Base de Dados Relacional

Uma base de dados relacional obriga ao uso de chaves primárias.

**Chaves Primárias:**

São colunas únicas que identificam ima linha de colunas.

**Chaves Estrangeiras:**

São atributos (colunas) de uma tabela que é chave primária de outra tabela.

**Null:**

Sempre que não existe informação, o valor a colocar é null.

Conceitos

**Dados e Informação**

* **Dados:**

José 50 Leiria.

* **Informação:**

O José tem 50 anos e mora em Leiria.

**Base de Dados**

* Permite armazenamento de grandes quantidades de dados.
* Possibilita realizar com rapidez um conjunto de operações simples de processamento de dados.

**Funções do Sistema de Gestão de Base de Dados** (SGBD)

* Armazenamento, pesquisa e atualização de dados.
* Acessibilidade e catálogo do sistema.
* Suportar transações e controlar concorrência.
* Permitir a recuperação.
* Definição de mecanismos de autorização.
* Suportar a comunicação de dados.
* Manutenção da integridade.
* Promover a independência dos dados.
* Disponibilizar outros serviços de utilidade.

Texto alternativo gerado automaticamente:
1 
António Freitas 
Leiria 
244888888 
2 
Manuel da Silva 
Lisboa 
212323238 
Maria de Sousa 
Porto 
2234567859 
Ficheiro 
Re isto 
Cam o 

Imagem sem título.png Texto alternativo gerado automaticamente:
Tabela CLIENTES 
N Cliente 
2 
Coluna 
100 
Nome 
António Freitas 
Manuel da Silva 
Maria de Sousa 
Morada 
Leil'la 
Lisboa 
Poito 
Telefone 
123456 
111222 
121212 
Linha 
Estrutura lógica de armazenamento


Técnica de Normalização

É um conjunto de 6 regras para garantir a qualidade dos dados
6 Formas Normais = (6 Regras de Técnicas de Normalização)

1ª Forma Normal 
18 de outubro de 2018
20:21


|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 1 FN 2 FN 3FN FNBC - Boyce Code 4 FN 5FN Desenhos a Tinta Digital São o nome das regras por ordem de Importância |  |  |
|  |  | São o nome das regras por ordem de Importância |
|  |  |  |

Nunca criar redundância de dados.
Assim a probabilidade de erros diminui.
Repetido ≠ Redundante



Acesso Servidor das Aulas PL

**PC do Laboratório**

**Nome Utilizador:** EI*<nº do estudante>*@BDados

**Password:** EI*<nº do estudante>*

**Máquina Virtual**

**Nome Utilizador:** Aulas

Texto alternativo gerado automaticamente:
Servidor BD 

Funções de Grupo

**Clausulas de grupo**

(Agrupar dados)

SELECT … --> ()

FROM …

WHERE … --> (Restringe os dados a apresentar)

**GROUP BY … --**> (Colunas pelas quais queremos agrupar)

**HAVING … -->** (Restringir cálculos dos grupos, (não da tabela!))

ORDER BY …

**HAVING** permite aplicar funções por restrições de grupo.

Quando se agrupa por coluna, é criado um valor distinto daquela coluna.

(Não faz sentido usar o **HAVING** sem o **GROUP BY**).

**Funções de grupo**

**COUNT**

(\*) ----> Conta o numero de linhas dentro de cada grupo.

SELECT COUNT(\*)

FROM empregados;

(coluna) --> Conta os valores que estão preenchidos naquela coluna (Não conta os NULL).

SELECT COUNT(chefe)

FROM empregado;

(distinct) -> Conta os valores diferentes existentes naquela coluna.

SELECT COUNT(DISTINCT chefe)

FROM empregado;

**SUM**

(coluna) ----> Soma todos os valores daquela coluna.

(AVG) --> Calcula a média dos valores que especificada.

(MIN) -> Procura o menor valor da coluna.

(MAX) -> Procura o maior valor da coluna.

A partir do momento que que fazemos **GROUP BY**, a tabela apenas conta com os grupos.

Ou seja, as colunas do **SELECT** deixam de aparecer no resultado, apenas a coluna do **GROUP BY**.

**Exemplo:**

Empregado

|  |  |  |
| --- | --- | --- |
| NUMEMP | NUMDEP | SALARIO |
| 1000 | 10 | 800 |
| 1001 | 10 | 1000 |
| 1002 | 10 | 1000 |
| 1003 | 20 | 800 |
| (1004 | 20 | 800 |

**SELECT** numdep, **COUNT**(\*),  **SUM**(salario)

**FROM** empregado

**GROUP BY** numdep

**HAVING SUM**(salario) >2000;

**Ordem de processamento dos comandos**

1º - FROM

2º - WHERE

3º - GROUP BY

4º - HAVING

5º - SELECT

6º - ORDER BY

Consultas Complexas

SELECT …

FROM … (SELECT … )

WHERE … --> (SELECT …)

**GROUP BY …**

**HAVING … -->** (SELECT …)

ORDER BY …

SELECT numemp, nomeemp

FROM empregado

WHERE salario > **(SELECT AVG(salario)**

**FROM empregado)**

ORDER BY … ;

**SELECT**

**FROM** tabelas …

**[WHERE** condições… (Subconsultas)**]**

**[GROUP BY** nome colunas, **]**

**[HAVING** condições… (Subconsultas)**]**

**[ORDER BY** …**]**

**;**

**Resultados de:**

|  |  |
| --- | --- |
| **1 só valor** | **Mais que 1 valor** |
| = | IN |
| !=, <> | NOT IN |
| <, <=, >, >= | ALL ou ANY |

Manipulações na Base de Dados

**DML**

* INSERT
* UPDATE
* DELETE

**DLC**

* COMMIT - Aplicar alterações
* ROLLBACK - Reverter alterações em memória

Para que as alterações do INSERT sejam aplicadas, é necessário a inserção do comando COMMIT .

Até lá as alterações apenas estão em memória.

PL/SQL

Todos os comandos terminam sempre com "**;**"

**Blocos de código**

BEGIN

<…>

END;

**Atribuir valores**

:=

**Verificação**

=

**Declarar variáveis**

Declare

Var1 INTEGER;

Var1 t\_aluno.ano%TYPE;

**Declarar variáveis que recebem valores de tabela**

(Dizer que o tipo da var é igual ao tipo de valor que a BD devolve)

Var1 t\_aluno.ano%TYPE;

**Procurar na base de dados**

SELECT curso, ano

**INTO var1, var2**

FROM t\_aluno

WHERE id=1000;

**Escrita no ecrã**

DBMS\_OUTPUT.PUT\_LINE('O valor é ' || <nome\_var>);

|  |  |  |  |
| --- | --- | --- | --- |
|  | Titulo |  |  |
|  | 1º CONSTRAINS  2º TRIGGERS (DML)  Before  ou  After |  |  |
|  |  | CREATE SQUENCE value  [START WITH value]  Endereço [INCREMENT BY value]  [MAXVALUE] |

GERAR ERRO

RAISEAPPLOCATION.ERROR