BANCO DE DADOS

SQL

Prof. Marcos Alexandruk

SUMÁRIO

BREVE HISTÓRIA DA LINGUAGEM SQL

Durante o desenvolvimento do Sistema R, pesquisadores da IBM desenvolveram a linguagem SEQUEL (Structured English Query Language), primeira linguagem de acesso a SGBDR (Sistemas Gerenciadores de Banco de Dados Relacionais).

E. F. Codd elaborou a teoria de que, usando a linguagem SQL, seria possível navegar por grandes quantidades de informações e obter rapidamente uma resposta à consulta.

Com o surgimento de um número cada vez maior de SGBDRs, 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.

Oito anos depois, 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 SQL3 e acrescentou-se uma nova parte que contempla o tratamento de XML.

PRINCIPAIS SISTEMAS DE BANCO DE DADOS QUE USAM SQL

- Apache Derby
- Caché
- DB2
- Ingres
- InterBase
- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server
- SQLite
- Sybase
- Informix
- Firebird
- HSQLDB (banco de dados feito em Java)
- PointBase (banco de dados relacional feito em Java)

ORACLE

A palavra *oráculo* significa *profecia* ou *alguém que faz previsões*. Acreditava-se que, se alguém fizesse uma pergunta a um oráculo obteria a resposta de uma divindade.

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 (mais tarde: Oracle Corporation). E lançado o Oracle 2.0 em linguagem assembly. É decidido que a próxima versão do SGDB será reescrita em liguagem C.

1983 Lançado o Oracle 3, o primeiro SGBD a rodar em mainframes e em minicomputadores.

1984 Desenvolvida a versão para PC do Oracle (utilizando 256 KB de memória). 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 Lançados pacotes financeiros e CASE (Computer Assisted Software Engeneering). 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.

1996 O conjunto de aplicações comerciais da Oracle inclui os seguintes pacotes: Financials, Suply Change Management, Manufacturing, Project Systems, Human Resources, Market Management. A Oracle adquire a IRI Software e passa a oferecer um conjunto de ferramentas OLAP (Online Analytical Processing), tecnologia que dará suporte ao data warehouse da Oracle.

1997 Lançado o Oracle 8.

1998 A Oracle oferece suporte ao Linux.

1999 Lançado o Oracle 8i.

2000 Lançado o Oracle 9i.

WEBSITES RECOMENDADOS

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

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

SQL (STRUTURED QUERY LANGUAGE)

Linguagem padrão para manipulação de dados em SGBRD (Sistemas Gerenciadores de Bancos de Dados Relacionais).

Os comandos SQL estão divididos em quatro categorias principais 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
CREATE INDEX	Cria um índice
ALTER TABLE	Altera ou insere uma coluna de uma tabela
ALTER INDEX	Altera um índice
DROP TABLE	Elimina uma tabela
DROP INDEX	Elimina um índice

DML (DATA MANIPULATION LANGUAGE)

Linguagem de Manipulação de Dados: usada para manipular dados.

COMANDO	FUNÇÃO
INSERT	Insere uma linha na tabela
DELETE	Exclui as linhas da tabela
UPDATE	Altera o conteúdo de colunas (campos) 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)

Obs.: 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
CREATE USER	Cria um usuário
ALTER USER	Altera um usuário
GRANT	Concede privilégios de acesso para um usuário
REVOKE	Revoga privilégios de acesso para um usuário

ALGUNS COMANDOS ÚTEIS

Alterar senha do usuário:

```
PASSWORD nome_usuario;
```

Alterar senha do usuário atual:

```
PASSWORD
```

Listar todos os usuários:

```
SELECT * FROM ALL_USERS;
```

Listar todas as tabelas:

```
SELECT TABLE_NAME FROM USER_TABLES;
```

Mostrar usuário atual:

```
SHOW USER
```

Informar constraints (restrições):

```
SELECT * FROM USER_CONSTRAINTS
WHERE TABLE_NAME = 'nome_da_tabela';
```

As informações são:

- **OWNER** Usuário que criou a restrição
- **CONSTRAINT_NAME** Nome da restrição
- **CONSTRAINT_TYPE** Tipo de restrição
  - P – PRIMARY KEY
  - R – FOREIGN KEY
  - C – CHECK
  - U – UNIQUE
- **TABLE NAME** Nome da tabela
- **SEARCH_CONDITION** Domínio permitido (check)
- **R_OWNER** Utilizada em restrições FK para indicar qual usuário criou a restrição PK na tabela referenciada
- **R_CONSTRAINT_NAME** Utilizada em restrições FK para indicar qual a restrição PK na tabela referenciada
- **STATUS** Indica se a restrição está ou não habilitada

NOMENCLATURA DE COLUNAS

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

TIPOS DE DADOS

Tipo	Descrição
char	Cadeia de caracteres de tamanho fixo n. O default é 1 e o máximo é 255.
varchar2 (n)	Cadeia de caracteres de tamanho variável. Tamanho máximo 4.000.
long	Cadeia de caracteres de tamanho variável. Tamanho máximo 2 GB. Só pode existir um campo deste tipo por tabela.
raw e long raw	Equivalente ao varchar2 e long, respectivamente. Utilizado para armazenar dados binários (sons, imagens, etc.) Limite: 2.000 bytes.
number (p,e)	Valores numéricos. Ex: number (5,2) armazena -999,99 a +999,99.
date	Armazena data e hora (inclusive minuto e segundo). Ocupam 7 bytes.

CRIAÇÃO DE TABELAS

Para criar uma tabela utilizamos o comando CREATE TABLE:

```
CREATE TABLE tabela1 (
coluna1 NUMBER(5),
coluna2 VARCHAR2(30));
```

VERIFICANDO A ESTRUTURA DE UMA TABELA

```
DESCRIBE tabela1;

DESC tabela1;
```

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 da tabela.

```
CREATE TABLE tabela1 (
coluna1 NUMBER(5),
coluna2 VARCHAR2(30),
CONSTRAINT tabela1_pk PRIMARY KEY(coluna1));

CREATE TABLE tabela1 (
coluna1 NUMBER(5),
coluna2 VARCHAR2(30),
CONSTRAINT tabela1_pk PRIMARY KEY(coluna1,coluna2));
```

CHAVE ESTRANGEIRA (FK – FOREIGN KEY)

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

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

```
CREATE TABLE tabela2 (
coluna1 NUMBER(5),
coluna2 NUMBER(5),
CONSTRAINT tabela2_pk PRIMARY KEY(coluna1),
CONSTRAINT tabela2_fk FOREIGN KEY(coluna2) REFERENCES tabela1(coluna1));

CREATE TABLE tabela2 (
coluna1 NUMBER(5),
coluna2 NUMBER(5),
CONSTRAINT tabela2_pk PRIMARY KEY(coluna1),
CONSTRAINT tabela2_fk FOREIGN KEY(coluna2) REFERENCES tabela1(coluna1)
ON DELETE SET NULL);
```

```
CREATE TABLE tabela2 (
coluna1 NUMBER(5),
coluna2 NUMBER(5),
CONSTRAINT tabela2_pk PRIMARY KEY(coluna1),
CONSTRAINT tabela2_fk FOREIGN KEY(coluna2) REFERENCES tabela1(coluna1)
ON DELETE CASCADE);
```

DEFAULT

Atribui um conteúdo-padrão a uma coluna da tabela.

```
CREATE TABLE tabela3 (
coluna1 NUMBER(5),
coluna2 NUMBER(5) DEFAULT 1,
coluna3 VARCHAR2(10) DEFAULT 'a',
coluna4 DATE DEFAULT SYSDATE);
```

NOT NULL

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

```
CREATE TABLE tabela4 (
coluna1 NUMBER(5),
coluna2 VARCHAR2(30) NOT NULL);

CREATE TABLE tabela4 (
coluna1 NUMBER(5),
coluna2 VARCHAR2(30) CONSTRAINT tabela4_nn NOT NULL);
```

UNIQUE

Indica que não poderá haver repetição no conteúdo da coluna.

CHAVE PRIMÁRIA	
Permite repetição?	NÃO
Permite valores nulos?	NÃO

UNIQUE	
Permite repetição?	NÃO
Permite valores nulos?	SIM

```
CREATE TABLE tabela5 (
coluna1 NUMBER(5),
coluna2 VARCHAR2(30) UNIQUE);

CREATE TABLE tabela5 (
coluna1 NUMBER(5),
coluna2 VARCHAR2(30) CONSTRAINT tabela5_un UNIQUE);
```

CHECK

Define um domínio de valores para o conteúdo da coluna.

```
CREATE TABLE tabela6 (
coluna1 NUMBER(5),
coluna2 CHAR(1) CONSTRAINT tabela6_ch CHECK (coluna2='M' OR coluna2='F'));

CREATE TABLE tabela6 (
coluna1 NUMBER(5),
coluna2 CHAR(2) CONSTRAINT tabela6_ch CHECK (coluna2 IN ('SP','RJ','MG')));

CREATE TABLE tabela6 (
coluna1 NUMBER(5),
coluna2 NUMBER(5) CONSTRAINT tabela6_ch CHECK (coluna2 < 1000));
```

DESATIVANDO CONSTRAINTS

```
ALTER TABLE tabela1 DISABLE CONSTRAINT tabela1_pk;
```

ATIVANDO CONSTRAINTS

```
ALTER TABLE tabela1 ENABLE CONSTRAINT tabela1_pk;
```

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 9i não havia 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 FK na tabela-filho, alterar os valores na coluna PK na tabela-pai, alterar (se necessário) os valores na coluna FK na tabela-filho e ativar novamente a restrição Foreign Key.

REMOVENDO CONSTRAINTS

```
ALTER TABLE tabela1 DROP CONSTRAINT tabela1_pk;
```

ADICIONANDO CONSTRAINTS

```
ALTER TABLE tabela1 ADD CONSTRAINT tabela1_pk PRIMARY KEY(coluna1);
```

EXCLUSÃO DE TABELAS

Para excluir uma tabela utilizamos o comando DROP TABLE:

```
DROP TABLE tabela1;
```

```
DROP TABLE tabela1 CASCADE CONSTRAINTS;
```

ALTERAÇÃO DE TABELAS

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

ADICIONANDO COLUNAS

```
ALTER TABLE tabela1
   ADD coluna3 VARCHAR2(50);
```

MODIFICANDO COLUNAS

Alterando a **largura** de uma coluna:

```
ALTER TABLE tabela1
   MODIFY coluna2 VARCHAR2(40);
```

Alterando o **nome** de uma coluna (Oracle 9.2.0 e posteriores):

```
ALTER TABLE tabela1
   RENAME COLUMN coluna2 TO coluna4;
```

EXCLUINDO COLUNAS

```
ALTER TABLE tabela1
   DROP coluna3;
```

```
ALTER TABLE tabela1
   DROP coluna3 CASCADE CONSTRAINTS;
```

ALTERANDO O NOME DE UMA TABELA

```
RENAME tabela1 TO tabela10;
```

CRIANDO UMA TABELA COM BASE EM OUTRA

Criando uma tabela com todas as linhas e colunas de outra:

```
CREATE TABLE tabela7 AS SELECT * FROM tabela10;
```

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

```
CREATE TABLE tabela7 (coluna1,coluna2) AS SELECT coluna1,coluna2 FROM tabela10;
```

TRUNCATE

Comando DDL utilizado para 'cortar' uma tabela.

```
TRUNCATE TABLE nome_tabela;
```

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

ÍNDICES

Fornecem acesso mais rápido a linhas específicas, eliminando a necessidade de varreduras completas de tabelas.

CRIANDO UM ÍNDICE

Para criar um índice utilizamos o comando CREATE INDEX:

```
CREATE INDEX tabela1_in ON tabela1 (coluna2);
```

EXCLUINDO UM ÍNDICE

Para excluir um índice utilizamos o comando DROP INDEX:

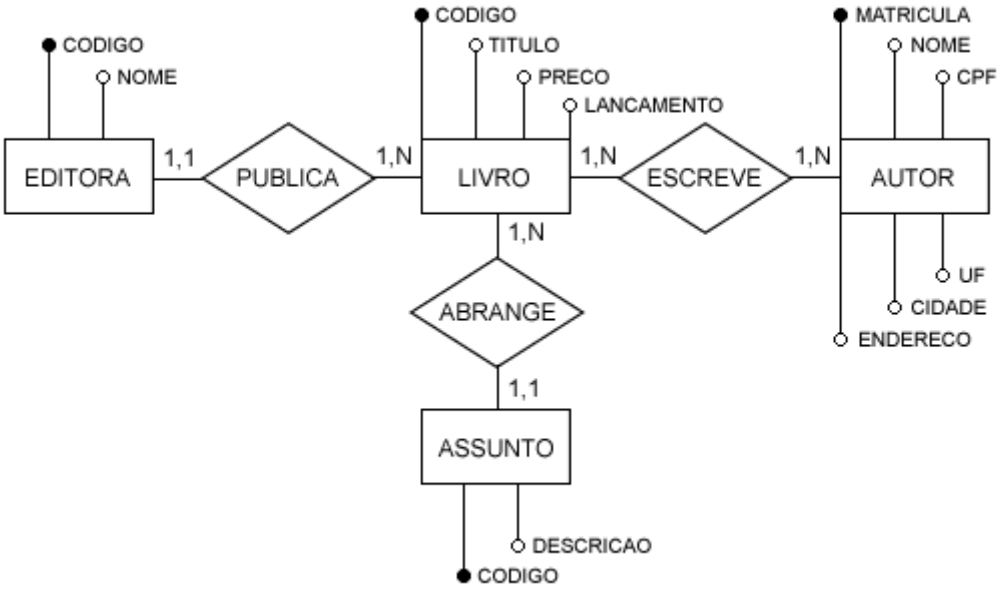
```
DROP INDEX tabela1_in;
```

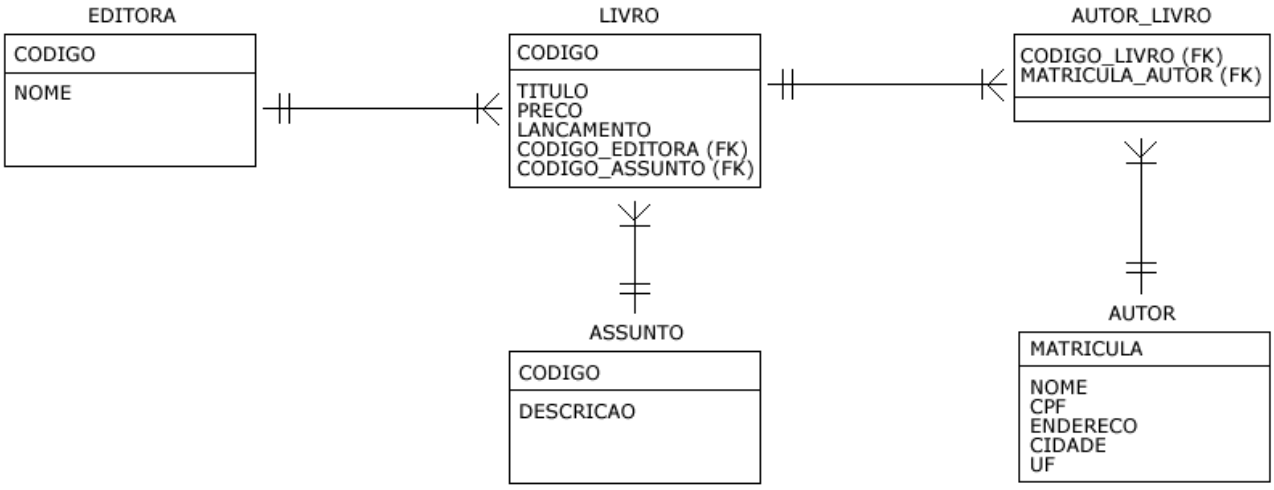
ROWID

Pseudo coluna (coluna virtual), 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_tabela;
```

DER - DIAGRAMA ENTIDADE RELACIONAMENTO





```
DROP TABLE EDITORA CASCADE CONSTRAINTS;
DROP TABLE ASSUNTO CASCADE CONSTRAINTS;
DROP TABLE AUTOR CASCADE CONSTRAINTS;
DROP TABLE LIVRO CASCADE CONSTRAINTS;
DROP TABLE AUTOR_LIVRO CASCADE CONSTRAINTS;

CREATE TABLE EDITORA (
CODIGO NUMBER(5),
NOME VARCHAR2(20) CONSTRAINT EDITORA_NOME_NNU NOT NULL UNIQUE,
CONSTRAINT EDITORA_PK PRIMARY KEY(CODIGO));

CREATE TABLE ASSUNTO (
CODIGO NUMBER(5),
DESCRICAO VARCHAR2(20) CONSTRAINT ASSUNTO_DESCRICAO_NN NOT NULL,
CONSTRAINT ASSUNTO_PK PRIMARY KEY(CODIGO));

CREATE TABLE AUTOR (
MATRICULA NUMBER(5),
NOME VARCHAR2(30) CONSTRAINT AUTOR_NOME_NN NOT NULL,
CPF VARCHAR2(12) CONSTRAINT AUTOR_CPF_UN UNIQUE,
ENDERECO VARCHAR2(30),
CIDADE VARCHAR2(30) DEFAULT 'SÃO PAULO',
UF CHAR(2) DEFAULT 'SP' CONSTRAINT AUTOR_UF_CH CHECK (UF IN ('SP', 'RJ', 'MG', 'ES')),
CONSTRAINT AUTOR_PK PRIMARY KEY(MATRICULA));

CREATE TABLE LIVRO (
CODIGO NUMBER(5),
TITULO VARCHAR2(30) CONSTRAINT LIVRO_TITULO_NN NOT NULL,
PRECO NUMBER(7,2),
LANCAMENTO DATE DEFAULT SYSDATE,
CODIGO_EDITORA NUMBER(5) NOT NULL,
CODIGO_ASSUNTO NUMBER(5) NOT NULL,
CONSTRAINT LIVRO_PK PRIMARY KEY(CODIGO),
CONSTRAINT LIVRO_EDITORA_FK FOREIGN KEY(CODIGO_EDITORA) REFERENCES EDITORA(CODIGO),
CONSTRAINT LIVRO_ASSUNTO_FK FOREIGN KEY(CODIGO_ASSUNTO) REFERENCES ASSUNTO(CODIGO));

CREATE TABLE AUTOR_LIVRO (
CODIGO_LIVRO NUMBER(5) NOT NULL,
MATRICULA_AUTOR NUMBER(5) NOT NULL,
CONSTRAINT AUTOR_LIVRO_PK PRIMARY KEY(CODIGO_LIVRO,MATRICULA_AUTOR),
CONSTRAINT AUTOR_LIVRO_LIVRO_FK FOREIGN KEY(CODIGO_LIVRO) REFERENCES LIVRO(CODIGO),
CONSTRAINT AUTOR_LIVRO_AUTOR_FK FOREIGN KEY(MATRICULA_AUTOR) REFERENCES AUTOR(MATRICULA));

INSERT INTO EDITORA VALUES (1,'EDITORA A');
INSERT INTO EDITORA VALUES (2,'EDITORA B');
INSERT INTO EDITORA VALUES (3,'EDITORA C');
INSERT INTO EDITORA VALUES (4,'EDITORA D');
INSERT INTO EDITORA VALUES (5,'EDITORA E');

INSERT INTO ASSUNTO VALUES (11,'ASSUNTO A');
INSERT INTO ASSUNTO VALUES (12,'ASSUNTO B');
INSERT INTO ASSUNTO VALUES (13,'ASSUNTO C');
INSERT INTO ASSUNTO VALUES (14,'ASSUNTO D');
INSERT INTO ASSUNTO VALUES (15,'ASSUNTO E');

INSERT INTO AUTOR VALUES (101,'AUTOR A','111222333-11','RUA A, 11','SÃO PAULO','SP');
INSERT INTO AUTOR VALUES (102,'AUTOR B','222333444-22','RUA B, 12','CAMPINAS','SP');
INSERT INTO AUTOR VALUES (103,'AUTOR C','333444555-33','RUA B, 13','RIO DE JANEIRO','RJ');
INSERT INTO AUTOR VALUES (104,'AUTOR D','444555666-44','RUA D, 14','NITERÓI','RJ');
INSERT INTO AUTOR VALUES (105,'AUTOR E','555666777-55','RUA E, 15','BELO HORIZONTE','MG');

INSERT INTO LIVRO VALUES (1001,'TITULO A',10.50,'01/01/2001',1,15);
INSERT INTO LIVRO VALUES (1002,'TITULO B',20.50,'01/01/2001',2,14);
INSERT INTO LIVRO VALUES (1003,'TITULO C',10.50,'01/01/2002',3,13);
INSERT INTO LIVRO VALUES (1004,'TITULO D',20.50,'01/01/2002',4,12);
INSERT INTO LIVRO VALUES (1005,'TITULO E',10.50,'01/01/2001',5,11);
INSERT INTO LIVRO VALUES (1006,'TITULO F',20.50,'01/01/2001',1,15);
INSERT INTO LIVRO VALUES (1007,'TITULO G',10.50,'01/01/2002',2,14);
INSERT INTO LIVRO VALUES (1008,'TITULO H',20.50,'01/01/2002',3,13);
INSERT INTO LIVRO VALUES (1009,'TITULO I',10.50,'01/01/2001',4,12);
INSERT INTO LIVRO VALUES (1010,'TITULO J',20.50,'01/01/2001',5,11);
INSERT INTO LIVRO VALUES (1011,'TITULO K',10.50,'01/01/2002',1,15);
INSERT INTO LIVRO VALUES (1012,'TITULO L',20.50,'01/01/2002',2,14);

INSERT INTO AUTOR_LIVRO VALUES (1001,101);
INSERT INTO AUTOR_LIVRO VALUES (1001,102);
INSERT INTO AUTOR_LIVRO VALUES (1002,101);
INSERT INTO AUTOR_LIVRO VALUES (1002,102);
INSERT INTO AUTOR_LIVRO VALUES (1003,103);
INSERT INTO AUTOR_LIVRO VALUES (1004,104);
INSERT INTO AUTOR_LIVRO VALUES (1005,101);
INSERT INTO AUTOR_LIVRO VALUES (1005,104);
INSERT INTO AUTOR_LIVRO VALUES (1006,101);
INSERT INTO AUTOR_LIVRO VALUES (1007,103);
INSERT INTO AUTOR_LIVRO VALUES (1008,102);
INSERT INTO AUTOR_LIVRO VALUES (1008,104);
INSERT INTO AUTOR_LIVRO VALUES (1009,104);
INSERT INTO AUTOR_LIVRO VALUES (1010,101);
INSERT INTO AUTOR_LIVRO VALUES (1010,105);
INSERT INTO AUTOR_LIVRO VALUES (1011,104);
INSERT INTO AUTOR_LIVRO VALUES (1012,105);
```

INSERT, UPDATE, DELETE

INSERT

Para incluir dados em uma tabela utilizamos o comando INSERT:

```
INSERT INTO nome_da_tabela (nome_da_coluna1,nome_da_coluna2...)
VALUES (valor_da_coluna1, valor_da_coluna2...);
```

EXEMPLO

```
INSERT INTO EDITORA (CODIGO,NOME)
VALUES (1,'Editora A');
```

UPDATE

Para modificar os dados inseridos em uma tabela utilizamos o comando UPDATE:

```
UPDATE nome_da_tabela
   SET (nome_da_coluna1=novo_valor1,nome_da_coluna2=novo_valor2...)
   WHERE ...;
```

EXEMPLO

```
UPDATE EDITORA
   SET NOME = 'Editora B'
   WHERE CODIGO = 1;
```

DELETE

Para excluir linhas de uma tabela utilizamos o comando DELETE:

```
DELETE nome_da_tabela       OU    DELETE FROM nome_da_tabela
   WHERE ...;                          WHERE ...;
```

EXEMPLO

```
DELETE EDITORA
   WHERE CODIGO = 1;

DELETE FROM EDITORA
   WHERE CODIGO = 1;
```

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

SQL - OPERADORES	
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.

SELECT

```
SELECT [DISTINCT | ALL] colunas
FROM tabelas
[WHERE condição]
[GROUP BY grupo
 HAVING codições_grupo]]
[ORDER BY colunas_classif];
```

Selecionar todos os campos da tabelas:

```
SELECT * FROM funcionarios;
```

Selecionar um campo da tabela:

```
SELECT nome FROM funcionarios;
```

Selecionar mais de um campo da tabela:

```
SELECT nome, salario FROM funcionarios;
```

ALIAS: Renomeia o nome da coluna (somente na consulta):

```
SELECT nome AS funcionario, salario AS rendimento FROM funcionarios;
```

DISTINCT: não exibe dados duplicados

```
SELECT cargo FROM funcionarios;
SELECT DISTINCT cargo FROM funcionarios;
```

ORDER BY: Classifica os resultados da pesquisa (asc, desc)

```
SELECT * FROM funcionarios ORDER BY cargo;
SELECT * FROM funcionarios ORDER BY cargo DESC;
```

WHERE: Seleciona linhas que satisfazem determinado critério

```
SELECT * FROM funcionarios WHERE cargo='Gerente';
SELECT * FROM funcionarios WHERE cargo='Gerente' AND salario>5000;
```

A cláusula WHERE utiliza os seguintes operadores de comparação:

```
igualdade          =
menor que          <
menor ou igual a   <=
maior que          >
maior ou igual a   >=
diferença          <>
```

VER: CLÁUSULA WHERE – página 11.

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

```
SELECT cargo, SUM(salario) FROM funcionarios GROUP BY cargo;
```

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

```
SELECT cargo FROM funcionarios GROUP BY by cargo HAVING cargo <> 'Gerente';
```

EXERCÍCIOS

1. Criar a tabela estados com os seguintes campos:

cod_estado	varchar2(2)	primary key
nome_estado	varchar2(20)	
regiao	varchar2(10)	
populacao	number(10)	

2. Inserir os seguintes registros:

'MG','Minas Gerais','Sudeste',20000000
'SP','São Paulo','Sudeste',38000000
'AM','Amazonas','Norte',3000000
'PA','Pará','Norte',6000000
'RS','Rio Grande do Sul','Sul',11000000
'PR','Paraná','Sul',10000000

3. Selecionar os estados da região Sudeste:

4. Selecionar os estados com população inferior a 10000000:

5. Selecionar os estados da região Sul com população superior a 10000000:

FUNÇÕES DE LINHA

UPPER

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

```
UPPER (nome_da_coluna)
```

EXEMPLO

```
SELECT UPPER(nome_pais) FROM paises;
```

NOTA: Imagine que você necessite localizar um determinado nome em uma tabela, mas não tem certeza de como este nome foi inserido na tabela (todos caracteres em maiúsculas, somente o primeiro caractere em maiúscula, etc.), neste caso a função UPPER poderá ser utilizada:

```
SELECT * FROM nome_da_tabela
   WHERE UPPER(nome_da_coluna) = 'VALOR';
```

LOWER

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

```
LOWER (nome_da_coluna)
```

EXEMPLO

```
SELECT LOWER(nome_pais) FROM paises;
```

INITCAP

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

```
INITCAP (nome_da_coluna)
```

EXEMPLO

```
SELECT INITCAP(nome_pais) FROM paises;
```

LPAD

Preenchimento à esquerda.

```
LPAD (nome_da_coluna,[tamanho],[caracter_de_preenchimento])
```

EXEMPLO 1: Preenchimento com '-'

```
SELECT LPAD(nome_pais,50,'-') paises FROM paises;
```

Nota: A primeira ocorrência da palavra 'paises' no comando acima será utilizada como 'rótulo' para coluna selecionada.

EXEMPLO 2: Preenchimento com espaços em branco

```
SELECT LPAD(nome_pais,50) paises FROM paises;
```

RPAD

Preenchimento à direita.

```
RPAD (nome_da_coluna,[tamanho],[caracter_de_preenchimento])
```

EXEMPLO 1: Preenchimento com '-'

```
SELECT RPAD(nome_pais,50,'-') paises FROM paises;
```

EXEMPLO 2: Preenchimento com espaços em branco

```
SELECT RPAD(nome_pais,50) paises FROM paises;
```

SUBSTR

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

```
SUBSTR(nome_da_coluna,posicao_inicial[,tamanho])
```

EXEMPLO 1

```
SELECT SUBSTR(nome_pais,1,5) paises FROM paises;
```

EXEMPLO 2

```
SELECT SUBSTR(nome_pais,3) paises FROM paises;
```

EXERCÍCIOS

1. Criar a tabela cep:

   ```
   cep      varchar2(9) primary key
   endereco varchar2(30)
   ```

2. Inserir os seguintes registros:

   ```
   '01001-000','rua direita'
   '01002-000','rua esquerda'
   '01003-000','rua central'
   ```

3. Retornar os registros da coluna endereco em maiúsculas:

4. Retornar os registros da coluna endereco em minúsculas:

5. Retornar os registros da coluna endereco com a letra inicial de cada palavra em maiúsculas e as demais em minúsculas:

6. Retornar os registros da coluna cep conforme a seguinte formatação:

   ```
   CEP
   --------------------
   -----------01001-000
   -----------01002-000
   -----------01003-000
   ```

7. Retornar os registros da endereco conforme a seguinte formatação:

   ```
   ENDERECO
   --------------------
   direita
   esquerda
   central
   ```

FUNÇÕES DE GRUPO

AVG

Retorna a média aritmética de uma coluna ou expressão.

```
AVG (nome_da_coluna ou expressão)
```

EXEMPLO

```
SELECT cd_produto, AVG(vl_unitario * qt_produto)
FROM item_pedido
GROUP BY cd_produto;
```

MAX

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

```
MAX (nome_da_coluna ou expressão)
```

EXEMPLO

```
SELECT cd_vendedor, MAX(nr_pedido) "ultimo pedido"
FROM pedido
GROUP BY cd_vendedor;
```

MIN

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

```
MIN (nome_da_coluna ou expressão)
```

EXEMPLO

```
SELECT cd_vendedor, MIN(nr_pedido) "primeiro pedido"
FROM pedido
GROUP BY cd_vendedor;
```

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.

```
COUNT (nome_da_coluna ou expressão)
```

EXEMPLO

```
SELECT cd_vendedor, COUNT(*) "total de pedidos"
FROM pedido
GROUP BY cd_vendedor;
```

SUM

Retorna a somatória de uma coluna ou expressão de agrupamento.

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

```
SUM (nome_da_coluna ou expressão)
```

EXEMPLO

```
SELECT cd_produto, SUM(vl_unitario * qt_produto) "total por produto"
FROM item_pedido
GROUP BY cd_produto;
```

EXERCÍCIOS

1. Criar e inserir dados nas tabelas "pedido" e "item_pedido" conforme os exemplos apresentados nos tópicos AVG, MAX, MIN, COUNT e SUM.

2. Fazer as consultas apresentadas nos tópicos acima.

FUNÇÕES NUMÉRICAS

ABS

Retorna o valor absoluto do número passado como parâmetro.

```
ABS (number)
```

EXEMPLO

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

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

CEIL

Retorna número inteiro maior ou igual ao número do argumento.

```
CEIL (number)
```

EXEMPLO

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

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

FLOOR

Retorna o maior número inteiro menor ou igual ao argumento.

```
FLOOR (number)
```

EXEMPLO

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

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

MOD

Retorna o resto da divisão do dividendo pelo divisor.

```
MOD (dividendo,divisor)
```

EXEMPLO

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

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

POWER

Retorna o valor da potenciação.

```
POWER (base,expoente)
```

EXEMPLO

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

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

SQRT

Retorna a raiz quadrada do número.

```
SQRT (number)
```

EXEMPLO

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

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

ROUND

Retorna o número arredondado em n casas decimais.

```
ROUND (number,[casas_decimais])
```

EXEMPLO

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

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

TRUNC

Retorna o número truncado em n casas decimais.

```
TRUNC (number,[casas_decimais])
```

EXEMPLO

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

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

SIGN

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

```
SIGN (number)
```

EXEMPLO 1

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

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

EXEMPLO 2

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

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

EXEMPLO 3

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

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

EXERCÍCIOS

1. Execute comandos SQL utilizando cada uma das funções numéricas apresentadas:

- ABS
- CEIL
- FLOOR
- MOD
- POWER
- SQRT
- ROUND
- TRUNC
- SIGN

FUNÇÕES DE CONVERSÃO

TO_CHAR

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

```
TO_CHAR (data[,formato_char])

TO_CHAR (number[,formato_char])
```

EXEMPLO 1

```
SELECT TO_CHAR (dt_emissao,'DD') AS dia
   TO_CHAR (dt_emissao,'MM') AS mes
   TO_CHAR (dt_emissao,'YYYY') AS ano
   FROM pedido;
```

```
dia mes ano
--- --- ----
28  10  2004
```

EXEMPLO 2

```
SELECT TO_CHAR (dt_emissao,'DD "DE" MONTH "DE" YYYY') "Pedidos" FROM pedido;
```

```
Pedidos
---------------------
28 DE OUTUBRO DE 2004
```

TO_DATE

Converte uma expressão tipo CHAR para DATE.

```
TO_DATE (char[,formato_char])
```

EXEMPLO

```
INSERT INTO pedido (nr_pedido,dt_emissao) values
   (2,TO_DATE('10/11/2004','DD/MM/YYYY'));
```

TO_NUMBER

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

```
TO_NUMBER (char,formato_number)
```

EXEMPLO

```
SELECT TO_NUMBER ('1,240.36','9,999.99') FROM dual;
```

```
TO_NUMBER('1,240.36','9,999.99')
--------------------------------
                        1240,36
```

EXERCÍCIOS

1. Execute comandos SQL utilizando cada uma das funções de conversão:

- TO_CHAR
- TO_DATE
- TO_NUMBER

SUBQUERY

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

EXEMPLOS:

```
CREATE TABLE cliente
(cd_cliente   NUMBER(4),
 nm_cliente   VARCHAR2(20));

create TABLE PEDIDO
(nr_pedido    NUMBER(4),
 cd_cliente   NUMBER(4));

INSERT INTO cliente VALUES (1,'Antonio');
INSERT INTO cliente VALUES (2,'Beatriz');
INSERT INTO cliente VALUES (3,'Claudio');

INSERT INTO pedido VALUES (10,1);
INSERT INTO pedido VALUES (11,1);
INSERT INTO pedido VALUES (12,2);
INSERT INTO pedido VALUES (14,'');
```

Qual o nome do cliente que efetuou o pedido número 10?

```
SELECT nm_cliente FROM cliente
WHERE cd_cliente =
(SELECT cd_cliente FROM pedido WHERE nr_pedido = 10);
```

Quais os clientes que efetuaram pelo menos um pedido?

```
SELECT cd_cliente, nm_cliente
FROM cliente
WHERE cd_cliente IN (SELECT cd_cliente FROM pedido);
```

Qual o código do cliente que efetuou o primeiro pedido?

```
SELECT cd_cliente
FROM pedido
WHERE nr_pedido = (SELECT MIN(nr_pedido) FROM pedido);
```

Podem-se usar os seguintes operadores:

=, <>, >, >=, <, <=	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.

EXERCÍCIOS

1. Crie as tabelas abaixo (além das tabelas cliente e pedido criadas anteriormente).

```
create table produto
(cd_produto number(4),
 nm_produto varchar2(30),
 vl_produto number(7,2));

create table historico
(nr_pedido number(4),
 cd_produto number(4),
 qt_produto number(4));

insert into produto values (101,'produto1',10.52);
insert into produto values (102,'produto2',50.75);
insert into produto values (103,'produto3',45.34);

insert into historico values (10,101,20);
insert into historico values (11,102,12);
insert into historico values (12,103,25);
```

Utilize subqueries para resolver as seguintes questões:

a. Atualizar o produto mais caro em 10%.

_____
_____
_____

b. Criar a tabela 'maiores' na qual constem os produtos cujo valor unitário seja maior que a média.

_____
_____

c. Eliminar o item mais barato da tabela 'maiores'.

_____
_____

EXERCÍCIOS

Utilize as tabelas criadas através do script da página 12 para responder as seguintes questões (utilize somente subqueries):

1. Apresentar os títulos dos livros que abordam o 'ASSUNTO B'.

2. Apresentar os nomes das editoras que publicaram livros sobre o 'ASSUNTO A'.

3. Apresentar os títulos dos livros escritos pelo 'AUTOR C'.

4. Apresentar os nomes dos autores que lançaram livros através da 'EDITORA D'.

5. Apresentar os nomes dos autores que lançaram livros através da 'EDITORA A' e SOBRE O 'ASSUNTO C'.

JOIN - JUNÇÕES DE TABELAS

EQUI JOIN

Reúne campos iguais de tabelas diferentes.

EXEMPLO 1.A

```
SELECT nm_cliente, nr_pedido
FROM cliente, pedido
WHERE cliente.cd_cliente = pedido.cd_cliente;
```

```
NM_CLIENTE            NR_PEDIDO
--------------------- ----------
Antonio                       10
Antonio                       11
Beatriz                       12
```

EXEMPLO 1.B – SINTAXE ANSI

```
SELECT nm_cliente, nr_pedido
FROM cliente
INNER JOIN pedido
ON cliente.cd_cliente = pedido.cd_cliente;
```

```
NM_CLIENTE            NR_PEDIDO
--------------------- ----------
Antonio                       10
Antonio                       11
Beatriz                       12
```

EXEMPLO 2.A

Utilizando aliases para tabelas

```
SELECT nm_cliente, nr_pedido
FROM cliente c, pedido p
WHERE c.cd_cliente = p.cd_cliente;
```

```
NM_CLIENTE            NR_PEDIDO
--------------------- ----------
Antonio                       10
Antonio                       11
Beatriz                       12
```

EXEMPLO 2.B – SINTAXE ANSI

Utilizando aliases para tabelas

```
SELECT nm_cliente, nr_pedido
FROM cliente c
INNER JOIN pedido p
ON c.cd_cliente = p.cd_cliente;
```

```
NM_CLIENTE            NR_PEDIDO
--------------------- ----------
Antonio                       10
Antonio                       11
Beatriz                       12
```

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)

EXEMPLO

Apresentar os nomes (nm_cliente) de todos os clientes e para aqueles que realizaram alguma compra, apresentar o número do pedido (nr_pedido):

```
SELECT nm_cliente, nr_pedido
FROM cliente c, pedido p
WHERE c.cd_cliente = p.cd_cliente(+);
```

```
NM_CLIENTE            NR_PEDIDO
-------------------- ----------
Antonio                      10
Antonio                      11
Beatriz                      12
Claudio
```

NOTA: O operador (+) está junto ao campo em cuja tabela "faltam" registros.

```
SELECT nm_cliente, nr_pedido
FROM cliente c
LEFT OUTER JOIN pedido p
ON c.cd_cliente = p.cd_cliente;
```

```
NM_CLIENTE            NR_PEDIDO
-------------------- ----------
Antonio                      10
Antonio                      11
Beatriz                      12
Claudio
```

RIGHT OUTER JOIN (JUNÇÃO EXTERNA À DIREITA)

EXEMPLO

Apresentar os números de todos os pedidos (nr_pedido) e os nomes (nm_cliente) dos respectivos clientes:

```
SELECT nm_cliente, nr_pedido
FROM cliente c, pedido p
WHERE c.cd_cliente(+) = p.cd_cliente;
```

```
NM_CLIENTE            NR_PEDIDO
-------------------- ----------
Antonio                      10
Antonio                      11
Beatriz                      12
                             13
```

```
SELECT nm_cliente, nr_pedido
FROM cliente c
REGHT OUTER JOIN pedido p
ON c.cd_cliente = p.cd_cliente;
```

```
NM_CLIENTE            NR_PEDIDO
-------------------- ----------
Antonio                      10
Antonio                      11
Beatriz                      12
Claudio
```

FULL OUTER JOIN (JUNÇÃO EXTERNA COMPLETA)

EXEMPLO

Apresentar os nomes (nm_cliente) de todos os clientes e todos os números dos pedidos (nr_pedido) emitidos:

```
SELECT nm_cliente, nr_pedido
FROM cliente c
FULL OUTER JOIN pedido p
ON c.cd_cliente = p.cd_cliente;

NM_CLIENTE            NR_PEDIDO
-------------------- ----------
Antonio                      10
Antonio                      11
Beatriz                      12
Claudio
```

NON EQUI JOIN

Reúne campos de tabelas que não têm nada em comum.

EXEMPLO

```
CREATE TABLE faixa
(classe varchar2(10),
 minimo number(7,2),
 maximo number(7,2));

INSERT INTO faixa VALUES ('barato',0,24.99);
INSERT INTO faixa VALUES ('médio',25.00,49.99);
INSERT INTO faixa VALUES ('caro',50.00,74.99);

SELECT cd_produto, nm_produto, vl_produto, classe
FROM produto p, faixa f
WHERE p.vl_produto BETWEEN f.minimo AND f.maximo;

CD_PRODUTO NM_PRODUTO                    VL_PRODUTO CLASSE
---------- ------------------------------ ---------- ----------
       101 produto1                            10,52 barato
       102 produto2                            50,75 caro
       103 produto3                            45,34 médio
```

Observe o operador relacional BETWEEN.

Pode-se utilizar qualquer operador relacional, exceto = (igual).

```
SELECT cd_produto, nm_produto, vl_produto, classe
FROM produto p
INNER JOIN faixa f
ON p.vl_produto BETWEEN f.minimo AND f.maximo;

CD_PRODUTO NM_PRODUTO                    VL_PRODUTO CLASSE
---------- ------------------------------ ---------- ----------
       101 produto1                            10,52 barato
       102 produto2                            50,75 caro
       103 produto3                            45,34 médio
```

SELF JOIN

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

EXEMPLO

```
CREATE TABLE curso
(cd_curso  number(4),
 nm_curso  varchar2(25),
 cd_prereq number(4));

INSERT INTO curso VALUES (0001,'Windows Básico','');
INSERT INTO curso VALUES (0002,'Windows Avançado',0001);
INSERT INTO curso VALUES (0003,'Word Básico','');
INSERT INTO curso VALUES (0004,'Word Avançado',0003);
INSERT INTO curso VALUES (0005,'Excel Completo','');

SELECT c.nm_curso, p.nm_curso "PRÉ-REQUISITO"
FROM curso c, curso p
WHERE c.cd_prereq = p.cd_curso;

NM_CURSO                  PRÉ-REQUISITO
------------------------- -------------------------
Windows Avançado          Windows Básico
Word Avançado             Word Básico
```

O exemplo acima mostrou o pré-requisito de cada curso que possui um.

```
SELECT c.nm_curso, p.nm_curso "PRÉ-REQUISITO"
FROM curso c
INNER JOIN curso p
ON c.cd_prereq = p.cd_curso;

NM_CURSO                  PRÉ-REQUISITO
------------------------- -------------------------
Windows Avançado          Windows Básico
Word Avançado             Word Básico
```

CROSS JOIN (JUNÇÃO CRUZADA)

Apresenta todas as combinações possíveis entre elementos de duas tabelas.

EXEMPLO

```
CREATE TABLE a
(nm_time varchar2(15));

CREATE TABLE b
(nm_time varchar2(15));

INSERT INTO a VALUES ('Corinthians');
INSERT INTO a VALUES ('Palmeiras');
INSERT INTO a VALUES ('Santos','');
INSERT INTO a VALUES ('São Paulo');

INSERT INTO b VALUES ('Bota Fogo');
INSERT INTO b VALUES ('Flamengo');
INSERT INTO b VALUES ('Fluminense');
INSERT INTO b VALUES ('Vasco');
```

```
SELECT a.nm_time, b.nm_time
FROM a, b;
```

```
NM_TIME         NM_TIME
--------------- -------------------------
Corinthians     Bota Fogo
Corinthians     Flamengo
Corinthians     Fluminense
Corinthians     Vasco
Palmeiras       Bota Fogo
Palmeiras       Flamengo
Palmeiras       Fluminense
Palmeiras       Vasco
Santos          Bota Fogo
Santos          Flamengo
Santos          Fluminense
Santos          Vasco
São Paulo       Bota Fogo
São Paulo       Flamengo
São Paulo       Fluminense
São Paulo       Vasco
```

```
SELECT a.nm_time, b.nm_time
FROM a
CROSS JOIN b;
```

```
NM_TIME         NM_TIME
--------------- -------------------------
Corinthians     Bota Fogo
Corinthians     Flamengo
Corinthians     Fluminense
Corinthians     Vasco
Palmeiras       Bota Fogo
Palmeiras       Flamengo
Palmeiras       Fluminense
Palmeiras       Vasco
Santos          Bota Fogo
Santos          Flamengo
Santos          Fluminense
Santos          Vasco
São Paulo       Bota Fogo
São Paulo       Flamengo
São Paulo       Fluminense
São Paulo       Vasco
```

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.

EXEMPLO

```
SELECT nm_cliente, nr_pedido
FROM cliente
NATURAL INNER JOIN pedido;
```

```
NM_CLIENTE            NR_PEDIDO
-------------------- ----------
Antonio                      10
Antonio                      11
Beatriz                      12
```

JUNÇÃO BASEADA EM NOMES DE COLUNAS

Principal diferença entre a junção natural e a baseada em nomes de colunas:

JUNÇÃO NATURAL: todas as colunas de mesmo nome nas tabelas serão utilizadas para realizar a junção.

JUNÇÃO BASEADA EM NOMES DE COLUNAS: somente serão utilizadas as colunas listadas na cláusula USING.

EXEMPLO

```
SELECT nm_cliente, nr_pedido
FROM cliente
INNER JOIN pedido
USING (cd_cliente);
```

```
NM_CLIENTE            NR_PEDIDO
--------------------- ----------
Antonio                       10
Antonio                       11
Beatriz                       12
```

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 coluna1, coluna2
   FROM tabela1
UNION [ALL]
SELECT coluna3, coluna4
   FROM tabela2;
```

- 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 coluna1, coluna2
   FROM tabela1
INTERSECT
SELECT coluna3, coluna4
   FROM tabela2;
```

MINUS (DIFERENÇA)

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

A ordem de declaração das consultas com relação ao predicado MINUS altera o resultado final.

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

```
SELECT coluna1, coluna2
   FROM tabela1
MINUS
SELECT coluna3, coluna4
   FROM tabela2;
```

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

EXEMPLOS

CODIGO	NOME	ESTADO
1	ANTONIO	SP
2	ANTONIO	RJ
3	ANTONIO	RS

```
SELECT * FROM CLIENTE
   WHERE ESTADO='SP'
UNION
SELECT * FROM CLIENTE
   WHERE ESTADO='RJ';

CODIGO NOME     ESTADO
------ -------- ------
1      ANTONIO  SP
2      ANTONIO  RJ
```

```
SELECT * FROM CLIENTE
   WHERE NOME='ANTONIO'
INTERSECT
SELECT * FROM CLIENTE
   WHERE ESTADO='SP';

CODIGO NOME     ESTADO
------ -------- ------
1      ANTONIO  SP
```

```
SELECT * FROM CLIENTE
   WHERE NOME='ANTONIO'
MINUS
SELECT * FROM CLIENTE
   WHERE ESTADO='RJ';

CODIGO NOME     ESTADO
------ -------- ------
1      ANTONIO  SP
3      ANTONIO  RS
```