME_DA_TABELA_1
MINUS
SELECT NOME_DA_COLUNA_3, NOME_DA_COLUNA_4
FROM NOME_DA_TABELA_2;
```

Observação: Outros bancos de dados utilizam o predicado EXCEPT para DIFERENÇA.

CLIENTE_A	
CPF	**NOME**
11122233344	Antonio Alvares
22233344455	Beatriz Bernardes
33344455566	Claudio Cardoso
44455566677	Daniela Dantas

CLIENTE_B	
CPF	**NOME**
22233344455	Beatriz Bernardes
44455566677	Daniela Dantas
55566677788	Ernesto Ermenegildo
66677788899	Fabiana Fonseca

```
SELECT * FROM CLIENTE_A
UNION
SELECT * FROM CLIENTE_B;
```

```
CPF          NOME
-----------  --------------------
11122233344  Antonio Alvares
22233344455  Beatriz Bernardes
33344455566  Claudio Cardoso
44455566677  Daniela Dantas
55566677788  Ernesto Ermenegildo
66677788899  Fabiana Fonseca
```

```
SELECT * FROM CLIENTE_A
INTERSECT
SELECT * FROM CLIENTE_B;
```

```
CPF          NOME
-----------  --------------------
22233344455  Beatriz Bernardes
44455566677  Daniela Dantas
```

```
SELECT * FROM CLIENTE_A
MINUS
SELECT * FROM CLIENTE_B;
```

```
CPF          NOME
-----------  --------------------
11122233344  Antonio Alvares
33344455566  Claudio Cardoso
```

16. MERGE

Permite selecionar linhas de uma ou mais fontes (tabelas, por exemplo) para inserção ou atualização. Pode executar em um único comando as três operações: INSERT, UPDATE e DELETE.

No exemplo a seguir são apresentadas duas tabelas (PRODUTO e PRODUTO_NOVO). A tabela PRODUTO deverá ser atualizada conforme segue:

1. Se o produto já constar na tabela PRODUTO, seu valor deverá ser simplesmente atualizado conforme o valor da tabela PRODUTO_NOVO;
2. Se o produto não existir na tabela PRODUTO, deverá ser inserido conforme os dados da tabela PRODUTO_NOVO.

PRODUTO		
CODPRO	**DESCPRO**	**VALORPRO**
11	PRODUTO 11	10.35
12	PRODUTO 12	12.55

PRODUTO_NOVO		
CODPRO	**DESCPRO**	**VALORPRO**
12	PRODUTO 12	14.95
13	PRODUTO 13	18.45

Observe como realizar as alterações na tabela PRODUTO utilizando MERGE:

```
MERGE INTO PRODUTO P
USING PRODUTO_NOVO N
ON (P.CODPRO = N.CODPRO)
WHEN MATCHED THEN
UPDATE SET P.VALORPRO = N.VALORPRO
WHEN NOT MATCHED THEN
INSERT (CODPRO,DESCPRO,VALORPRO)
VALUES (N.CODPRO,N.DESCPRO,N.VALORPRO);
```

O resultado será o seguinte:

PRODUTO		
CODPRO	**DESCPRO**	**VALORPRO**
11	PRODUTO 11	10.35
12	PRODUTO 12	14.95
13	PRODUTO 13	18.45

17. VIEW

Tabela virtual que não armazena fisicamente os dados. Apresenta colunas de uma ou mais tabelas.

CRIANDO UMA VIEW

Para criar uma view utilize:

```
CREATE OR REPLACE VIEW NOME_DA_VIEW AS ...;
```

Observação: REPLACE recria a view, se ela já existir.

Exemplo:

```
CREATE VIEW CLIENTE_SP AS SELECT * FROM CLIENTE WHERE UF = 'SP';
```

A view foi criada com base na tabela CLIENTE, apenas com as linhas dos clientes do estado de São Paulo.

EXCLUINDO UMA VIEW

Para excluir uma view utilize:

```
DROP VIEW NOME_DA_VIEW;
```

Exemplo:

```
DROP VIEW CLIENTE_SP;
```

Observação: O comando DROP VIEW exclui a view do dicionário de dados. Não terá nenhum outro efeito sobre as tabelas referenciadas.

18. INDEX

Fornece acesso mais rápido a linhas específicas, eliminando a necessidade de uma varredura completa na tabela.

CRIANDO UM INDEX

Para criar um índice utilize o comando CREATE INDEX:

```
CREATE INDEX NOME_DO_INDEX ON NOME_DA_TABELA(NOME_DA_COLUNA);
```

Exemplo:

```
CREATE INDEX CLIENTE_NOME_IN ON CLIENTE(NOME);
```

EXCLUINDO UM ÍNDICE

Para excluir um índice utilize o comando DROP INDEX:

```
DROP INDEX NOME_DO_INDEX;
```

Exemplo:

```
DROP INDEX CLIENTE_NOME_IN;
```

19. SEQUENCE

Sequência ou contador automático utilizado para gerar identificadores únicos (códigos de produtos, números de pedidos, etc.).

Para criar uma SEQUENCE utilizamos o seguinte comando:

```
CREATE SEQUENCE NOME_DA_SEQUENCE;
```

O comando acima criará uma nova sequência que inicia em 1 e incrementada também em 1.

No exemplo a seguir criamos uma sequência que tem início em 10 e incrementada em 10. Quando atingir o valor 1000 (MAXVALUE) voltará ao valor inicial 10 (CYCLE).

```
CREATE SEQUENCE TESTE
START WITH 10
INCREMENT BY 10
MAXVALUE 1000
CYCLE;
```

Para consultar a sequência utilizamos o comando:

```
SELECT NOME_DA_SEQUENCE.CURRVAL FROM DUAL;
```

Exemplo: Inserir uma linha na tabela CLIENTE utilizando a sequência SEQ_CLI:

```
INSERT INTO CLIENTE (CODIGO,NOME) VALUES (SEQ_CLI.NEXTVAL,'ANTONIO');
```

Para verificar as sequências criadas para um usuário e os seus parâmetros utilizamos o seguinte comando:

```
SELECT * FROM USER_SEQUENCES;
```

Para alterar a sequência criada no exemplo acima utilizamos o seguinte comando:

```
ALTER SEQUENCE TESTE
MINVALUE 50
INCREMENT BY 5;
```

Para eliminar uma sequência utilizamos o comando:

```
DROP SEQUENCE NOME_DA_SEQUENCE;
```

20. ROWID

Pseudo coluna (coluna virtual) que informa como o Oracle pode localizar em disco as linhas das tabelas (por arquivo, bloco dentro daquele arquivo e linha dentro daquele bloco).

```
SELECT ROWID FROM NOME_DA_TABELA;
```

Exemplo:

```
SELECT ROWID FROM CLIENTE;
```

APÊNDICE: USANDO O SQL*PLUS

EDITAR INSTRUÇÕES SQL

COMANDOS DE EDIÇÃO	
COMANDO	**DESCRIÇÃO**
A[PPEND] texto	Anexa o texto na linha atual.
C[HANGE] /antigo/novo	Altera o texto antigo para novo na linha atual.
CL[EAR] BUFF[ER]	Apaga todas as linhas do buffer.
DEL	Exclui a linha atual.
DEL x	Exclui a linha especificada em x.
L[IST]	Lista todas as linhas no buffer.
L[IST] x	Lista a linha especificada em x.
R[UN] ou /	Executa a instrução armazenada no buffer.
x	Torna corrente a linha especificada em x.

SALVAR, RECUPERAR E EXECUTAR ARQUIVOS

OPÇÕES PARA FORMATAÇÃO DE COLUNAS	
OPÇÃO	**DESCRIÇÃO**
SAV[E] nome_arquivo	Salva o conteúdo do buffer em um arquivo.
REPLACE nome_arquivo	Sobrescreve o conteúdo do buffer em um arquivo.
APPEND nome_arquivo	Anexa o conteúdo do buffer em um arquivo.
GET nome_arquivo	Recupera o conteúdo do arquivo.
STA[RT] nome_arquivo	Recupera o conteúdo do arquico e tenta executá-lo.
@ nome_arquivo	Idem ao STA[RT].
ED[IT]	Copia o conteúdo do buffer em um arquivo chamado afiedit.buf e inicia o editor de texto. Quando o editor é fechado, o conteúdo do arquivo editado é copiado para o buffer do SQL*Plus.
ED[IT] nome_arquivo	Idem ao ED[IT], porém pode-se especificar o nome do arquivo para edição.
SPO[OL] nome_arquivo	Copia a saída do SQL*Plus para o arquivo.
SPO[OL] OFF	Interrompe a cópia da saída do SQL*Plus no arquivo e fecha o arquivo.

FORMATAR COLUNAS

```
COL[UMN] coluna | apelido opções
```

COMANDOS DE EDIÇÃO	
OPÇÃO	**DESCRIÇÃO**
FOR[MAT] formato	Define o formato de exibição da coluna. (Consulte as opções para: strings, números e datas.)
HEA[DING]	Define o cabeçalho da coluna.
JUS[TIFY] LEFT | CENTER | RIGHT	Posiciona a saída da coluna na esquerda, no centro ou na direita.
WRA[PPED]	Passa o final de uma string para a próxima linha. Palavras podem ser divididas.
WOR[D_WRAPPED]	Passa o final de uma string para a próxima linha. Palavras não são divididas.
CLE[AR]	Limpa a formatação de colunas.

- Colunas com caracteres: Utilize Ax. Por exemplo: A15 define a largura da coluna como 15 caracteres.
- Colunas com números: Há várias opções. Por exemplo: 99.99 exibirá 5 como 5.00. $99.99 exibirá 5 como $5.00.
- Colunas com datas: DD-MM-YYYY (Dia e mês serão exibidos com dois dígitos e ano com quatro dígitos).

Exemplo:

```
CREATE TABLE PRODUTO (
CODIGO NUMBER(4),
NOME VARCHAR2(30),
VALOR NUMBER(4,2));

INSERT INTO PRODUTO VALUES (1,'IMPRESSORA JATO DE TINTA',99);

COLUMN CODIGO FORMAT 99
COLUMN NOME FORMAT A15 WORD WRAPPED
COLUMN VALOR FORMAT 99.99

SELECT * FROM PRODUTO;

CODIGO NOME             VALOR
------ -------------- ------
     1 IMPRESSORA JATO  99.00
       DE TINTA

-- LIMPAR A FORMATAÇÃO DA COLUNA NOME

COLUMN NOME CLEAR

CODIGO NOME                           VALOR
------ ------------------------------ ------
     1 IMPRESSORA JATO DE TINTA        99.00

-- LIMPAR A FORMATAÇÃO DE TODAS AS COLUNAS

CLEAR COLUMNS
```

DEFINIR O TAMANHO DA PÁGINA

Define que após determinado número de linhas o SQL*Plus exibirá novamente o cabeçalho (com nomes de colunas ou apelidos, etc.)

- Número máximo para tamanho de página: 50.000.

```
SET PAGESIZE 100
```

DEFINIR O TAMANHO DA LINHA

Define o tamanho da linha em quantidade de caracteres.

- Número máximo para tamanho de linha: 32.767.

```
SET LINESIZE 80
```

REFERÊNCIAS BIBLIOGRÁFICAS

BEIGHLEY, Lynn. Use a cabeça SQL. Rio de Janeiro: Alta Books, 2008.

FANDERUFF, Damaris. Dominando o Oracle 9i: Modelagem e Desenvolvimento. São Paulo: Makron, 2003.

GRAVES, Mark. Projeto de banco de dados com XML. 1a ed. São Paulo: Pearson, 2003.

MORELLI, Eduardo Terra. Oracle 9i fundamental: SQL, PL/SQL e administração. São Paulo: Érica, 2005.

SILVA, Robson S., Oracle Database 10g Express Edition. 1a ed. São Paulo: Érica, 2007.

WATSON, John, RAMKLASS, Roopesh. Oracle 11g - Fundamentos SQL I. Rio de Janeiro: Alta Books, 2010.