ORGANIZAÇÃO EM BANCO DE DADOS

Prof. Msc. Marcos Alexandruk

alexandruk@uni9.pro.br

https://github.com/alexandruk/organizacaoembancodedados

Apresentação da Disciplina

Organização em Banco de Dados

Procedural Language / Structured Query Language:

SQL - Linguagem de Definição de Dados (DDL)
SQL - Linguagem de Manipulação de Dados (DML)
SQL - Linguagem de Consulta de dados (DQL)

Tipos de dados
Constantes e variáveis
Estrutura básica dos blocos PL/SQL
Instruções SQL dentro dos blocos PL/SQL
Estruturas de decisão
Estruturas de repetição
Cursores explícitos e implícitos
Procedures
Functions
Triggers

Administração de Banco de Dados

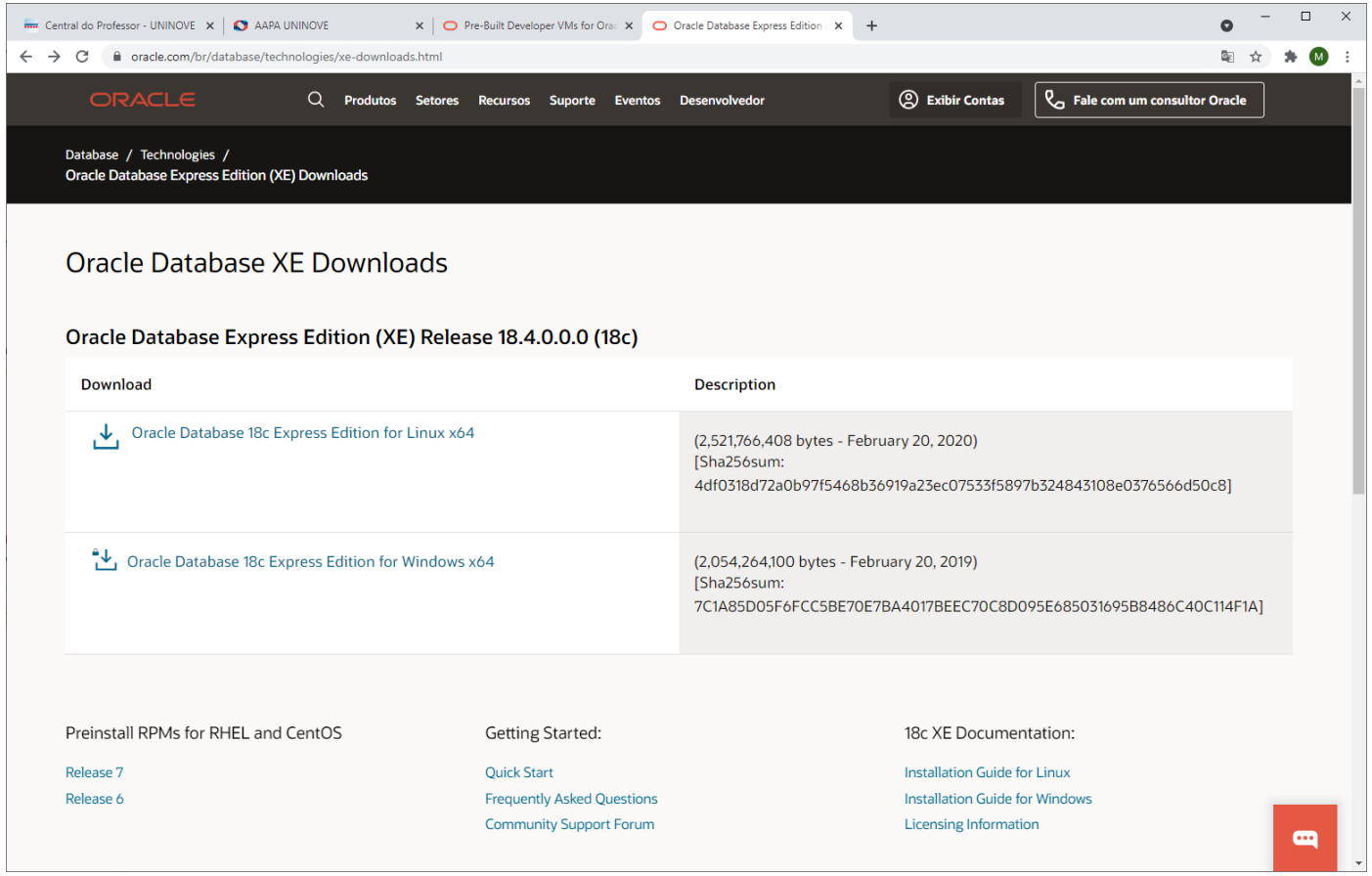
Funções do DBA (database administrator) e mercado de trabalho
Arquitetura do banco de dados - estrutura física:
- arquivos, memória e processos de segundo plano
Arquitetura do banco de dados - estrutura lógica:
- instâncias, tablespaces, blocos, extensões e segmentos
Arquitetura do banco de dados - estrutura lógica
- tabelas e constraints
Arquitetura do banco de dados - estrutura lógica:
- índices e visões
Gerenciamento de instância
Gerenciamento da estrutura física:
- arquivos, memória e processos de segundo plano
Gerenciamento da estrutura lógica:
- tablespaces, datafiles, segmentos, extensões e blocos
Gerenciamento de usuários (schemas)
Gerenciamento de perfis de usuários
Gerenciamento de privilégios de sistema
Gerenciamento de privilégios de objetos
Gerenciamento de privilégios de grupos (roles)
Gerenciamento de índices
Backup e recovery lógico (exportação e importação)
Backup físico: offline
Backup físico: online
Backup utilizando o RMAN (Recovery Manager)

Oracle 18c XE

https://www.oracle.com/br/database/technologies/appdev/xe.html

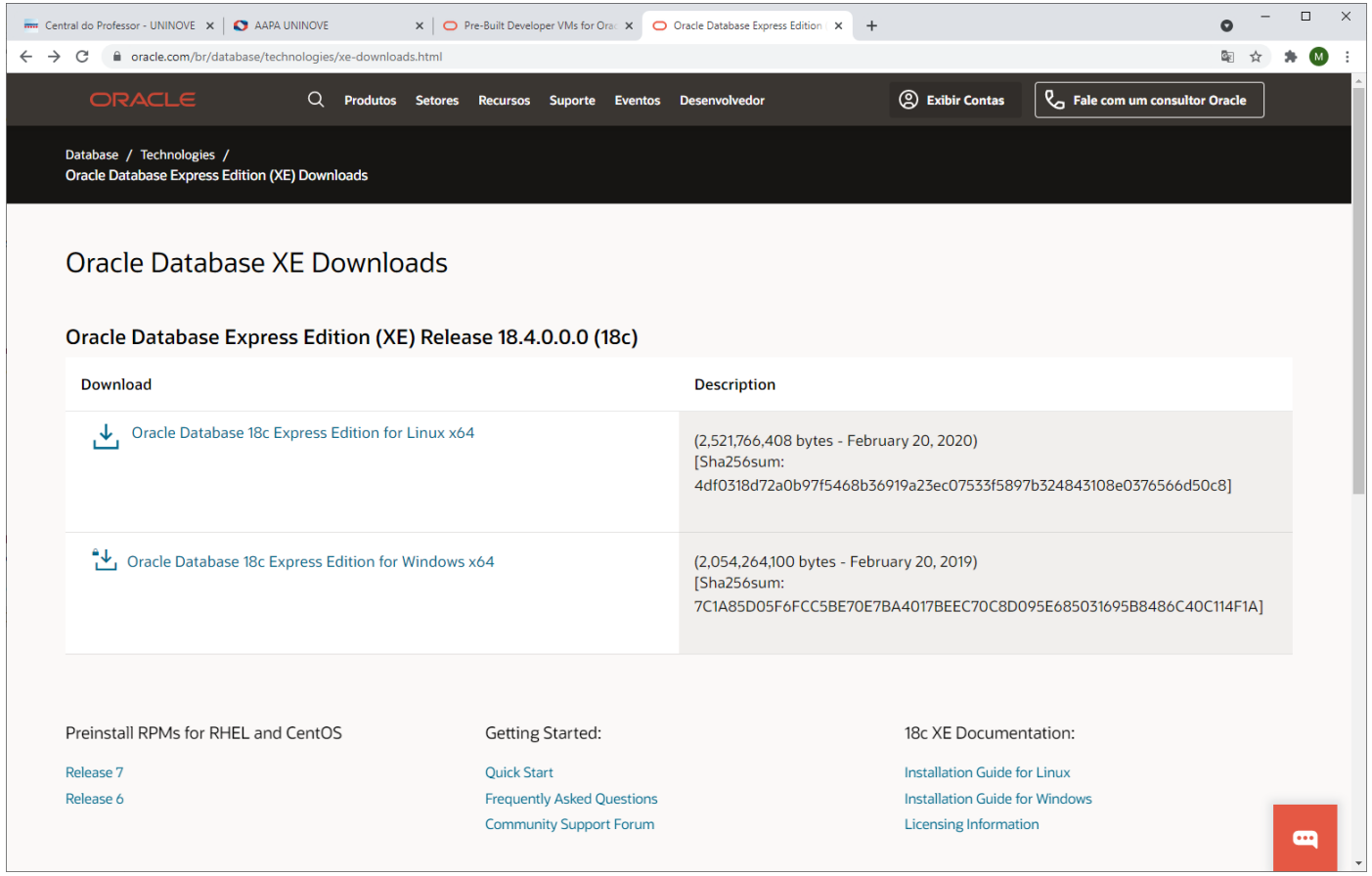
Oracle 18c XE

https://www.oracle.com/br/database/technologies/xe-downloads.html

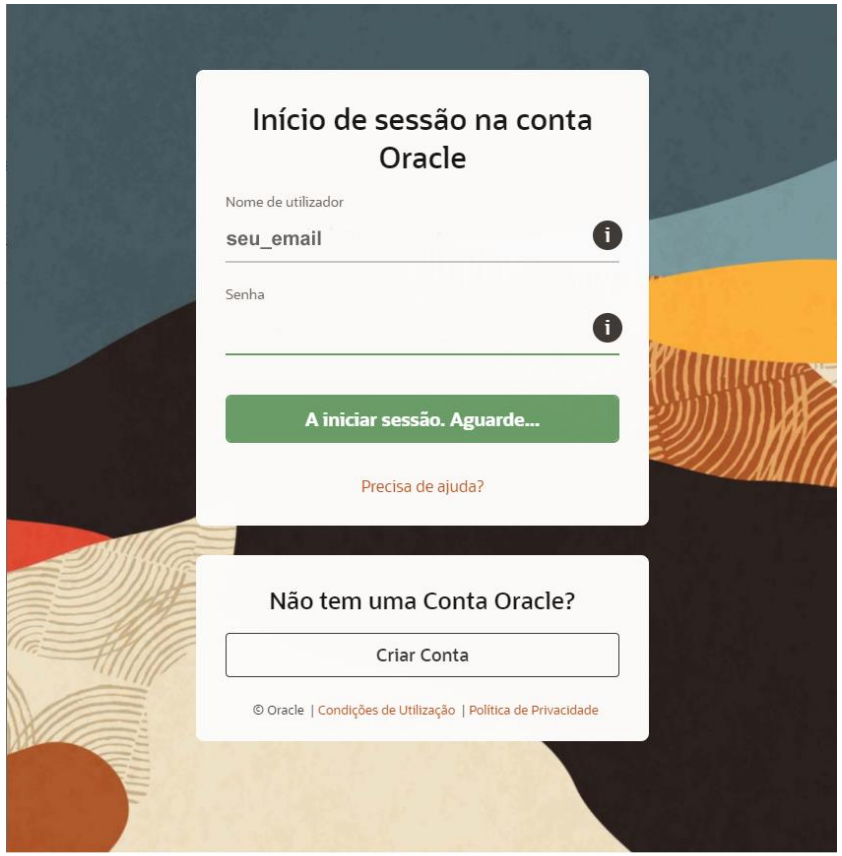


Oracle 18c XE

https://www.oracle.com/br/database/technologies/xe-downloads.html



Cadastro Oracle



Para realizar o Download:

1. Caso já tenha um cadastro na Oracle:

Informar o e-mail e a senha

2. Caso NÃO tenha um cadastro na Oracle:

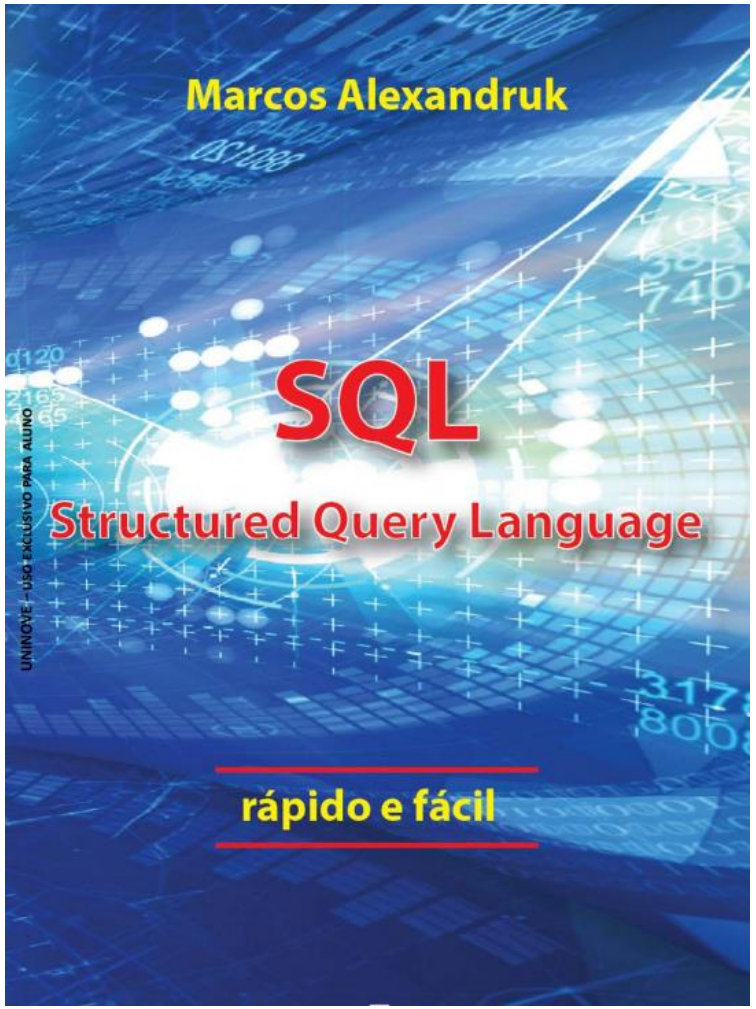
Clicar em Criar Conta

Preencher os dados solicitados (com e-mail válido)

Aguardar o e-mail de confirmação da Oracle

Fazer o login (conforme opção 1 acima)

SQL (Structured Query Language) rápido e fácil



Referência para os tópicos:

SQL - Linguagem de Definição de Dados (DDL)
SQL - Linguagem de Manipulação de Dados (DML)
SQL - Linguagem de Consulta de dados (DQL)

Disponível na Biblioteca Virtual e no GitHub

Instalação e configuração do ambiente

Oracle 11g Express Edition (32 bits)

1. Download do Software (o link será informado pelo professor)
2. Instalação
    - Atenção à senha dos usuários **SYS** e **SYSDBA** (durante a instalação)
    - Visto que não é um banco de produção, recomendo usar a senha **oracle**
3. Iniciar o serviço: Clicar na opção **Iniciar Banco de Dados**
    - (Aguardar alguns segundos até que o serviço seja iniciado)
4. Clicar em **Executar Linha de Comando SQL**
    - Este procedimento inicializa o SQL*Plus
    - SQL *Plus é um ambiente linha de comando onde você executará as instruções SQL
5. Logar no Oracle como usuário SYSTEM
    - Digite: **conn system** e pressione a tecla Enter
    - Digite a senha **oracle** (ou outra escolhida por você) e pressione a tacla Enter

Oracle 11g Express Edition (32 bits)

1. Criar o usuário aluno (este usuário será utilizado como padrão para nossas próximas aulas)
   - Digite: **create user aluno identified by oracle;**
   - (Este comando cria um usuário denominado aluno com senha oracle.)
2. Conceder privilégio connect para o usuário aluno:
   - Digite: **grant connect to aluno;**
3. Conceder privilégio resource para o usuário aluno:
   - Digite: **grant resource to aluno;**
4. Conectar-se como usuário aluno:
   - Digite: **conn aluno** e pressione a tecla Enter
   - Informe a senha **oracle** e pressione a tecla Enter

Os novos objetos de banco de dados (tables, views, etc.) serão criados usando este usuário (aluno).

SQL - Structured Query Language

Conteúdo Programático

- Bancos de Dados Relacionais
- Conceitos gerais da SQL (Structured Query Language)
- Principais SGBDR's (Sistemas de Gerenciamento de Bancos de Dados Relacionais)
- Tipos de dados
- Restrições (Constraints)
- Usando instruções DDL para criar, alterar e eliminar tabelas
- Usando instruções DML para inserir, atualizar e excluir dados
- Usando operadores para restringir e classificar dados
- Usando funções de linha, de conversão e de grupo
- Usando *subqueries* para realizar consultas em múltiplas tabelas
- Usando *joins* para exibir dados de várias tabelas
- Usando *merge* para mesclar dados de várias tabelas
- Usando os operadores de conjunto: UNION, MINUS e INTERSECT
- Criando e eliminando views
- Criando e eliminando indexes

Bancos de Dados Relacionais

Breve história dos bancos de dados relacionais e da SQL:

1970 – Edgar Frank Codd (IBM) publica o artigo "A Relational Model of Data for Large Shared Data Banks".

1970 – Donald D. Chamberlin e Raymond F. Boyce (IBM) apresentam uma linguagem, baseada na álgebra relacional, denominada SEQUEL (Structured English Query Language). O nome foi alterado, posteriormente, para SQL (Structures Query Language).

1979 – A Relational Software Inc (atual Oracle) apresentou ao mercado o primeiro produto comercial com base na SQL (Oracle v2).

1986 – A ANSI (American National Standards Institute) e a ISO (International Organization for Standardization) passaram a "padronizar" a SQL.

Bancos de Dados Relacionais

O DB-Engines Ranking classifica mensalmente os sistemas de gerenciamento de banco de dados de acordo com sua popularidade. Abaixo o ranking em janeiro/2020:

1º Oracle [R]	**11º** Cassandra [C]	**21º** Hbase [C]
2º MySQL [R]	**12º** Splunk [SE]	**22º** Neo4j [G]
3º MS SQL Server [R]	**13º** MariaDB [R]	**23º** Couchbase [D]
4º PostgreSQL [R]	**14º** Hive [R]	**24º** MS Azure Cosmos DB [C|D|KV|G]
5º MongoDB [D]	**15º** Teradata [R]	**25º** MS Azure SQL [R]
6º IBM DB2 [R]	**16º** Amazon DynamoDB [D|KV]	**26º** Google BigQuery [R]
7º Elasticsearch [SE]	**17º** Solr [SE]	**27º** Informix [R]
8º Redis [KV]	**18º** FileMaker [R]	**28º** Memcached [KV]
9º MS Access [R]	**19º** SAP HANA [R]	**29º** Vertica [R]
10º SQLite [R]	**20º** SAP Adaptive Server [R]	**30º** Firebird [R]

Database Model: [R] Relational | [C] Column | [D] Document | [KV] Key-Value | [G] Graph | [SE] Search Engine

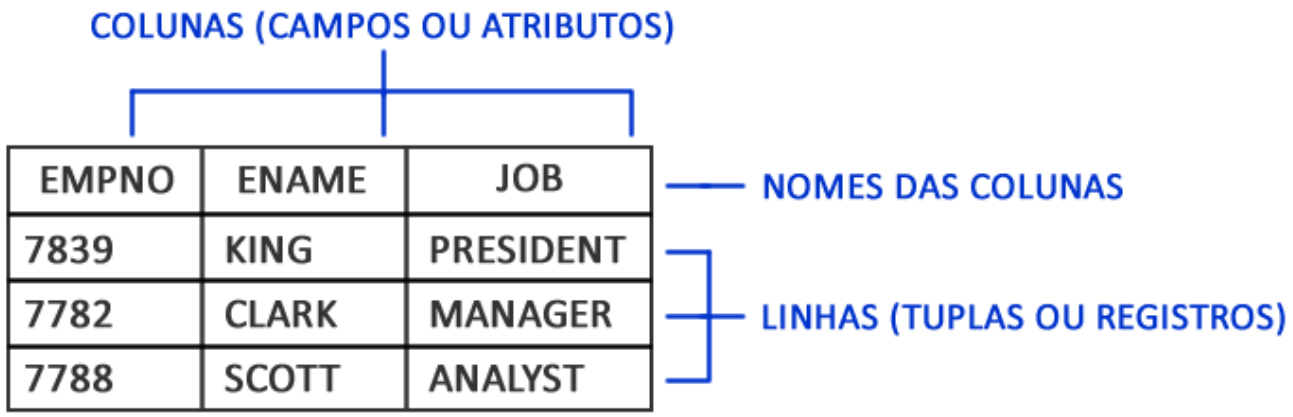
https://db-engines.com/en/ranking

Bancos de Dados Relacionais

A estrutura básica dos Bancos de Dados Relacionais é denominada **tabela** (table) ou "relações".

Uma tabela é composta por uma estrutura bidimensional formada por linhas (dependendo do contexto também chamadas de registros ou tuplas) e colunas (também chamadas de campos ou atributos).

Apresenta, portanto, uma estrutura similar às planilhas de aplicativos como o MS Excel.



Subgrupos da SQL

A SQL (Structured Query Languagem) apresenta os seguintes subgrupos:

DDL (Data Definition Language)

DML (Data Manipulation Language)

DTL (Data Transact Language)

DCL (Data Control Language)

*Alguns autores incluem um quinto grupo, denominado **DQL** (Data Query Language), que inclui basicamente o comando SELECT. Outros preferem relacionar este comando no subgrupo DML (Data Manipulation Language).*

Principais comandos DDL (Data Definition Language):

A seguir os principais comandos do subgrupo denominado DDL:

CREATE: Usado para criar novos objetos (tabelas, visões, etc.) no banco de dados

ALTER: Usado para alterar as estruturas dos objetos do banco de dados

DROP: Usado para eliminar os objetos do banco de dados

TRUNCATE: Usado para eliminar permanentemente os dados inseridos em objetos

Principais comandos DML (Data Manipulation Language):

A seguir os principais comandos do subgrupo denominado DML:

INSERT: Usado para inserir dados em objetos (tabelas, visões, etc.) do banco de dados

UPDATE: Usado para atualizar dados em objetos do banco de dados

DELETE: Usado para excluir dados em objetos do banco de dados

SELECT: Usado para apresentar os dados inseridos em objetos do banco de dados

Principais comandos DTL (Data Transact Language):

A seguir os principais comandos do subgrupo denominado DTL:

COMMIT: Usado para gravar no banco de dados as instruções SQL de uma transação

ROLLBACK: Usado para descartar as instruções SQL de uma transação

SAVEPOINT: Usado para decompor transações muito longas, registrando "pontos de salvamento". Caso ocorram erros a transação não precisa ser revertida até o início.

TRANSAÇÃO: *Conjunto de instruções SQL que deve ser executado na sua totalidade.*

Principais comandos DCL (Data Control Language):

A seguir os principais comandos do subgrupo denominado DCL:

GRANT: Usado para conceder um privilégio de sistema ou de objeto a um usuário

REVOKE: Usado para revogar um privilégio de sistema ou de objeto de um usuário

Alguns privilégios de sistema: CREATE TABLE, CREATE VIEW, DROP TABLE, DROP VIEW

Alguns privilégios de objeto: SELECT, INSERT, UPDATE, DELETE

Tipos de Dados

Um banco de dados relacional armazena uma **coleção de dados organizado que se relacionam**. Portanto, um cuidado especial necessário é a escolha do tipo correto de dados de acordo com o que será armazenado em cada coluna da tabela.

Embora a ANSI e a ISO tenham padronizado a SQL, há algumas pequenas variações na nomenclatura dos tipos de dados adotadas por diferentes SGBD's.

A seguir será apresentada a nomenclatura adotada no SGBD Oracle Database (que ocupa a primeira posição no DB-Engines Ranking).

NUMBER (p,s)

Armazena valores numéricos com precisão **p** e escala **s**. A precisão **p** representa o número total de dígitos e a escala **s** representa o total e dígitos à direita do **.** (ponto) ou da **,** (vírgula) dependendo do sistemas utilizado (inglês ou internacional).

A precisão **p** varia de 1 a 38 e a escala **s** varia de -84 a 127.

Exemplo:

NUMBER (4): Armazena números inteiros com até quatro dígitos. Ex: 5698

NUMBER (5,2): Armazena números com até cinco dígitos e com duas casas decimais Ex: 362.45

CHAR (n)

Armazena caracteres alfanuméricos com **tamanho fixo**. Tamanho mínimo 1 (default) e máximo 2000 caracteres.

Exemplo:

CHAR (2): Para armazenar siglas de UF's (Estados): SP

VARCHAR2 (n)

Armazena caracteres alfanuméricos com **tamanho variável**. Tamanho mínimo 1 e máximo 4000 caracteres.

Exemplo:

VARCHAR(50): Para armazenar nomes de cidades: Rio de Janeiro

Observação: *A maioria dos SGBD's utilizam a denominação* ***VARCHAR*** *para armazenar estes tipos de dados.*

DATE

Armazena valores referentes a data e horários. (Datas entre 01/01/4712 a.C. a 31/12/9999 d.C.)

TIMESTAMP

Armazena, além de dia, mês e ano, hora, minuto, segundo e frações de segundo.

Por que 01/01/4712 a.C.?
Porque o Dia Juliano começa em 01/01/4712 a.C.
Quem introduziu o Dia Juliano, foi o filólogo francês Giuseppe Giusto Scaliger (1540-1609), e assim o denominou em homenagem a seu pai, o humanista italiano Julius Caesar Scaliger (1484-1558).
Em astronomia, com aplicações na cronologia de acontecimentos históricos, o dia juliano é um método de contar os dias sequencialmente, começando em uma data arbitrária no passado.
Obter o Dia Juliano no Oracle: **SELECT TO_CHAR(TO_DATE(TO_CHAR(1), 'J'),'DD/MM/YYYY') FROM DUAL;**
Contagem de dias desde 01/01/4712 a.C.: **SELECT TO_CHAR(SYSDATE, 'J') FROM DUAL;**
O tipo de dados DATE da Oracle tem o tamanho máximo de 7 bytes.

RAW

Armazena dados binários com tamanho máximo igual a 2000 bytes.

LONG RAW

Armazena dados binários com tamanho máximo igual a 2 GB.

BLOB

Armazena dados binários com tamanho máximo igual a 4 GB.

BFILE

Armazena dados binários em um arquivo externo com tamanho máximo igual a 4 GB.

ROWID

Armazena dados em um sistema numérico de base 64 para representar o endereço exclusivo de uma linha de tabela.

Constraints

PRIMARY KEY: Usada para identificar com exclusividade cada linha da tabela. A Primary Key (Chave Primária) pode ser simples (abrangendo apenas uma coluna) ou composta (abrangendo mais de uma coluna). Cada tabela pode ter apenas uma Chave Primária (simples ou composta).

FOREIGN KEY: Usada para implementar, através e campos comuns, o relacionamento entre tabelas. Uma Foreign Key (Chave Estrangeira) sempre referenciará uma Primary Key (Chave Primária) de outra ou da mesma tabela (autorrelacionamento).

UNIQUE: Usada para garantir que apenas valores exclusivos sejam inseridos na coluna que recebe esta restrição. Porém, diferentemente da Primary Key (Chave Primária) não controla a ausência de valores (NULL).

CHECK: Usada para garantir que o valor que será inserido no campo da tabela corresponde a um conjunto de valores (domínio) previamente determinado.

NOT NULL: Usada para garantir a inserção de valores no campo que recebe esta constraint.

DEFAULT: *Não é uma constraint, pois não impõe nenhuma restrição à tabela. Usado para inserir automaticamente um valor "default" (padrão) no campo da tabela caso nenhum valor seja informado.*

DDL - Data Definition Language

Criação de tabelas

O exemplo abaixo apresenta o código SQL para criação de uma tabela denominada **CLIENTE** com suas respectivas constraints:

```
CREATE TABLE CLIENTE (
ID NUMBER (4),
NOME VARCHAR2 (50) NOT NULL,
CPF CHAR (11),
DATA_NASC DATE,
UF CHAR (2),
CONSTRAINT CLIENTE_PK PRIMARY KEY (ID),
CONSTRAINT CLIENTE_UN UNIQUE (CPF),
CONSTRAINT CLIENTE_CK CHECK (UF IN ('SP','RJ','MG'))
);
```

Nota: Código compatível com o Oracle Database. Outros SGBD's não utilizam, por exemplo, o tipo de dado VARCHAR2.

Criação de tabelas

O exemplo abaixo apresenta o código SQL para criação de uma tabela denominada **PEDIDO** com suas respectivas constraints.

A tabela **PEDIDO** relaciona-se com a tabela **CLIENTE**, criada anteriormente, através da **Foreign Key** (Chave Estrangeira) **PEDIDO_CLIENTE_FK**.

```
CREATE TABLE PEDIDO (
NR NUMBER (5),
DATA_PEDIDO DATE DEFAULT SYSDATE,
ID NUMBER (4),
CONSTRAINT PEDIDO_PK PRIMARY KEY (NR),
CONSTRAINT PEDIDO_CLIENTE_FK FOREIGN KEY (ID) REFERENCES CLIENTE (ID)
);
```

Nota: A cláusula DEFAULT determina que seja inserida a data atual do sistema (SYSDATE), caso não seja informada nenhuma data quando um novo registro (linha) for incluído na tabela.

Alteração de tabelas

O comando **ALTER TABLE** é usado para alterar uma tabela anteriormente criada. As principais alterações que podem ser realizadas são as seguintes:

- Adicionar uma ou mais colunas na tabela
- Modificar o tamanho de uma coluna
- Renomear uma coluna
- Eliminar uma coluna
- Adicionar constraints
- Eliminar constraints
- Desabilitar constraints
- Habilitar constraints

Alteração de tabelas: Adicionar uma coluna

Antes de realizar este tipo de alteração, deve-se definir o nome da nova coluna, o tipo dos dados que serão armazenados e, se necessário (dependendo do tipo dos dados) seu respectivo tamanho.

O exemplo abaixo apresenta o código SQL para adicionar uma coluna denominada **E_MAIL** na tabela **CLIENTE**.

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

Alteração de tabelas: Modificar o tamanho de uma coluna

Outra situação que um DBA (Database Administrator) pode encontrar é a necessidade de alterar o tamanho de uma coluna.

O exemplo abaixo apresenta o código SQL para alterar o tamanho de uma coluna denominada **E_MAIL** localizada na tabela **CLIENTE** para 60 caracteres.

```
ALTER TABLE CLIENTE
MODIFY E_MAIL VARCHAR2 (60);
```

Alteração de tabelas: Renomear uma coluna

Há situações não muito frequentes em que é necessário alterar o nome de uma coluna. Por exemplo, quando seu nome foi incorretamente digitado.

O exemplo abaixo apresenta o código SQL para alterar o nome de uma coluna denominada **E_MAIL** localizada na tabela **CLIENTE** para **EMAIL**.

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

Alteração de tabelas: Eliminar uma coluna

Outro tipo de alteração é a eliminação de colunas de tabelas. Este tipo de operação deve ser feito com muito cuidado, pois os dados contidos na coluna eliminada também deixarão de existir no banco.

O exemplo abaixo apresenta o código SQL para eliminar uma coluna denominada **EMAIL** localizada na tabela **CLIENTE**.

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

Alteração de tabelas: Adicionar constraints

O exemplo abaixo apresenta o código SQL para adicionar uma constraint **PRIMARY KEY**:

```
ALTER TABLE CLIENTE
ADD CONSTRAINT CLIENTE_PK PRIMARY KEY (ID);
```

O exemplo abaixo apresenta o código SQL para adicionar uma constraint **FOREIGN KEY**:

```
ALTER TABLE PEDIDO
ADD CONSTRAINT PEDIDO_CLIENTE_FK FOREIGN KEY ID REFERENCES CLIENTE (ID);
```

O exemplo abaixo apresenta o código SQL para adicionar uma constraint **NOT NULL**:

```
ALTER TABLE CLIENTE
MODIFY NOME CONSTRAINT NOME_NN NOT NULL;
```

NOTA: *A constraint* ***NOT NULL*** *não pode ser declarada no modo OUT OF LINE. Por isso, não deve ser utilizada a instrução* ***ADD*** *(adicionar). Usa-se, portanto, a instrução* ***MODIFY****, conforme demonstrado acima.*

Alteração de tabelas: Desabilitar e habilitar constraints

Desabilitar uma constraint indica que ela não atuará sobre as operações a serem realizadas no banco, porém ela continuará a existir e poderá ser novamente habilitada.

Deve-se tomar muito cuidado ao inserir, alterar ou excluir dados enquanto uma constraint estiver desabilitada, pois qualquer uma destas operações poderá causar inconsistências no banco de dados.

O exemplo a seguir apresenta o código SQL para **desabilitar** a constraint **CLIENTE_PK**:

```
ALTER TABLE CLIENTE
DISABLE CONSTRAINT CLIENTE_PK;
```

O exemplo a seguir apresenta o código SQL para **habilitar** novamente a constraint **CLIENTE_PK**:

```
ALTER TABLE CLIENTE
ENABLE CONSTRAINT CLIENTE_PK;
```

Alteração de tabelas: Eliminar constraints

Caso seja necessário eliminar uma constraint deve-se usar o comando DROP CONSTRAINT.

Esta alteração deve ser realizada com extrema cautela para não gerar inconsistências no banco de dados.

O exemplo a seguir apresenta o código SQL para **eliminar** a constraint **CLIENTE_PK**:

```
ALTER TABLE CLIENTE
DROP CONSTRAINT CLIENTE_PK;
```

Eliminação de tabelas

Tabelas que perderam sua utilidade ou que já cumpriram sua função temporária poderão ser excluídas do banco de dados.

O exemplo a seguir apresenta o código SQL para **eliminar** uma tabela denomina **CLIENTE**:

```
DROP TABLE CLIENTE
CASCADE CONSTRAINTS;
```

NOTA: *A cláusula* ***CASCADE CONSTRAINTS*** *é utilizada para eliminar as constraints associadas à tabela CLIENTE.*

Renomear tabelas

Visto que a estrutura de um banco de dados está sujeita a alterações, às vezes torna-se necessário renomear algumas tabelas.

O exemplo a seguir apresenta o código SQL para **renomear** uma tabela de **CLIENTE** para **CLIENTES**.

```
RENAME CLIENTE TO CLIENTES;
```

"Cortar" ou "truncar" tabelas

O comando **TRUNCATE** mantém a estrutura (nome, colunas, tipos de dados, etc.), mas elimina definitivamente todas as linhas da tabela.

O exemplo a seguir apresenta o código SQL para ***"truncar"*** uma tabela denominada **CLIENTE**.

```
TRUNCATE TABLE CLIENTE;
```

NOTA: *Após a execução do comando acima a tentativa de reverter a exclusão dos dados através do comando ROLLBACK não terá efeito. TRUNCATE faz parte do subgrupo DDL. Os comandos que fazem parte deste subgrupo não podem ser revertidos através de ROLLBACK.*

DML - Data Manipulation Language

SELECT - Consultando dados

O comando SELECT é usado para realizar consultas em tabelas ou visões. Através das consultas obtém-se todos os dados de uma tabela ou, através de filtros, apenas os desejados.

É necessário informar os nomes das colunas onde os dados estão localizados e o nome da tabela ou das tabelas no caso de subqueries e joins.

A consulta abaixo retorna dados contidos na coluna **NOME** da tabela **CLIENTE**.

```
SELECT NOME FROM CLIENTE;
```

A próxima consulta utiliza o caractere "coringa" ***** para retornar os dados contidos em **todas as colunas** da tabela **CLIENTE**.

```
SELECT * FROM CLIENTE;
```

INSERT - Inserindo dados

O comando INSERT é usado para inserir ou incluir novos dados em tabelas ou visões.

É necessário informar o nome da tabela, das colunas (opcional em alguns casos) e os valores a serem inseridos em cada linha da tabela.

Os nomes das colunas e os dados a serem inseridos são declarados dentro de parênteses e separados por vírgula.

A instrução abaixo insere **1001** e **'John Smith'**, respectivamente, nas colunas **ID** e **NOME** da tabela **CLIENTE**.

```
INSERT INTO CLIENTE (ID,NOME) VALUES (1001,'John Smith');
```

NOTA: *Valores numéricos são declarados sem o uso de aspas simples ' '. A maioria dos outros tipos de dados requerem o uso de aspas simples*.

DELETE - Excluindo dados

O comando DELETE é usado para excluir dados (linhas inteiras) de tabelas ou visões.

É necessário informar o nome da tabela e os valores da primary key das linhas a serem excluídas.

A instrução abaixo exclui da tabela **CLIENTE** a linha com ID é igual a **1001**.

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

É possível também eliminar em um único e simples comando todas as linhas da tabela.

A instrução abaixo exclui **todas as linhas** da tabela **CLIENTE**.

```
DELETE FROM CLIENTE;
```

NOTA: *Embora seja possível usar como referência outra coluna que não seja primary key para a eliminação de linhas da tabela, isto somente deve ser realizado quando não houver outra opção e com extrema cautela.*

UPDATE - Atualizando dados

O comando UPDATE é usado para atualizar dados em tabelas ou visões.

É necessário informar o nome da tabela, das colunas que terão os valores alterados e os novos valores. Cada linha ser atualizada deve ser referenciada através da primary key.

A instrução abaixo altera o **NOME** que corresponde ao **ID** igual a **1001** para '**Joseph Smith**'.

```
UPDATE CLIENTE SET NOME = 'Joseph Smith' WHERE ID = 1001;
```

É possível também atualizar com um único e simples comando todas as linhas da tabela.

A instrução abaixo (embora não tenha sentido) atualiza **todas as linhas** da tabela **CLIENTE**.

```
UPDATE CLIENTE SET NOME = 'Joseph Smith';
```

NOTA: *Embora seja possível usar como referência outra coluna que não seja primary key para a atualização de linhas da tabela, isto somente deve ser realizado quando não houver outra opção e com extrema cautela.*

Operadores

A cláusula WHERE antecede as condições que deverão ser verificadas para apresentação do resultado de uma determinada consulta. Para "filtrar" os dados são utilizados operadores.

A SQL apresenta vários tipos de operadores que podem ser agrupados conforme segue:

- Operadores de comparação
- Operadores lógicos
- Operadores SQL

Operadores de comparação

A SQL define seis operadores de comparação:

OPERADOR	DESCRIÇÃO
=	Igual
<	Menor
<=	menor ou igual
>	maior
>=	maior ou igual
<>	diferente

```
SELECT CODIGO FROM PRODUTO
WHERE VALOR_UNIT > 1000;
```

Operadores lógicos

A SQL define três operadores lógicos:

OPERADOR	DESCRIÇÃO
AND	Corresponde ao E lógico. Para que a expressão seja satisfeita as duas condições apresentadas devem ser VERDADEIRAS (TRUE).
OR	Corresponde ao OU lógico. Para que a expressão seja satisfeita apenas uma das condições apresentadas precisa ser VERDADEIRA (TRUE).
NOT	Operador lógico de negação.

```
SELECT NOME FROM CLIENTE
WHERE UF = 'SP' OR UF = 'RJ';
```

Operadores SQL

A SQL define diversos operadores específicos da linguagem:

OPERADOR	DESCRIÇÃO
IS NULL	Verifica se o conteúdo da coluna é NULL (valores ausentes)
IS NOT NULL	Negação do operador IS NULL
LIKE	Compara uma cadeia de caracteres utilizando padrões de comparação: % substitui zero, um ou mais caracteres _ substitui exatamente um caractere
NOT LIKE	Negação do operador LIKE
IN	Verifica se um valor específico pertence a um conjunto de valores
NOT IN	Negação do operador IN
BETWEEN	Determina um intervalo entre dois valores: BETWEEN 'valor1' AND 'valor2'
NOT BETWEEN	Negação do operador BETWEEN
EXISTS	Verifica se a consulta retorna (TRUE) ou não (FALSE) algum resultado
NOT EXISTS	Negação do operador EXISTS

Operadores SQL

```
SELECT ID FROM CLIENTE
WHERE NOME LIKE 'JOHN%';
```

```
SELECT ID FROM CLIENTE
WHERE NOME LIKE 'JO_ _';
```

```
SELECT NOME FROM CLIENTE
WHERE UF IN ('SP','RJ');
```

```
SELECT NOME FROM CLIENTE
WHERE ID BETWEEN 1001 AND 2000;
```

Alias

Alias ou apelidos são nomes alternativos, temporários e frequentemente abreviados para designar tabelas ou colunas de tabelas.

O uso de apelidos torna muito práticas as consultas, principalmente aquelas que envolvem mais de uma tabela.

Há situações em que o uso de apelidos é praticamente indispensável como no tipo de junção denominado SELF JOIN.

A consulta abaixo utiliza o alias **C** em substituição ao nome da tabela **CLIENTE** e **NOME_CLIENTE** em substituição ao nome da coluna **CLIENTE**.

Utiliza-se também **C.NOME** para indicar que esta coluna pertence a tabela **CLIENTE** (que recebeu o apelido **C**).

```
SELECT C.NOME NOME_CLIENTE FROM CLIENTE C;
```

A utilização de alias não causa nenhuma alteração nos nome originais, conforme registrados nas tabelas.

DISTINCT

A cláusula **DISTINCT** é usada em uma coluna quando é necessário omitir valores repetidos no resultado de uma consulta. Por exemplo, uma consulta na qual é preciso apresentar os nomes das cidades para as quais determinada empresa realizou vendas. É provável que muitas vendas tenham sido realizado para mesma cidade. Portanto, sem a cláusula **DISTINCT** o nome da mesma cidade apareceria no resultado da consulta muitas vezes.

A consulta abaixo utiliza a cláusula **DISTINCT** para que os nomes das cidades apareçam no resultado a consulta apenas uma vez.

```
SELECT DISTINCT CIDADE FROM CLIENTE;
```

ORDER BY

A cláusula **ORDER BY** é usada ordenar a saída dos dados, resultado de uma consulta, em ordem crescrente (default) ou decrescente.

A consulta abaixo utiliza a cláusula **ORDER BY** para que os nomes dos clientes apareçam em ordem crescente. Visto que este é o padrão não é opcional usar a abreviação **ASC**.

```
SELECT ID, NOME FROM CLIENTE ORDER BY NOME;
```

A consulta abaixo apresentará os nomes dos clientes apareçam em ordem decrescente. Neste caso e obrigatório utilizar a abreviação **DESC**.

```
SELECT ID, NOME FROM CLIENTE ORDER BY NOME DESC;
```

GROUP BY

A cláusula **GROUP BY** é usada quando é necessário agrupar dados, por exemplo, por filial, por cidade, por estado, etc.

Para realizar os agrupamentos são utilizadas as **funções de agrupamento**. Estas funções permitem que seja apresentada para cada agrupamento a quantidade, a soma, a média, etc.

A consulta abaixo apresentará **quantidade** de pedidos agrupados por **CIDADE**.

```
SELECT CIDADE, COUNT(NR) FROM PEDIDO GROUP BY CIDADE;
```

Cláusula HAVING

A cláusula **HAVING** é usada quando é necessário filtrar os dados agrupados da consulta de acordo com uma determinada condição.

A consulta abaixo apresentará **quantidade** de pedidos agrupados para a **CIDADE** de Miami.

```
SELECT CIDADE, COUNT(NR) FROM PEDIDO GROUP BY CIDADE HAVING CIDADE = 'MIAMI';
```

NOTA: *A cláusula **WHERE** usada para filtrar linhas não pode ser usada para filtrar agrupamentos, neste caso deve-se usar obrigatoriamente, como demonstrado, a cláusula **HAVING**.*

Functions (Funções)

Funções

A SQL apresenta funções nativas que podem ser agrupadas conforme segue:

- Funções de grupo ou de agregação
- Funções de linha
- Funções numéricas
- Funções de conversão
- Outras funções

Funções de grupo ou de agregação

Operam em grupos de linhas e retornam um valor para o grupo.

As funções de grupo ou agregação são as seguintes:

- SUM
- AVG
- MAX
- MIN
- COUNT
- MEDIAN
- STDDEV
- VARIANCE

SUM

A função **SUM** retorna o **total** por grupo das colunas selecionadas.

Exemplo:

```
Tabela: PEDIDO
------------------
NR_PEDIDO VALOR UF
--------- ----- --
        1  2500 SP
        2  1200 RJ
        3  1600 SP
        4  1800 RJ
```

O valor **total** dos pedidos por **UF** é obtido através da seguinte consulta:

```
SELECT UF, SUM(VALOR) FROM PEDIDO GROUP BY UF;

UF SUM(VALOR)
-- ----------
RJ        3000
SP        4100
```

AVG

A função **AVG** retorna a **média** por grupo das colunas selecionadas.

Exemplo:

```
Tabela: PEDIDO
------------------
NR_PEDIDO VALOR UF
--------- ----- --
        1  2500 SP
        2  1200 RJ
        3  1600 SP
        4  1800 RJ
```

O valor **médio** dos pedidos por **UF** é obtido através da seguinte consulta:

```
SELECT UF, AVG(VALOR) FROM PEDIDO GROUP BY UF;

UF MIN(VALOR)
-- ----------
RJ       1500
SP       2050
```

MAX

A função **MAX** retorna o **maior valor** (máximo) por grupo das colunas selecionadas.

Exemplo:

```
Tabela: PEDIDO
------------------
NR_PEDIDO VALOR UF
--------- ----- --
        1  2500 SP
        2  1200 RJ
        3  1600 SP
        4  1800 RJ
```

O **maior** valor de pedido por **UF** é obtido através da seguinte consulta:

```
SELECT UF, MAX(VALOR) FROM PEDIDO GROUP BY UF;

UF MAX(VALOR)
-- ----------
RJ       1800
SP       2500
```

MIN

A função **MIN** retorna o **menor valor** (mínimo) por grupo das colunas selecionadas.

Exemplo:

```
Tabela: PEDIDO
------------------
NR_PEDIDO VALOR UF
--------- ----- --
        1  2500 SP
        2  1200 RJ
        3  1600 SP
        4  1800 RJ
```

O **menor** valor de pedido por **UF** é obtido através da seguinte consulta:

```
SELECT UF, MIN(VALOR) FROM PEDIDO GROUP BY UF;

UF MIN(VALOR)
-- ----------
RJ       1200
SP       1600
```

COUNT

A função **COUNT** retorna o **número de linhas** por grupo das colunas selecionadas.

Exemplo:

```
Tabela: PEDIDO
------------------
NR_PEDIDO VALOR UF
--------- ----- --
        1  2500 SP
        2  1200 RJ
        3  1600 SP
        4  1800 RJ
```

A **quantidade** de pedidos por **UF** é obtida através da seguinte consulta:

```
SELECT UF, COUNT(VALOR) FROM PEDIDO GROUP BY UF;

UF COUNT(VALOR)
-- ------------
RJ             2
SP             2
```

MEDIAN

A função **MEDIAN** retorna a **mediana** por grupo das colunas selecionadas.

Exemplo:

```
Tabela: EMPREGADO
------------------
DEPT SALARIO
---- -------
  10    2000
  10    1000
  10    5000
  10    4000
  10    3000
```

A função MEDIAN ordena os valores e, caso o número de elementos seja ímpar, retorna o valor central, caso contrário, retorna a média entre os dois valores centrais.

A **mediana** dos salários dos funcionários é obtida através da seguinte consulta:

```
SELECT DEPT, MEDIAN(SALARIO) FROM EMPREGADO GROUP BY DEPT;

DEPT MEDIAN(SALARIO)
---- ---------------
10               3000
```

STDDEV

A função **STDDEV** retorna e **desvio padrão** por grupo das colunas selecionadas.

Exemplo:

```
Tabela: EMPREGADO
------------------
DEPT SALARIO
---- -------
  10    2000
  10    1000
  10    5000
  10    4000
  10    3000
```

O desvio padrão corresponde à raiz quadrada da variância. (Abordada a seguir.)

O **desvio padrão** dos salários dos funcionários é obtido através da seguinte consulta:

```
SELECT DEPT, STDDEV(SALARIO) FROM EMPREGADO GROUP BY DEPT;

DEPT STDDEV(SALARIO)
---- ---------------
10         1581.1388
```

VARIANCE

A função **VARIANCE** retorna a **variância** por grupo das colunas selecionadas.

Exemplo:

```
Tabela: EMPREGADO
------------------
DEPT SALARIO
---- -------
  10    2000
  10    1000
  10    5000
  10    4000
  10    3000
```

A variância é definida como a dispersão ou variação de um grupo de valores numéricos em uma amostra e corresponde ao quadrado do desvio padrão.

A **variância** correspondente aos salários dos funcionários é obtida através da seguinte consulta:

```
SELECT DEPT, VARIANCE(SALARIO) FROM EMPREGADO GROUP BY DEPT;

DEPT VARIANCE(SALARIO)
---- -----------------
10              250000
```

Funções de linha

A documentação da Oracle apresenta 21 funções de linha. Serão abordadas **oito funções de linhas de uso mais comum**, conforme segue:

- UPPER
- LOWER
- INITCAP
- SUBSTR
- REPLACE
- CONCAT
- TRIM
- LENGTH

UPPER

A função **UPPER** retorna todos os caracteres em **maiúsculas**.

Exemplo:

```
Tabela: CLIENTE
---------------
ID   NOME
---- ----------
1001 JOHN SMITH
1002 Mary Clark
1003 noah baker
```

As consultas apenas apresentarão os dados conforme a função de linha utilizada.
Não alterarão os dados gravados nas tabelas.

A consulta a seguir:

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

Retornará:

```
ID   NOME
---- ----------
1001 JOHN SMITH
1002 MARY CLARK
1003 NOAH BAKER
```

LOWER

A função **LOWER** retorna todos os caracteres em **minúsculas**.

Exemplo:

```
Tabela: CLIENTE
---------------
ID   NOME
---- ----------
1001 JOHN SMITH
1002 Mary Clark
1003 noah baker
```

A consulta a seguir:

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

Retornará:

```
ID   NOME
---- ----------
1001 john smith
1002 mary clark
1003 noah baker
```

INITCAP

A função **INITCAP** retorna os primeiros caracteres de cada palavra em **maiúsculas** e os demais em minúsculas.

Exemplo:

```
Tabela: CLIENTE
---------------
ID   NOME
---- ----------
1001 JOHN SMITH
1002 Mary Clark
1003 noah baker
```

A consulta a seguir:

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

Retornará:

```
ID   NOME
---- ----------
1001 John Smith
1002 Mary Clark
1003 Noah Baker
```

SUBSTR

A função **SUBSTR**, quando usada com três parâmetros, apresenta a quantidade de caracteres definida no terceiro parâmetro a partir da posição definida no segundo parâmetro e, quando usada com dois parâmetros, apresenta os caracteres a partir da posição definida no segundo parâmetro até o final da string.

Exemplo:

```
Tabela: CIDADE
--------------
NOME
--------------
MIAMI
```

A consulta:

```
SELECT SUBSTR(NOME,2,3) FROM CIDADE;
```

Retornará:

```
----------
IAM
```

A consulta:

```
SELECT SUBSTR(NOME,2) FROM CIDADE;
```

Retornará:

```
----------
IAMI
```

REPLACE

A função **REPLACE**, quando usada com três parâmetros, substitui os caracteres da string apresentados no segundo parâmetro pelos caracteres do terceiro parâmetro e, quando usada com dois parâmetros, retira da string os parâmetros apresentados no segundo parâmetro e não o substitui por nenhum outro.

Exemplo:

```
Tabela: CIDADE
--------------
NOME
--------------
MIAMI
```

A consulta:

```
SELECT REPLACE(NOME,'I','Z')
FROM CIDADE;
```

Retornará:

```
----------
MZAMZ
```

A consulta:

```
SELECT REPLACE(NOME,'I')
FROM CIDADE;
```

Retornará:

```
----------
MAM
```

CONCAT

A função **CONCAT** concatena os valores das strings.

Exemplo:

```
Tabela: CIDADE
--------------
NOME  ESTADO
----- --------
MIAMI FL
```

A consulta:

```
SELECT CONCAT(NOME,ESTADO)
FROM CIDADE;
```

Retornará:

```
----------
MIAMIFL
```

TRIM

A função **TRIM** remove todos os caracteres especificados de uma string, que poderão estar no início (LEADING), no final (TRAILING) ou em ambos (BOTH) os lados da string.

Exemplo:

```
Tabela: CIDADE
--------------
NOME
--------------
MIAMI
```

A consulta:

```
SELECT TRIM(LEADING 'M' FROM NOME)
FROM CIDADE;
```

Retornará:

```
----------
IMIA
```

A consulta:

```
SELECT TRIM(TRAILING 'I' FROM NOME)
FROM CIDADE;
```

Retornará:

```
----------
MIAM
```

A consulta:

```
SELECT TRIM(BOTH 'I' FROM NOME)
FROM CIDADE;
```

Retornará:

```
----------
MIAM
```

LENGTH

A função **LENGTH** retorna a quantidade de caracteres da string.

Exemplo:

```
Tabela: CIDADE
--------------
NOME
--------------
MIAMI
```

A consulta:

```
SELECT LENGTH(NOME)
FROM CIDADE;
```

Retornará:

```
----------
         5
```

Funções numéricas

A documentação da Oracle apresenta 26 funções numéricas. Serão abordadas sete funções numéricas de uso mais comum, conforme segue:

- **ABS**
- ACOS
- ASIN
- ATAN
- ATAN2
- BITAND
- **CEIL**
- COS
- COSH
- EXP
- **FLOOR**
- LN
- LOG
- **MOD**
- NANVL
- **POWER**
- REMAINDER
- **ROUND**
- SIGN
- SIN
- SINH
- **SQRT**
- TAN
- TANH
- TRUNC
- WIDTH_BUCKET

ABS

A função **ABS** retorna o valor absoluto de um número, independentemente se o número for positivo ou negativo.

A consulta:

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

Retornará:

```
--
10
```

A consulta:

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

Retornará:

```
--
10
```

CEIL

A função **CEIL** retorna o menor valor inteiro que é maior ou igual ao número de entrada fornecido.

A consulta:

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

Retornará:

```
--
10
```

A consulta:

```
SELECT CEIL(10.2) FROM DUAL;
```

Retornará:

```
--
11
```

FLOOR

A função **FLOOR** retorna o maior valor inteiro menor ou igual ao número de entrada fornecido.

A consulta:

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

Retornará:

```
--
10
```

A consulta:

```
SELECT CEIL(10.2) FROM DUAL;
```

Retornará:

```
--
10
```

ROUND

A função **ROUND** arredonda o valor para um número especificado de casas decimais, informado no segundo argumento. Quando recebe apenas o valor, sem o segundo argumento, arredonda para um número inteiro.

A consulta:

```
SELECT ROUND(10.56) FROM DUAL;
```

Retornará:

```
--
11
```

A consulta:

```
SELECT ROUND(10.56,1) FROM DUAL;
```

Retornará:

```
----
10.6
```

MOD

A função **MOD** retorna o resto da divisão. A função recebe dois argumentos: o primeiro é o dividendo e o segundo o divisor.

A consulta:

```
SELECT MOD(10,3) FROM DUAL;
```

Retornará o resto da divisão de 10 por 3:

```
-
1
```

POWER

A função **POWER** retorna o valor da potenciação. A função recebe dois argumentos: o primeiro é a base e o segundo o expoente.

A consulta:

```
SELECT POWER(10,2) FROM DUAL;
```

Retornará o valor correspondente a 10 elevado ao quadrado:

```
---
100
```

SQRT

A função **SQRT** retorna a raiz quadrada do valor informado.

A consulta:

```
SELECT SQRT(36) FROM DUAL;
```

Retornará o valor correspondente à raiz quadrada de 36:

```
-
6
```

Funções de conversão

A documentação da Oracle apresenta 35 funções de conversão. Serão abordadas três funções numéricas de uso mais comum, conforme segue:

- ASCIISTR
- BIN_TO_NUM
- CAST
- CHARTOROWID
- COMPOSE
- CONVERT
- DECOMPOSE
- HEXTORAW
- NUMTODSINTERVAL
- NUMTOYMINTERVAL
- RAWTOHEX
- RAWTONHEX
- ROWIDTOCHAR
- ROWIDTONCHAR
- SCN_TO_TIMESTAMP
- TIMESTAMP_TO_SCN
- TO_BINARY_DOUBLE
- TO_BINARY_FLOAT
- **TO_CHAR**
- TO_CLOB
- **TO_DATE**
- TO_DSINTERVAL
- TO_LOB
- TO_MULTI_BYTE
- TO_NCHAR
- TO_NCLOB
- **TO_NUMBER**
- TO_DSINTERVAL
- TO_SINGLE_BYTE
- TO_TIMESTAMP
- TO_TIMESTAMP_TZ
- TO_YMINTERVAL
- TO_YMINTERVAL
- TRANSLATE ... USING
- UNISTR

TO_CHAR

A função **TO_CHAR** converte valores dos tipos DATE ou NUMBER para um valor do tipo CHAR.

A consulta a seguir retornará o dia do mês, conforme a data do sistema:

```
SELECT TO_CHAR(SYSDATE,'DD') FROM DUAL;
```

Retornará (por exemplo):

```
---
15
```

A consulta a seguir retornará o mês por extenso, conforme a data do sistema:

```
SELECT TO_CHAR(SYSDATE,'MONTH') FROM DUAL;
```

Retornará (por exemplo):

```
-------
JANEIRO
```

A consulta a seguir retornará o valor, conforme notação do sistema inglês (milhares separados por vírgula e casas decimais por ponto):

```
SELECT TO_CHAR(1520,'999,999.99') FROM DUAL;
```

Retornará:

```
-------
1520.00
```

TO_DATE

A função **TO_DATE** converte uma string do tipo CHAR para o tipo DATE.

A consulta:

```
SELECT TO_DATE (SYSDATE,'MM/DD/YY') FROM DUAL;
```

Retornará, por exemplo:

```
--------
01/25/19
```

A função TO_DATE também é utilizada para inserir datas em colunas do tipo DATE.

Exemplo:

```
INSERT INTO CLIENTE (..., DATA_NASC, ...)
VALUES (..., TO_DATE('03/28/1984','MM/DD/YYYY), ...);
```

TO_NUMBER

A função **TO_NUMBER** converte uma string (válida) em um valor do tipo NUMBER.

A consulta:

```
SELECT TO_NUMBER ('$3,250.00','$999,999.99) FROM DUAL;
```

Retornará:

```
--------
3250
```

O primeiro argumento apresenta a string que deverá ser convertida para o tipo NUMBER e o segundo argumento apresenta a "máscara" a ser considerada para conversão da string.

Subqueries (Subconsultas)

SUBQUERIES

Uma **SUBQUERY** (SUBCONSULTA) é uma instrução SQL condicionada dentro de outra instrução SQL. O Oracle Database, por exemplo, permite até 255 níveis de subconsultas.

Há dois tipos de subqueries:

- Subqueries de uma linha
- Subqueries de várias linhas

SUBQUERIES DE UMA LINHA

Subqueries de uma linha retornam 0 (zero) ou uma linha para uma instrução externa.

Exemplo:

```
Tabela: CLIENTE                    Tabela: PEDIDO
--------------------               --------------------
ID_CLIENTE NOME                    NR_PEDIDO ID_CLIENTE
---------- ----------              --------- ----------
1001       JOHN SMITH                      1       1002
1002       MARY CLARK                      2       1001
1003       NOAH BAKER                      3       1004
1004       ROSE MOORE                      4       1003
```

A subquery seguir apresentará o NOME do cliente que fez o pedido com número igual a 3.

```
SELECT NOME FROM CLIENTE WHERE ID_CLIENTE =
  (SELECT ID_CLIENTE FROM PEDIDO WHERE NR_PEDIDO = 3)
```

A consulta retornará:

```
----------
ROSE MOORE
```

SUBQUERIES DE UMA LINHA

Etapas da subquery:

1ª etapa:

```
(SELECT ID_CLIENTE FROM PEDIDO WHERE NR_PEDIDO = 3)
```

Retorna **1004** e este valor é utilizado na próxima etapa:

2ª etapa:

```
SELECT NOME FROM CLIENTE WHERE ID_CLIENTE =
```

1004

No final, conforme exemplo apresentado, a subquery retorna:

```
----------
ROSE MOORE
```

SUBQUERIES DE VÁRIAS LINHAS

Subqueries de várias linhas podem retornar mais de uma linha. Os operadores **IN**, **ANY** e **ALL** são utilizados para verificar se os valores apresentados estão contidos em uma lista de valores.

Operador IN: Retorna todas as linhas que corresponderem a qualquer elemento da lista.

```
Tabela: CLIENTE                    Tabela: PEDIDO
---------------------              ---------------------
ID_CLIENTE NOME                    NR_PEDIDO ID_CLIENTE
---------- ----------              --------- ----------
1001       JOHN SMITH                      1       1002
1002       MARY CLARK                      2       1001
1003       NOAH BAKER                      3       1004
1004       ROSE MOORE                      4       1003
```

A subquery seguir apresentará os NOMES dos clientes contidos na lista (1, 2 e 4).

```
SELECT NOME FROM CLIENTE WHERE ID_CLIENTE IN
  (SELECT ID_CLIENTE FROM PEDIDO WHERE NR_PEDIDO IN (1,2,4)
```

A consulta retornará:

```
----------
MARY CLARK
JOHN SMITH
NOAH BAKER
```

SUBQUERIES DE VÁRIAS LINHAS

Operador ANY: Retorna todas as linhas que corresponderem a **qualquer** valor presente em uma lista. Antes do operador ANY deve-se utilizar um dos seguintes operadores =, <, <=, >, >=, <>.

```
Tabela: CLIENTE                    Tabela: PEDIDO
---------------------              ---------------------
ID_CLIENTE NOME                    NR_PEDIDO ID_CLIENTE
---------- ----------              --------- ----------
1001       JOHN SMITH                      1       1002
1002       MARY CLARK                      2       1001
1003       NOAH BAKER                      3       1004
1004       ROSE MOORE                      4       1003
```

A subquery seguir apresentará os NOMES dos clientes contidos na lista (1 e 4).

```
SELECT NOME FROM CLIENTE WHERE ID_CLIENTE < ANY
  (SELECT ID_CLIENTE FROM PEDIDO WHERE NR_PEDIDO IN (1,4))
```

➡ **Retorna 1002 e 1003**

A consulta retornará:

```
----------
JOHN SMITH
MARY CLARK
```

1001	**É menor que 1002. Portanto atende a condição.**
1002	**Não é menor que 1002, mas é menor que 1003. Portanto atende a condição**
1003	**Não é menor que 1002 ou 1003. Portanto não atende a condição.**
1004	**Não é menor que 1002 ou 1003. Portanto não atende a condição.**

SUBQUERIES DE VÁRIAS LINHAS

Operador ALL: Retorna todas as linhas que corresponderem a **todos** os valores presentes em uma lista. Antes do operador ALL deve-se utilizar um dos seguintes operadores =, <, <=, >, >=, <>.

```
Tabela: CLIENTE                    Tabela: PEDIDO
---------------------              ---------------------
ID_CLIENTE NOME                    NR_PEDIDO ID_CLIENTE
---------- ----------              --------- ----------
1001       JOHN SMITH                      1       1002
1002       MARY CLARK                      2       1001
1003       NOAH BAKER                      3       1004
1004       ROSE MOORE                      4       1003
```

A subquery seguir apresentará os NOMES dos clientes contidos na lista (1 e 4).

```
SELECT NOME FROM CLIENTE WHERE ID_CLIENTE < ALL
  (SELECT ID_CLIENTE FROM PEDIDO WHERE NR_PEDIDO IN (1,4))
```

➡ **Retorna 1002 e 1003**

A consulta retornará:

```
----------
JOHN SMITH
```

1001	**É menor que 1002 e menor que 1003. Portanto atende a condição.**
1002	**Não é menor que 1002. Portanto não atende a condição**
1003	**Não é menor que 1002. Portanto não atende a condição.**
1004	**Não é menor que 1002. Portanto não atende a condição.**

Joins (Junções de tabelas)

JOINS

Uma **JOIN** (JUNÇÃO) é um tipo de consulta SQL usada para recuperar dados de várias tabelas.

O Oracle Database, por exemplo, oferece diversos tipos de joins. Serão abordadas **quatro tipos de joins de uso mais comum**, conforme segue:

- **EQUI JOIN**
- NATURAL JOIN
- COLUMN NAME JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- **NON EQUI JOIN**
- **SELF JOIN**
- **CROSS JOIN**

EQUI JOIN

Apresenta todos os registros onde são encontradas correspondências (igualdade de valores) entre campos comuns de duas ou mais tabelas.

```
Tabela: CLIENTE                Tabela: PEDIDO
---------------------          --------------------
ID_CLIENTE NOME                NR_PEDIDO ID_CLIENTE
---------- ----------          --------- ----------
1001       JOHN SMITH                  1       1002
1002       MARY CLARK                  2       1001
1003       NOAH BAKER                  3       1004
1004       ROSE MOORE                  4       1003
```

A subquery seguir apresentará os NOMES dos clientes e os respectivos números de pedidos por eles emitidos.

```
SELECT C.ID_CLIENTE, P.NR_PEDIDO
FROM CLIENTE C INNER JOIN PEDIDO P
ON C.ID_CLIENTE = P.ID_CLIENTE
```

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

A consulta retornará:

```
---------- ----
JOHN SMITH    2
MARY CLARK    1
NOAH BAKER    4
ROSE MOORE    3
```

NON EQUI JOIN

Apresenta todos os registros onde não necessariamente são encontradas correspondências (igualdade de valores) entre campos de duas ou mais tabelas.

Este tipo de join relaciona um valor obtido em uma tabela a um "range" ou faixa de valores de outra, conforme apresentado a seguir.

```
Tabela: PRODUTO                Tabela: CLASSE
---------------------          ---------------------------
ID_PRODUTO  VALOR              FAIXA  VALOR_MIN  VALOR_MAX
----------  ---------          -----  ---------  ---------
1001            75.00          A           0.01      50.00
1002           120.00          B          50.01     100.00
1003            30.00          C         100.01     150.00
```

A subquery a seguir apresentará o ID_PRODUTO e a FAIXA (A, B ou C) que cada PRODUTO está relacionado.

```
SELECT P.ID_PRODUTO, C.FAIXA
FROM PRODUTO P INNER JOIN CLASSE C
ON P.VALOR BETWEEN C.VALOR_MIN AND C.VALOR_MAX;
```

A consulta retornará:

```
---------- ----
1001       B
1002       C
1003       A
```

SELF JOIN

Apresenta todos os registros onde são encontradas correspondências (igualdade de valores) entre campos de uma mesma tabela.

```
Tabela: EMPREGADO
-----------------------------
ID_EMP  NOME_EMP    ID_GERENTE
------- ----------- ----------
1001    JOHN SMITH
1002    MARY CLARK  1001
1003    NOAH BAKER
1004    ROSE MOORE  1003
```

A subquery a seguir apresentará o NOMES dos EMPREGADOS e os respectivos NOMES de seus GERENTES.

```
SELECT E1.NOME_EMP, E2.NOME_EMP "GERENTE"
FROM EMPREGADO E1
INNER JOIN EMPREGADO E2
ON E1.ID_GERENTE = E2.ID_EMP;
```

A consulta retornará:

```
NOME_EMP    GERENTE
----------- ----------
MARY CLARK  JOHN SMITH
ROSE MOORE  NOAH BAKER
```

CROSS JOIN

Apresenta todos as combinações possíveis entre os registros das tabelas declaradas na consulta.

```
Tabela: TIME_SP                Tabela: TIME_EN
--------------------           --------------------
ID_TIME  NOME_TIME             ID_TIME  NOME_TIME
-------- -----------           -------- ----------
      11 REAL MADRID                 21 CHELSEA
      12 BARCELONA                   22 LIVERPOOL
```

A subquery a seguir apresenta todas as combinações possíveis nas quais times espanhóis (TIME_SP) enfrentam times ingleses (TIME_EN).

```
SELECT A.TIME_SP, B.TIME_EN
FROM TIME_SP A
CROSS JOIN TIME_EN B;
```

A consulta retornará:

```
TIME_SP      TIME_EN
------------ ----------
REAL MADRID  CHELSEA
REAL MADRID  LIVERPOOL
BARCELONA    CHELSEA
BARCELONA    LIVERPOOL
```

Merge (Mesclar tabelas)

MERGE

Utilizado para **mesclar** dados de duas ou mais tabelas. Pode combinar em uma única instrução os comandos INSERT, UPDATE e DELETE.

```
Tabela: EMPREGADO_A
--------------------
ID    NOME          SALARIO
----- ----------- -------
1001  JOHN SMITH      1000
1002  MARY CLARK      1200
```

```
Tabela: EMPREGADO_B
--------------------
ID    NOME          SALARIO
----- ----------- -------
1001  JOHN SMITH      1500
1003  NOAH BAKER      1000
```

A instrução a seguir atualizará a tabela EMPREGADO_A a partir de dados mais atualizados encontrados na tabela EMPREGADO_B e também incluirá na tabela EMPEGADO_A os dados da tabela EMPREGADO_B que não constam nela.

```
MERGE INTO EMPREGADO_A A
USING EMPREGADO_B B ON (A.ID = B.ID)
WHEN MATCHED THEN
UPDATE SET A.NOME = B.NOME, A.SALARIO = B.SALARIO
WHEN NOT MATCHED THEN
INSERT (A.ID, A.NOME, A.SALARIO) VALUES (B.ID, B.NOME, B.SALARIO);
```

Operadores de Conjunto

UNION

Retorna todas as linhas ou valores de duas ou mais tabelas. Retorna, caso existam, apenas uma vez valores duplicados.

```
Tabela: EMPREGADO_A
--------------------
ID    NOME          SALARIO
----- ----------- -------
1001  JOHN SMITH      1000
1002  MARY CLARK      1200
```

```
Tabela: EMPREGADO_B
--------------------
ID    NOME          SALARIO
----- ----------- -------
1001  JOHN SMITH      1500
1003  NOAH BAKER      1000
```

A instrução a seguir apresentará todos os NOMES dos empregados que constam nas tabelas **EMPREGADO_A** e **EMPREGADO_B**. Os nomes que se encontram duplicados nas tabelas consultadas serão apresentados apenas uma vez.

```
SELECT NOME FROM EMPREGADO_A
UNION
SELECT NOME FROM EMPREGADO_B;
```

A consulta retornará:

```
------------
JOHN SMITH
MARY CLARK
NOAH BAKER
```

UNION ALL

Retorna todas as linhas ou valores de duas ou mais tabelas. Retorna, caso existam, valores duplicados.

```
Tabela: EMPREGADO_A
--------------------
ID    NOME          SALARIO
----- ------------ -------
1001  JOHN SMITH      1000
1002  MARY CLARK      1200
```

```
Tabela: EMPREGADO_B
--------------------
ID    NOME          SALARIO
----- ------------ -------
1001  JOHN SMITH      1500
1003  NOAH BAKER      1000
```

A instrução a seguir apresentará todos os NOMES dos empregados que constam nas tabelas **EMPREGADO_A** e **EMPREGADO_B**. Os nomes que se encontram duplicados nas tabelas consultadas serão apresentados mais de uma vez.

```
SELECT NOME FROM EMPREGADO_A
UNION ALL
SELECT NOME FROM EMPREGADO_B;
```

A consulta retornará:

```
------------
JOHN SMITH
MARY CLARK
JOHN SMITH
NOAH BAKER
```

INTERSECT

Retorna todas as linhas ou valores comuns às duas tabelas.

```
Tabela: EMPREGADO_A
--------------------
ID    NOME          SALARIO
----- ------------ -------
1001  JOHN SMITH      1000
1002  MARY CLARK      1200
```

```
Tabela: EMPREGADO_B
--------------------
ID    NOME          SALARIO
----- ------------ -------
1001  JOHN SMITH      1500
1003  NOAH BAKER      1000
```

A instrução a seguir apresentará todos os NOMES dos empregados que constam na tabela **EMPREGADO_A** e também na tabela **EMPREGADO_B**.

```
SELECT NOME FROM EMPREGADO_A
INTERSECT
SELECT NOME FROM EMPREGADO_B;
```

A consulta retornará:

```
------------
JOHN SMITH
```

MINUS

Retorna todas as linhas ou valores da primeira tabela que não aparecem na segunda.

```
Tabela: EMPREGADO_A
--------------------
ID    NOME          SALARIO
----- ------------ -------
1001  JOHN SMITH      1000
1002  MARY CLARK      1200
```

```
Tabela: EMPREGADO_B
--------------------
ID    NOME          SALARIO
----- ------------ -------
1001  JOHN SMITH      1500
1003  NOAH BAKER      1000
```

A instrução a seguir apresentará todos os NOMES dos empregados que constam na tabela **EMPREGADO_A** e não constam na tabela **EMPREGADO_B**.

```
SELECT NOME FROM EMPREGADO_A
MINUS
SELECT NOME FROM EMPREGADO_B;
```

A consulta retornará:

```
------------
MARY CLARK
```

INDEXES

Armazenam valores das colunas indexadas juntamente com o RowID da respectiva linha. O objetivo principal é melhorar o tempo de resposta ao realizar consultas que usam como referência as colunas indexadas.

```
Tabela: EMPREGADO
-------------------------
ID    NOME         SALARIO
----- ----------- -------
1001  JOHN SMITH     1000
1002  MARY CLARK     1200
1003  NOAH BAKER     1000
```

A instrução a seguir criará um índice denominado **EMPREGADO_IDX** na coluna **NOME** da tabela **EMPREGADO**.

```
CREATE INDEX EMPREGADO_IDX
ON EMPREGADO (NOME);
```

Para eliminar o índice **EMPREGADO_IDX** deve-se usar o seguinte comando:

```
DROP INDEX EMPREGADO_IDX;
```

Nota: O RowID do Oracle Database utiliza um sistema numérico de base 64 para representar o endereço exclusivo de uma linha da tabela.

VIEWS

Representações lógicas de uma ou mais tabelas. Derivam seus dados de "tabelas-base" ou de outras views.

Operações realizadas nas views afetam as "tabelas base".

```
Tabela: EMPREGADO
-------------------------
ID    NOME          SALARIO
----- ----------- -------
1001  JOHN SMITH      1000
1002  MARY CLARK      1200
1003  NOAH BAKER      1000
```

A instrução a seguir criará uma VIEW denominada **EMPREGADO_VW** com base nas colunas **ID** e **NOME** da tabela **EMPREGRADO**.

```
CREATE VIEW EMPREGADO_VW (ID, NOME)
AS
SELECT ID, NOME FROM EMPREGADO;
```

Para eliminar a view **EMPREGADO_VW** deve-se usar o seguinte comando:

```
DROP VIEW EMPREGADO_VW;
```

VÍDEOS - SQL (Structured Query Language)

Vídeo 1

https://vimeo.com/618838154

Vídeo 2

https://vimeo.com/618852025

PL/SQL

Procedural Language / Structured Query Language

Objetivo

Apresentar os principais recursos da PL/SQL (Procedural Language / Structured Query Language) para manipulação de dados utilizando linguagem estruturada de programação com a criação de procedures, functions, triggers e packages.

Conteúdo Programático

- Programação para banco de dados
- Declarações
- Tipos de dados
- Constantes e variáveis
- Comandos SQL dentro de um bloco PL/SQL
- Instruções IF-THEN-ELSE e CASE
- Instruções LOOP, FOR e WHILE
- Tratamento de exceções
- Cursores explícitos e implícitos
- Procedures
- Functions
- Triggers
- Packages

Programação para Banco de Dados

PL/SQL: Procedural Language/Stuctured Query Language

Principais recursos da linguagem:

- Executar comandos SQL para manipular dados nas tabelas
- Criar constantes e variáveis
- Criar cursores para tratar o resultado de uma consulta
- Criar registros para guardar o resultado de um cursor ou campo de tabela
- Tratar erros
- Utilizar comandos de controle (if, if-then-else, case) e repetição (loop, for, while)

Programação para Banco de Dados

PL/SQL: Procedural Language/Stuctured Query Language

Vantagens:

- Versatilidade
- Portabilidade
- Integração com o SGBD (Sistema Gerenciador de Banco de Dados)
- Capacidade procedural (comandos de controle e repetição)
- Redução de tráfego de rede

Estrutura de um bloco PL/SQL

Estrutura de um bloco PL/SQL anônimo:

```
DECLARE
 Inicializações, declaração de constantes, variáveis e cursores
BEGIN
 Comandos SQL, estruturas de programação e outros blocos PL/SQL
 BEGIN
 ...
 END;
EXCEPTION (opcional)
 Tratamento de exceções, emissão de mensagens
END;
```

Nota: Além de blocos anônimos, a PL/SQL permite a criação de PROCEDURES, FUNCTIONS, TRIGGERS, etc. que serão abordados nesta disciplina.

Declarações

Na área DECLARE podemos declarar:

- constantes
- variáveis
- cursores
- estruturas
- tabelas

Comentários no código PL/SQL

Para inserir comentários de uma linha utilizamos:

```
-- Comentário
```

Para inserir comentários de múltiplas linhas utilizamos:

```
/*
Comentário linha 1
Comentário linha 2
...
*/
```

Tipos de Dados

Os principais tipos de dados simples que podem ser utilizados em declarações PL/SQL são:

- **CHAR:** Alfanumérico, tamanho fixo, limite: 2000 caracteres
- **VARCHAR2:** Alfanumérico, tamanho variável, limite: 4000 caracteres
- **CLOB:** (Character Long Object) Alfanumérico, tamanho variável, limite 4 Gb
- **LONG:** Alfanumérico, limites: 2 GB, apenas um por tabela
- **ROWID:** Armazena os valores dos ROWIDs das linhas das tabelas
- **BLOB:** (Binary Long Object) Binário, tamanho variável, limite: 4 Gb
- **BFILE:** (Binary File) Armazena uma referência a um arquivo externo
- **RAW:** Hexadecimais, tamanho variável, limite: 2 Kb
- **LONG ROW:** Hexadecimais, tamanho variável, limite: 2 Gb
- **NUMBER:** Numérico, limite: 38 dígitos
    - SUBTIPOS: **DECIMAL, INT, NUMERIC, FLOAT**
- **DATE:** Data e hora
- **TIMESTAMP:** Data e hora (com milésimos de segundo)
- **BOOLEAN:** Armazena os valores: TRUE, FALSE ou NULL

IMPORTANTE

Quando utilizamos o SQL Plus é preciso digitar o seguinte comando para que o ambiente retorne o resultado dos blocos PL/SQL:

```
SET SERVEROUTPUT ON
```

Constantes

Para declarações de constantes, a palavra **CONSTANT** deve aparecer antes do tipo de dado e a seguir utiliza-se o operador **:=** para atribuir-lhe um valor.

Exemplo:

```
DECLARE
 PI CONSTANT NUMBER(3,2) := 3.14;
BEGIN
 DBMS_OUTPUT.PUT_LINE(PI);
END;
/
```

A procedure **DBMS_OUTPUT.PUT_LINE** permite direcionar a saída PL/SQL para uma tela.

Variáveis

As variáveis são inicializadas de maneira similar às constantes: declarando-se o tipo e atribuindo-lhe um valor.

Variáveis não inicializadas explicitamente recebem o valor NULL.

Pode-se aplicar a restrição NOT NULL a uma variável. Neste caso, ela deverá ser inicializada.

```
DECLARE
 V_1 NUMBER(4):= 1;
 V_2 NUMBER(4) NOT NULL := 1;
BEGIN
 V_1 := 10;
 V_2 := 20;
 DBMS_OUTPUT.PUT_LINE('V_1 vale ' || V_1);
 DBMS_OUTPUT.PUT_LINE('V_2 vale ' || V_2);
END;
/
```

Na PL/SQL **||** é usado para concatenação.

Variáveis

Podemos atribuir valores às variáveis de duas maneiras:

- Utilizando o operador de atribuição:

```
TOTAL := QUANT * VALOR;
```

- Utilizando um comando SELECT com a cláusula INTO (na seção BEGIN do bloco):

```
SELECT RA, NOME
INTO V_RA, V_NOME
FROM ALUNO;
```

Comandos SQL dentro de uma bloco PL/SQL

Comandos DML (SELECT, INSERT, UPDATE e DELETE) podem ser utilizados dentro de um bloco PL/SQL.

O comando SELECT deverá receber obrigatoriamente a cláusula INTO para que o resultado seja armazenado em variáveis e deverá retornar apenas uma linha.

Caso mais de uma linha seja retornada, apresentará o erro: **too_many_rows** e se não retornar nenhuma linha, apresentará o erro: **no_data_found**. (Detalhes serão apresentados em Tratamento de Exceções).

```
CREATE TABLE ALUNO (
RA NUMBER(4),
NOME VARCHAR2(30));

INSERT INTO ALUNO (RA,NOME) VALUES (1001,'ANTONIO');
INSERT INTO ALUNO (RA,NOME) VALUES (1002,'BEATRIZ');

DECLARE
 V_RA NUMBER(4);
 V_NOME VARCHAR2(30);
BEGIN
 SELECT RA, NOME INTO V_RA, V_NOME FROM ALUNO WHERE RA=1001;
 DBMS_OUTPUT.PUT_LINE(V_RA ||' - '|| V_NOME);
END;
/
```

Herança de Tipo e Tamanho

As constantes e variáveis podem herdar o tipo de outras variáveis, de colunas ou até da linha inteira de uma tabela.

Desta forma, diminuem-se as manutenções oriundas de alterações realizadas nas colunas de tabelas (Exemplo: Quando altera-se o tamanho da coluna).

Herdando o tipo de uma coluna de uma tabela:

```
V_NOME ALUNO.NOME%TYPE;
```

Herdando o tipo de uma linha inteira de uma tabela:

```
V_ALUNO ALUNO_ROWTYPE;
```

Herdando o tipo de uma variável previamente declarada:

```
V_NOME_ALUNO V_NOME%TYPE;
```

Herança de Tipo e Tamanho

Exemplo: Variáveis herdam os respectivos tipos das colunas da tabela:

```
CREATE TABLE ALUNO (
RA NUMBER(4),
NOME VARCHAR2(40));

INSERT INTO ALUNO (RA,NOME) VALUES (1001,'ANTONIO');
INSERT INTO ALUNO (RA,NOME) VALUES (1002,'BEATRIZ');

DECLARE
 V_RA ALUNO.RA%TYPE;
 V_NOME ALUNO.NOME%TYPE;
BEGIN
 SELECT RA, NOME INTO V_RA, V_NOME FROM ALUNO WHERE RA=1001;
 DBMS_OUTPUT.PUT_LINE(V_RA ||' - '|| V_NOME);
END;
/
```

Herança de Tipo e Tamanho

Exemplo: Variáveis herdam uma linha inteira tabela:

```
CREATE TABLE ALUNO (
RA NUMBER(4),
NOME VARCHAR2(40));

INSERT INTO ALUNO (RA,NOME) VALUES (1001,'ANTONIO');
INSERT INTO ALUNO (RA,NOME) VALUES (1002,'BEATRIZ');

DECLARE
 V_ALUNO ALUNO%ROWTYPE;
BEGIN
 SELECT * INTO V_ALUNO FROM ALUNO WHERE RA=1001;
 DBMS_OUTPUT.PUT_LINE(V_ALUNO.RA ||' - '|| V_ALUNO.NOME);
END;
/
```

Herança de Tipo e Tamanho

Exemplo: Variáveis herdam o tipo de uma variável previamente declarada:

```
CREATE TABLE ALUNO (
RA NUMBER(4),
NOME VARCHAR2(40));

INSERT INTO ALUNO (RA,NOME) VALUES (1001,'ANTONIO');
INSERT INTO ALUNO (RA,NOME) VALUES (1002,'BEATRIZ');

DECLARE
 V_RA NUMBER(4);
 V_NOME VARCHAR2(30);
 V_RA_ALUNO V_RA%TYPE;
 V_NOME_ALUNO V_NOME%TYPE;
BEGIN
 SELECT RA, NOME INTO V_RA_ALUNO, V_NOME_ALUNO FROM ALUNO WHERE RA=1001;
 DBMS_OUTPUT.PUT_LINE(V_RA_ALUNO ||' - '|| V_NOME_ALUNO);
END;
/
```

Instruções IF-THEN-ELSE e CASE

IF-THEN-ELSE

Executa um conjunto de ações de acordo com uma ou mais condições.

```
DECLARE
 V_RA ALUNO.RA%TYPE := 1001;
 V_NOTA ALUNO.NOTA%TYPE;
 V_CONCEITO VARCHAR2(12);
BEGIN
 SELECT NOTA INTO V_NOTA FROM ALUNO WHERE RA = V_RA;
 IF V_NOTA <= 5 THEN V_CONCEITO := 'REGULAR';
 ELSIF V_NOTA < 7  THEN V_CONCEITO := 'BOM';
 ELSE V_CONCEITO := 'EXCELENTE';
 END IF;
 DBMS_OUTPUT.PUT_LINE (V_CONCEITO);
END;
/
```

Instruções IF-THEN-ELSE e CASE

CASE

Retorna determinado resultado de acordo com o valor da variável de comparação.

```
DECLARE
 V_RA ALUNO.RA%TYPE := 1;
 V_NOTA ALUNO.NOTA%TYPE;
 V_CONCEITO VARCHAR2(12);
BEGIN
 SELECT NOTA INTO V_NOTA FROM ALUNO WHERE RA = V_RA;
 V_CONCEITO :=
 CASE
  WHEN V_NOTA <= 5 THEN 'REGULAR'
  WHEN V_NOTA < 7 THEN 'BOM'
  ELSE 'EXCELENTE'
 END;
 DBMS_OUTPUT.PUT_LINE (V_CONCEITO);
END;
/
```

Instruções LOOP, FOR e WHILE

LOOP

Repete um bloco de comando n vezes, até que a variável contadora atinja o seu valor final.

A variável contadora não deve ser declarada na seção DECLARE e deixará de existir após a execução do comando END LOOP.

```
DECLARE
 V_AUX NUMBER(2) := 0;
BEGIN
 FOR V_CONTADOR IN 1..10
  LOOP
   V_AUX := V_AUX +1;
   DBMS_OUTPUT.PUT_LINE (V_AUX);
  END LOOP;
END;
/
```

Instruções LOOP, FOR e WHILE

WHILE

Repete um bloco de comandos enquanto a condição que segue o comando WHILE for verdadeira.

```
DECLARE
 V_AUX NUMBER(2) := 0;
BEGIN
 WHILE V_AUX < 10
  LOOP
   V_AUX := V_AUX +1;
   DBMS_OUTPUT.PUT_LINE (V_AUX);
  END LOOP;
END;
/
```

Tratamento de Exceções

Exceções são erros ou imprevistos que podem ocorrer durante a execução de um bloco PL/SQL.

Nesses casos, o gerenciador de banco de dados aborta a execução e procura a área de exceções **EXCEPTIONS**.

As exceções podem ser:

- Predefinidas
- Definidas pelo usuário

Tratamento de Exceções

PREDEFINIDAS

Disparadas automaticamente quando, no bloco PL/SQL, uma regra Oracle for violada.

Podem ser identificadas por um número ou um nome.

ERRO	NOME	DESCRIÇÃO
ORA-00001	DUP_VAL_ON_INDEX	Tentativa de armazenar valor duplicado em uma coluna que possui chave primária ou única.
ORA-01012	NOT_LOGGED_ON	Tentativa acessar o banco de dados sem estar conectado a ele.
ORA-01403	NO_DATA_FOUND	Ocorre quando um comando SELECT ... INTO não retorna nenhuma linha.
ORA-01422	TOO_MANY_ROWS	Ocorre quando um comando SELECT ... INTO retorna mais de uma linha.
ORA-01476	ZERO_DIVIDE	Tentativa de dividir qualquer número por zero.

Tratamento de Exceções

PREDEFINIDAS

Exemplo:

```
DECLARE
 V_RA ALUNO.RA%TYPE;
 V_NOME ALUNO.NOME%TYPE;
BEGIN
 SELECT RA, NOME INTO V_RA, V_NOME FROM ALUNO WHERE RA=1001;
 DBMS_OUTPUT.PUT_LINE(V_RA ||' - '|| V_NOME);
EXCEPTION
 WHEN NO_DATA_FOUND
  THEN DBMS_OUTPUT.PUT_LINE ('Não há nenhum aluno com este RA');
 WHEN TOO_MANY_ROWS
  THEN DBMS_OUTPUT.PUT_LINE ('Há mais de um aluno com este RA');
 WHEN OTHERS
  THEN DBMS_OUTPUT.PUT_LINE ('Erro desconhecido');
END;
```

Tratamento de Exceções

DEFINIDAS PELO USUÁRIO

Além dos erros tratados automaticamente pelo Oracle, regras de negócio específicas podem ser tratadas.

As exceções definidas pelo usuário devem ser declaradas e chamadas explicitamente pelo comando RAISE.

```
DECLARE
 V_RA ALUNO.RA%TYPE;
 V_NOTA ALUNO.NOTA%TYPE;
 V_CONTA NUMBER(2);
 ALUNO_EXCEPT EXCEPTION;
BEGIN
 SELECT COUNT(RA) INTO V_CONTA FROM ALUNO;
 IF V_CONTA = 10 THEN
  RAISE ALUNO_EXCEPT;
  ELSE INSERT INTO ALUNO VALUES (20,'SILVA');
 END IF;
EXCEPTION
 WHEN CONTA_ALUNO THEN DBMS_OUTPUT.PUT_LINE('Turma cheia!');
END;
/
```

Cursores

Cursores são áreas compostas de linhas e colunas em memória que servem para armazenar o resultado de uma seleção que retorna 0 (zero) ou mais linhas.

Na PL/SQL os cursores podem ser de dois tipos:

- Explícitos
- Implícitos

Cursores

EXPLÍCITOS

São utilizados para execução de consultas que possam retornar nenhuma ou mais de uma linha.

Este tipo de cursor deve ser explicitamente declarado na área DECLARE.

Após sua declaração, o cursor deverá ser manipulado com o uso de alguns comandos:

- **OPEN**: abre o cursor
- **FETCH**: disponibiliza a linha corrente e posiciona na próxima linha do cursor
- **CLOSE**: fecha o cursor

Cursores

EXPLÍCITOS

Para cada cursor, quatro atributos podem ser verificados, e seus valores podem ser alterados a cada execução de um comando FETCH. Esses atributos são:

- **nome_do_cursor%FOUND**: Retorna TRUE caso FETCH consiga retornar alguma linha e FALSE caso contrário.
- **nome_do_cursor%NOTFOUND**: Retorna FALSE caso FETCH consiga retornar alguma linha e TRUE caso contrário.
- **nome_do_cursor%ROWCOUNT**: Retorna o número de linhas já processadas pelo cursor.
- **nome_do_cursor%ISOPEN**: Retona TRUE caso o cursor esteja aberto e FALSE caso contrário.

Cursores

EXPLÍCITOS

Exemplo:

```
DECLARE
 CURSOR C_CLIENTE IS SELECT ID, NOME FROM CLIENTE;
 V_CLIENTE C_CLIENTE%ROWTYPE;
BEGIN
 OPEN C_CLIENTE;
  LOOP
   FETCH C_CLIENTE INTO V_CLIENTE;
   EXIT WHEN C_CLIENTE%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(V_CLIENTE.NOME);
  END LOOP;
 CLOSE C_CLIENTE;
END;
/
```

Cursores

IMPLÍCITOS

Utilizados para processar instruções (**SELECT ... INTO**, **INSERT**, **UPDATE** e **DELETE**).

Os comandos **OPEN**, **FETCH** e **CLOSE** não podem ser aplicados a este tipo de cursor.

Cursores implícitos esperam que apenas uma linha seja retornada. Por isso exceções tais como **NO_DATA_FOUND** (nenhuma linha satisfaz os critérios de seleção) ou **TOO_MANY_ROWS** (mais de uma linha satisfaz o critério de seleção) devem ser observadas.

Os atributos **%FOUND**, **%NOTFOUND**, **%ROWCOUNT** e **%ISOPEN** também podem ser verificados neste tipo de cursor.

Procedures

Subprogramas que executam uma determinada ação. Não retornam valores diretamente para o ambiente. Portanto, não podem ser utilizadas como argumento em um comando SELECT.

```
CREATE OR REPLACE PROCEDURE SOMA (
 P_1 NUMBER,
 P_2 NUMBER) IS
 V_TOTAL NUMBER;
BEGIN
 V_TOTAL := P_1 + P_2;
 DBMS_OUTPUT.PUT_LINE(V_TOTAL);
END SOMA;
/

SET SERVEROUTPUT ON

EXEC SOMA (2, 5);
```

NOTA: O comando **SET SERVEROUTPUT ON** é usado para visualizar a saída (resultado) acima.

Functions

Functions (Funções) são subprogramas que executam uma determinada ação e retornam valores. Portanto, podem ser invocadas por meio de um comando SELECT.

```
CREATE OR REPLACE FUNCTION SOMA (
 P_1 NUMBER,
 P_2 NUMBER)
 RETURN NUMBER
 IS
 V_TOTAL NUMBER;
BEGIN
 V_TOTAL := P_1 + P_2;
 RETURN V_TOTAL;
END SOMA;
/

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

Triggers

Triggers (gatilhos) são blocos PL/SQL disparados automática e implicitamente sempre que ocorrer um evento associado a uma tabela (INSERT, UPDATE ou DELETE).

```
CREATE TABLE PRODUTO (
CODIGO NUMBER (4),
VALOR NUMBER (7,2) );

CREATE TABLE HISTORICO (
CODIGO NUMBER (4),
VALOR_ANTIGO NUMBER (7,2),
VALOR_NOVO NUMBER (7,2),
DATA_ATUAL DATE );

CREATE OR REPLACE TRIGGER VALOR_HISTORICO
 BEFORE UPDATE OF VALOR ON PRODUTO
 FOR EACH ROW
BEGIN
 INSERT INTO HISTORICO VALUES (:OLD.CODIGO, :OLD.VALOR, :NEW.VALOR, SYSDATE);
END;
/
```

Packages

Packages (pacotes) são objetos de banco de dados equivalentes a bibliotecas que armazenam:

- procedures
- functions
- definições de cursores
- variáveis e constantes
- definições de exceções

Um package é comporto de duas partes:

1. **Especificação:** Área onde são feitas as declarações públicas. As variáveis, constantes, cursores, exceções e subprogramas estarão disponíveis para uso externo ao package.
2. **Corpo:** Área onde são feitas as declarações privadas que estarão disponíveis apenas dentro do package e a definição de ações para os subprogramas públicos e privados.

O Corpo do package é um objeto de dicionário de dados separado da Especificação. Ele não poderá ser compilado com sucesso a menos que a Especificação já tenha sido compilada.

Packages

Exemplo:

```
CREATE OR REPLACE PACKAGE PACK_1 IS
 PROCEDURE PROC_1;
 FUNCTION FUNC_1 RETURN VARCHAR2;
END PACK_1;
/

CREATE OR REPLACE PACKAGE BODY PACK_1 IS
 PROCEDURE PROC_1
 IS
BEGIN
 DBMS_OUTPUT.PUT_LINE('Mensagem da Procedure');
END PROC_1;
FUNCTION FUNC_1 RETURN VARCHAR2 IS
BEGIN
 RETURN('Mensagem da Function');
END FUNC_1;
END PACK_1;
/

EXEC PACK_1.PROC_1;

SELECT PACK_1.FUNC_1 FROM DUAL;
```

VÍDEOS - PL/SQL (Procedural Language/Structured Query Language)

Vídeo 1

https://vimeo.com/618859783

Vídeo 2

https://vimeo.com/618869456