**SQL**

**Dado:** Fato do mundo real registrado.

Ex.: endereço, data.

**Informação**: Fato útil que pode ser extraído direto ou indiretamente a partir de dados.  
**ex.:** data de entrada, idade.  
  
**Alias**

Para encurtar o script podemos criar um (alias) que serve como codinome para chamar algo já mencionado anteriormente.

ex.: SELECT \* FROM ATENDIME a  
WHERE a.CD\_ATENDIMENTO='325457';

Aqui coloquei o (a) para gerar o alias assim não precisando mencionar o nome da table  
completo.

------------------------------------------------------------------------------------------------------------------------------

**Operadores lógicos**

Operadores lógicos testam a legitimidade de algumas condições. Os operadores lógicos, como

operadores de comparação, retornam um tipo de dados Boolean com um valor TRUE, FALSE ou

UNKNOWN.

------------------------------------------------------------------------------------------------------------------------------

**WHERE** - Irei impor uma condição para o script onde determino o que irá trazer.

(ex.: SELECT \* FROM ATENDIME a

WHERE a.CD\_ATENDIMENTO='325457';)

------------------------------------------------------------------------------------------------------------------------------

**Trunc – pode ser utilizar para mascarar o campo de hora em campos de data e hora.**

***SELECT cod\_aluno ,Trunc(nascimento) AS NASCIMENTO, NOME  
FROM taluno  
WHERE Trunc(nascimento) <> '29/08/2002'  
ORDER BY cod\_aluno asc;***

***Selecionei os campos de código de aluno, nascimento e nome da tabela TALUNO aonde quis que eles fossem diferentes da data 29/08/2002 e a ordem através código de aluno usei o (TRUNC) par que não precisa-se colocar a hora também no campo.***

**BETEWEEN** - Condição entre dados para compara um valor entre um intervalo de exatamente dois valores.

Ex.:

SELECT \* FROM SENSO  
WHERE COD\_UF='35'  
 AND POPULACAO BETWEEN 5000 AND 10000  
AND ANO='2014'  
ORDER BY POPULACAO DESC;

Obs. do script: Selecionar na tabela SENSO com o COD\_UF=’35’ as populações entre 5000 e 10000 no ano de 2014 e ordenar a em ordem decrescente.

-----------------------------------------------------------------------------------------------------------------------------

**IN** - Faz a comparação de um valor contra um conjunto fixo ou até mesmo com uma subquery.

------------------------------------------------------------------------------------------------------------------------------

**HAVING** – Utilizamos quando precisamos de junção de colunas, é aplicada às linhas no conjunto de resultados. Somente os grupos que atendem os critérios de HAVING são exibidos na saída da consulta. Você pode aplicar apenas uma cláusula HAVING em colunas que também são exibidas na cláusula GROUP BY ou em uma função de agregação.

*Ex.:*

*SELECT COD\_UF,ESTADO,COUNT(\*) QTD,  
SUM(POPULACAO)  
FROM SENSO  
WHERE ANO='2014'  
GROUP BY COD\_UF, ESTADO HAVING SUM(POPULACAO)<5000000;*

Por exemplo, imagine que você está unindo as tabelas titles e publishers para criar uma consulta que mostra o preço médio do livro de um conjunto de editoras. Você quer ver o preço médio de um único conjunto específico de editoras, talvez somente as editoras no estado da Califórnia. Além disso, desejar ver o preço médio apenas se estiver acima de R$ 10,00.

Você pode definir o primeiro critério incluindo uma cláusula WHERE, que descarta todas as editoras que não estejam na Califórnia, antes de calcular os preços médios. O segundo critério requer uma cláusula HAVING, porque o critério está baseado nos resultados de agrupamento e resumo dos dados. A instrução SQL resultante se parecerá com esta:

*SELECT titles.pub\_id, AVG(titles.price)*

*FROM titles INNER JOIN publishers*

*ON titles.pub\_id = publishers.pub\_id*

*WHERE publishers.state = 'CA'*

*GROUP BY titles.pub\_id   
HAVING AVG(price) > 10*

Você pode criar cláusulas HAVING e WHERE no painel Critérios. Por padrão, se você especificar um critério de pesquisa para uma coluna, o critério se tornará parte da cláusula HAVING. Porém, você pode alterar o critério para ser uma cláusula WHERE.

Você pode criar uma cláusula WHERE e uma cláusula HAVING envolvendo a mesma coluna. Para fazer isso, você deve adicionar a coluna duas vezes ao painel Critérios e, depois, especificar uma instância como parte da cláusula HAVING e a outra instância como parte da cláusula WHERE.

**Funções de Caracteres**

|  |  |
| --- | --- |
| Função | Propósito |
| Concat (column1|expression1, column2|expression2) | Concatena a primeira string de caracteres com a segunda. Equivalente ao operador de concatenação(||). |
| Initcap (column|expression) | Converte strings de caracteres deixando a primeira letra de cada palavra em maiúscula e as demais em minúsculas. |
| instr (COLUMN\Expression,m) | Retorna a posição numérica do caractere dentro da string. |
| length (column\expression) | Retorna o numero de caracteres da string. |
| lower (column\expression) | Converte strings de caracteres para minúsculas. |
| lpad(column\expression,n, ‘string’) | Retorna uma string com tamanho total de n alinhada à direita. |
| replace  (x,search\_string,replace\_string) | Procura a string Search\_string em x e substitui por replace\_string. |
| rpad  (column\expression,n,’string’) | Retorna uma string com tamanho total de n alinhada à esquerda. |
| substr  ( column\expression,  m[n]) | Retorna os caracteres especificados a partir da string de caracteres, começando na posição m, com tamanho de n caracteres. Se m for negativo ,a contagem inicia a partir do final da string. Se n for omitido são retornados todos os caracteres até o final da string. |
| upper (column\expression) | Converte strings de caracteres para maiúsculas. |

EX.:

SELECT Concat (COD\_ALUNO,NOME) FROM TALUNO; --- concatena strings ---   
SELECT InitCap (NOME) FROM TALUNO; ---- Primeiro caractere de cada string fica maiúsculo ---   
SELECT InStr (NOME,'R')FROM TALUNO; - diz em qual posição o caractere está---  
SELECT Length (NOME) FROM TALUNO;  
SELECT COD\_ALUNO, LPad(COD\_ALUNO,5,'0') FROM TALUNO; -- acrescenta caracteres a esquerda ---

**DEFINIÇÕES DA LINGUAGEM: LANGUAGE STATEMENTS**

|  |  |
| --- | --- |
| DML – SÃO UTILIZADOS PARA O GERENCIAMENTO DE DADOS DENTRO DE OBJETOS DO BANCO. | |
| SELECT | – RECUPERA DADOS DO BANCO DE DADOS. |
| INSERT | – INSERIR DADOS EM UMA TABELA. |
| UPDATE | – ATUALIZA OS DADOS EXISTENTES EM UMA TABELA. |
| DELETE | – EXCLUI REGISTROS DE UMA TABELA. |

PARA DELETAR UMA LINHA EXPECIFICA DA TABLE   
 DELETE FROM NOME DA TABELA  
WHERE CONDIÇÃO PARA A EXCLUSÃO COD\_ALUNO = 5;  
E ASSIM CONSIGO EXCLUIR UMA LINHA ESPECIFICA DA TABELA

EX COM UPDATE .:

**INSERT INTO TALUNO (COD\_ALUNO, NOME, CIDADE, CEP)  
VALUES (SEQ\_ALUNO.NEXTVAL,'antonio','PETROLINA','45613216');  
  
---- ULTILIZEI O UPDATE PARA CONSERTAR ----   
  
UPDATE TALUNO SET  
NOME='ANTONIO'  
 WHERE COD\_ALUNO=24;**

|  |  |
| --- | --- |
| DDL - SÃO USADAS PARA DEFINIR A ESTRUTURA DE BANCO DE DADOS OU ESQUEMA. | |
| CREATE | PARA CRIAR OBJETOS NO BANCO DE DADOS, O PRÓPRIO BANCO DE DADOS, TABELAS, INDEXES, PROCEDURES, VIEWS, FUNCTIONS E TRIGGERS. |
| ALTER | ALTERA A ESTRUTURA DA BASE DE DADOS, O PROPRIO BANCO DE DADOS, TABELAS, INDEXES. |
| DROP | APAGA OBJETO DO BANCO DE DADOS, O PROPRIO BANCO DE DADOS, TABELAS INDEXES, PROCEDURES, VIEWS, FUNCTIONS E TRIGGERS. |
| TRUNCATE | REMOVE TODOS OS REGISTROS DE UMA TABELA, INCLUINDO TODOS OS ESPAÇOS ALOCADOS PARA OS REGISTROS SÃO REMOVIDOS. |
|  |  |

-- criei um campo nascimento e usando o tipo DATE e por DEFAULT SYSDATE - 3000 PEGANDO TODOS OS

-- ALUNOS DESSA TABELA TALUNO E FAZENDO DECRESCER 3000 DIAS PARTINDO DA DATA ATUAL---

**EX.:  
ALTER TABLE TALUNO ADD NASCIMENTO DATE DEFAULT SYSDATE -3000;**

|  |  |
| --- | --- |
| DCL – SÃO USADAS PARA DEFINIR ACESSO/CONTROLE DOS DADOS /OBJETOS | |
| GRANT | ATRIBUI PRIVILEGIOS DE ACESSO DO USUARIO A OBJETOS DO BANCO DE DADOS. |
| REVOKE | REMOVE OS PRIVILÉGIOS DE ACESSO AOS OBJETOS OBTIDOS COM O COMANDO GRANT. |

|  |  |
| --- | --- |
| TCL – SÃO USADOS PARA GERENCIAR AS MUDANÇAS FEITAS POR INSTRUÇÕES DML. ELA PERMITE QUE AS DECLARAÇÕES A SEREM AGRUPADAS EM TRANSAÇÕES LÓGICAS. | |
| COMMIT | – SALVAR O TRABALHO FEITO. ( Confirmando as alterações feitas em (INSERT, DELETE, UPDATE)). |
| SAVEPOINT | – IDENTIFICAR UM PONTO EM UMA TRANSAÇÃO PARA QUE MAIS TARDE VOCÊ PODE EFETUAR UM ROLLBACK. |
| ROLLBACK | – RESTAURAR BANCO DE DADOS AO ORIGINAL DESDE O ULTIMO COMMIT.( DESFAZ ALTERAÇÕES REALIZADAS (INSERT, UPDATE E DELETE). |
| JOINS | **–** Esta clausula é usada para combinar as linhas de duas ou mais tabelas baseado no relacionamento entre elas. |
| (INNER) JOIN: | Retorna registros que têm valores correspondentes em ambas as tabelas |
| LEFT(OUTER)JOIN: | Retorna todos os registros da tabela à esquerda e os registros correspondentes da tabela da direita. |
| RIGTH (OUTER) JOIN: | Retorna toso os registros da tabela da direita e os registros correspondentes da tabela da esquerda. |
| FULL (OUTER) JOIN: | Retorna todos os registros quando houver uma correspondência na tabelas |

**JOINS**Interface gráfica do usuário

Descrição gerada automaticamente com confiança médiaServe para acessar dados a partir de mais de uma tabela.

**PRODUTO CARTESIANO**

Um produto cartesiano é formado quando:

* Uma condição de join é omitida
* Uma condição de join invalida
* Todas as linhas da primeira tabela são unidas à todas as linhas da segunda tabela

Para evitar um produto cartesiano, sempre inclua uma condição de join válida na clausula WHERE.

* Utilizar prefixos de tabela para qualificar nomes de colunas que estão em múltiplas tabelas.
* Aumentar a performance utilizando prefixos de tabela.
* Fazendo distinção de colunas que possuírem nomes idênticos mas residem em tabelas diferentes utilizando alias de coluna

**SUBQUERYS – CONSULTAS**Uma Subconsulta (ou mais conhecida, subquery) é uma instrução SELECT que está condicionada dentro de outra instrução SQL.

Ex.:  
SELECT A.ID\_PACIENTE,  
A.NOME\_PACIENTE,  
A.NM\_BAIRRO,  
(SELECT B.NOME\_UNID\_INTER FROM UNID\_INTER B WHERE B.ID\_UNID\_INTER = 4) AS INTERNADO  
FROM PACIENTES A;

------------------------------------------------------------------------------------------------------------------------------

**Funções de Agregação**

**As funções de agregação executam um cálculo em um conjunto de valores e retornaram um único valor.**

**Com exceção de COUNT, as funções de agregação ignoram valores nulos.**

**As funções de agregação normalmente são usadas com a cláusula GROUP BY da instrução SELECT.**

**As funções de agregação podem ser usadas como expressões apenas nos seguintes casos:  
Alista de seleção de uma instrução SELECT (uma subconsulta ou uma consulta externa).**

**Uma cláusula HAVING.**

-- CASE ---  
SELECT NOME,ESTADO,  
CASE  
WHEN ESTADO = 'RS' THEN 'GAUCHO' --- WHEN = QUANDO O ESTADO FOR = A "RS" ENTÃO = 'GAUCHO'  
WHEN ESTADO = 'AC' THEN 'ACREANO' --- WHEN = QUANDO O ESTADO FOR = "AC" ENTÃO = 'ACREANO'  
WHEN ESTADO = 'RJ' AND SALARIO >500 THEN 'CARIOCA' --- WHEN = QUANDO O ESTADO FOR = "RJ" E SALARIO MAIOR QUE 500 ENTÃO = 'CARIOCA'ELSE 'OUTROS' --- CASO NÃO SEJA NENHUM DOS ACIMA = 'OUTROS'  
END AS APELIDO ---- FIM  
FROM TALUNO; --DA TABLE TALUNO

|  |  |
| --- | --- |
| Função | Campos |
| AVG | **Retorna média dos valores em um grupo. Valores nulos são ignorados** |
| MIN | **Retorna o valor mínimo na expressão. Pode ser seguido pela cláusula OVER** |
| MAX | **Retorna o valor máximo na expressão.** |
| SUM | **Retorna a soma de todos os valores ou somente os valores DISTINCT na expressão. SUM pode ser usado exclusivamente com colunas numéricas. Valores nulos são ignorados.** |
| COUNT | **Retorna o número de itens de um grupo.** |
| STDDEV | **Retorna o desvio padrão estatístico de todos os valores da expressão especificada** |
| VARIANCE | **Determina a variância de n, ignorando valores nulos.** |

SELECT NOME, ESTADO,  
Decode(ESTADO, 'RS', 'GAUCHO',  
'AC', 'ACREANO',  
'SP', 'PAULISTA',  
 'PE', 'PERNAMBUCANO',  
 'OUTROS') AS APELIDO  
FROM TALUNO;

SELECT NOME,SALARIO,  
CASE  
WHEN NOME IS NULL THEN 'INDIGENTE'  
WHEN SALARIO >620 THEN 'RICO'  
ELSE 'ESTAGIARIO'  
END AS APELIDO  
FROM TALUNO;