Criar tabelas

CREATE TABLE nome\_da\_tabela

(

Nome\_da\_coluna1 tipo\_de\_dado [NULL|NOT NULL],

Nome\_da\_coluna2 tipo\_de\_dado [NULL|NOT NULL],

Nome\_da\_coluna3 tipo\_de\_dado [NULL|NOT NULL],

…

restrições...

);

Tipos de dados básicos

| CHAR (tamanho) | Sequência de caracteres de tamanho fixo. |
| --- | --- |
| VARCHAR (tamanho) | Sequência de caracteres de tamanho variável. |
| NUMERIC / DECIMAL /  NUMBER (total, decimais) | Armazena um valor, com ou sem  decimais. |
| DATE / DATETIME | Armazena uma data (pode incluir  também a hora). |

Inserindo dados nas tabelas

INSERT INTO nome\_da\_tabela \* Não há garantia da ordem das colunas

VALUES ('dado',dado, dado);

INSERT INTO nome\_da\_tabela (coluna1,coluna2,coluna3)

VALUES ('dado',dado, dado);

Consultando dados

Consultar todos os dados

SELECT \* FROM nome\_da\_tabela;

Consultar os dados das colunas da tabela

SELECT coluna1, coluna2, ...

FROM nome\_da\_tabela

[WHERE condicao];

Alterando dados da tabela

UPDATE nome\_da\_tabela

SET campo1 = valor1;

* O problema de usar o comando acima é que ele se aplica a toda a tabela.

UPDATE nome\_da\_tabela

SET campo1 = valor1;

WHERE campo1 = valor2;

Podemos usar também expressões aritméticas e lógicas (V/F).

Exemplos:

* SELECT com preco \* 2
* SELECT e WHERE com idade > 18
* SELECT e WHERE com idade > 18 AND peso < 80
* SELECT e WHERE com idade <> 18
* UPDATE com x = x + 10
* UPDATE com preco = preco \* 1.1

Excluindo registros da tabela

DELETE FROM nome\_da\_tabela

WHERE condição;

Ordenando dados

SELECT coluna1, coluna2, ...

FROM nome\_da\_tabela

[WHERE condição]

ORDER BY coluna1 [ASC | DESC], coluna2 [ASC | DESC]...

* Usamos ASC para ordenar do menor para o maior (ascendente) ou DESC para ordenar do maior para o menor (descendente). O padrão é ASC.

Também é possível ordenar por mais de uma coluna

SELECT coluna1, coluna2

FROM nome\_da\_tabela

ORDER BY coluna DESC, modelo ASC;

Contando Registros

SELECT COUNT(\*)

FROM nome\_da\_tabela;

SELECT COUNT(\*)

FROM nome\_da\_tabela

WHERE coluna = condição;

Evitando duplicatas

SELECT DISTINCT coluna

FROM nome\_da\_tabela;

Valores nulos

CREATE TABLE PESSOAS

(

Cpf VARCHAR(20)NOT NULL,

Nome VARCHAR(150) NOT NULL,

idade NUMBER(3) NULL,

endereco VARCHAR(150)

--o campo endereço é NULL implicitamente

);

* Para especificar que um campo é obrigatório indicamos NOT NULL, e para especificar que um campo é de conteúdo opcional indicamos NULL.

Operadores LIKE e IN

* Pessoas com nomes iniciando com a letra 'A':

SELECT \*

FROM PESSOAS

WHERE nome LIKE 'A%';

* Pessoas com nomes iniciando com 'Ana':

SELECT \*

FROM PESSOAS

WHERE nome LIKE 'Ana%';

* O símbolo ‘\_’ substitui exatamente um caractere.

--Pode corresponder a Maria ou Mario:

SELECT \*

FROM PESSOAS

WHERE nome LIKE 'Mari\_ da Silva';

* O operador IN determina se um valor corresponde a qualquer um dos valores de uma lista:

--Pessoas que tenham 25, 30 ou 40 anos de idade:

SELECT \*

FROM PESSOAS

WHERE idade IN (25, 30, 40);

Removendo e adicionando novas colunas na tabela

Remover

ALTER TABLE PESSOAS

DROP COLUMN idade;

Adicionar

ALTER TABLE PESSOAS

ADD sexo CHAR(1);

Manipulando datas

CREATE TABLE PESSOAS

(

cpf VARCHAR(20) NOT NULL,

nome VARCHAR(150) NOT NULL,

datanasc DATE NULL,

endereco VARCHAR(150) NULL

);

Inserindo datas em um campo DATE

INSERT INTO PESSOAS (cpf, nome,datanasc, endereco)

VALUES('29048','Roberto','03-FEB-1980','Rua D, 80'

);

--Formato alternativo (padrão ANSI YYYY-MM-DD

):

--Deve-se acrescentar a palavra DATE antes da data.

INSERT INTO PESSOAS (cpf, nome,datanasc, endereco)

VALUES('29048','Roberto', DATE'1980-02-03','Rua D, 80'

);

Funções de conversão de datas

Aritmética de datas

Em SQL é possível realizar as seguintes operações sobre datas:

* DATE + NUMBER = DATE
* DATE – NUMBER = DATE
* DATE – DATE = número de dias entre as datas

SELECT SYSDATE + 1

getdate();

Integridade de entidade

* Chamamos a chave principal de chave primária (ou, em inglês, PRIMARY KEY - PK). As Outras Chaves Candidatas, Então, Se Tornam Chaves Alternativas (ou, Em Inglês, ALTERNATE KEYS -AKs).
* Em SQL usamos as restrições PRIMARY KEY (para a principal) e UNIQUE (para as demais AKs). Naturalmente, toda PK é também NOT NULL.

CREATE TABLE ALUNOS

(

nroMatricula VARCHAR(10)PRIMARY KEY,

cpf VARCHAR(20) UNIQUE,

email VARCHAR(100) UNIQUE,

Nome VARCHAR(150) NOT NULL,

anoIngresso NUMBER(4) NOT NULL,

endereco VARCHAR(150) NULL,

sexo CHAR(1) NOT NULL

);

* A forma alternativa que usa a palavra CONSTRAINT.

CREATE TABLE ALUNOS

(

nroMatricula VARCHAR(10)NOT NULL,

Cpf VARCHAR(20)NOT NULL,

Email VARCHAR(100)NULL,

Nome VARCHAR(150) NOT NULL,

anoIngresso NUMBER(4)NOTNULL,

endereco VARCHAR(150) NULL,

sexo CHAR(1) NOT NULL,

CONSTRAINT PK\_ALUNOS PRIMARY KEY (nroMatricula),

CONSTRAINT AK1\_ALUNOS UNIQUE (cpf),

CONSTRAINT AK2\_ALUNOS UNIQUE (email)

);

Integridade de dominio

A integridade de domínio visa garantir que os dados Armazenados respeitem determinados valores permitidos. Podemos restringir o intervalo de dados permitido para um campo. Alguns exemplos de restrição de domínio são:

* garantir que o preço de um produto não pode ser zero ou ter um valor
* negativo;
* garantir que o campo status de um pedido tenha somente um dos seguintes valores: ‘ABERTO’, ‘PENDENTE’, ‘FECHADO’.
* garantir que o campo gênero somente aceite os valores ‘M’, ‘F’ ou ‘NB’.

ALTER TABLE ALUNOS

ADD CONSTRAINT CK\_AnoIngr CHECK (anoIngresso > 2000);

ALTER TABLE ALUNOS

ADD CONSTRAINT CK\_sexo CHECK (sexo IN ('M', 'F', ‘NB’));

Integridade referencial

* A integridade referencial é usada entre duas tabelas para garant ir que os dados de uma coluna da primeira tabela se referem aos dados registrados em uma coluna da segunda tabela.

CREATE TABLE ESTADOS

(

Uf CHAR(2)NOT NULL,

Nome VARCHAR(40) NOT NULL,

regiao CHAR(2) NOT NULL,

CONSTRAINT PK\_ESTADOS PRIMARY KEY (uf)

);

CREATE TABLE CIDADES

(

cod\_cidade NUMBER(4)NOT NULL,

Nome VARCHAR(60) NOT NULL,

uf CHAR(2) NOT NULL,

CONSTRAINT PK\_CIDADES PRIMARY KEY (cod\_cidade,

CONSTRAINT FK\_EST\_CID FOREIGN KEY (uf)REFERENCES ESTADOS(uf)

);

* Assim não se pode cadastrar uma cidade com estado xx por exemplo.

Consultas sobre múltiplas tabelas

Para consultar dados de uma ou mais tabelas relacionadas devemos utilizar operações denominadas “junções” (JOINs). A junção de duas ou mais tabelas é equivalente – em termos de resultado final – à realização do produto cartesiano, comparando o valor de certos atributos, e aplicando uma projeção e uma seleção ao resultado.

EQUI - JOINs

Um EQUI - JOIN relaciona linhas de uma tabela com as linhas de outra tabela, a partir de um critério de igualdade (ou equivalência). Normalmente essa igualdade ocorre entre uma PK (da tabela pai) e uma FK (da tabela filho).

SELECT EST.uf, EST.nome, CID.uf, CID.nome

FROM ESTADOS EST INNER JOIN CIDADES CID

ON EST.uf = CID.uf;

Exemplos:

Forma mais antiga

SELECT EST.uf, EST.nome, CID.uf, CID.nome

FROM ESTADOS EST,CIDADES CID

WHERE EST.uf = CID.uf;

Forma mais moderna (preferida)

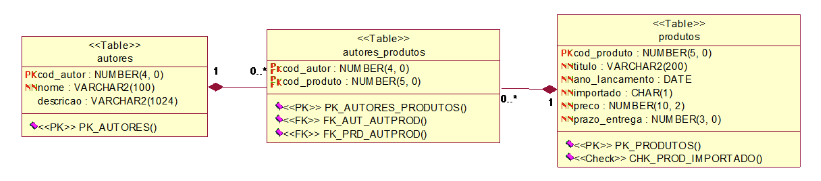
SELECT EST.uf, EST.nome, CID.uf, CID.nome

FROM ESTADOS EST INNER JOIN CIDADES CID

ON EST.uf = CID.uf;

Consultando dados a partir de mais de duas tabelas

Join encadenados



SELECT AU.nome, PROD.titulo

FROM AUTORES AU

JOIN AUTORES\_PRODUTOS AP

ON (AU.cod\_autor = AP.cod\_autor)

JOIN PRODUTOS PROD

ON (AP.cod\_produto = PROD.cod\_produto);

Precedência de JOINs

* A fim de melhor controlar a precedência de JOINs encadeados, devem ser utilizados parênteses

SELECT USU.nome, TEL.ddd, TEL.numero

FROM

USUARIOS USU JOIN

(CLIENTES CLI ON USU.COD\_USUARIO = CLI.COD\_CLIENTE

JOIN TELEFONES TEL

ON CLI.COD\_CLIENTE = TEL.COD.CLIENTE;

Visões

Uma visão é uma espécie de tabela virtual criada a partir de uma consulta, a fim de representar a visão de um usuário ou de uma aplicação sobre o banco de dados.

Uma visão pode ser consultada da mesma forma que uma tabela, mas sua atualização está sujeita a certas regras. Uma visão não armazena dados ou cópias dos dados.

Visões podem ser definidas para:

* Definir a percepção que um usuário ou aplicação possui do banco de dados;
* Simplificar a construção de consultas, definindo consultas frequentes ou complexas;
* Possibilitar a independência de dados;
* Fornecer segurança de acesso.

CREATE OR REPLACE VIEW AS

consulta

Exemplo

CREATE VIEW DADOSCLIENTES AS

SELECT C.cod\_cliente, U.nome, U.cpf, U.email,

U.username, C.data\_nascimento, C.data\_cadastro

FROM USUARIOS U INNER JOIN CLIENTES C

ON U.cod\_usuario = C.cod\_cliente;

Selecionar dados de uma view como fosse de uma tabela

SELECT nome, email, data\_cadastro

FROM DADOSCLIENTES

WHERE email LIKE ‘%@pucrs.br’

ORDER BY data\_cadastro DESC, nome ASC

Para deletar uma view

DROP VIEW DADOSCLIENTES

Funções

* Funções sobre linhas, as quais operam sobre cada linha do resultado individualmente.
* Funções sobre conjuntos de linhas, que operam sobre diversas linhas, calculando valores sobre todo o conjunto (para determinar totais, médias, o maior valor, entre outras possibilidades).

Funções de Agregação

Uma função de agregação (ou função agregada) é uma função que opera sobre um conjunto de linhas. As funções de agregação permitem calcular:

* Valores totais para toda uma tabela.
* Subtotais para toda uma tabela, agrupando o resultado por determinado atributo e apresentando-o como uma nova coluna.

SELECT função\_agregada

FROM nome\_da\_tabela

[...]

As funções agregadas mais comuns são:

SELECT AVG(preco) MEDIA FROM PRODUTOS;

SELECT AVG(NVL(preco,0)) MEDIA FROM PRODUTOS;

SELECT MAX(preco) FROM PRODUTOS;

SELECT COUNT(\*) NUM\_CLIENTES FROM CLIENTES;

SELECT COUNT(ddd) FROM TELEFONES;

* A função NVL()converte valores nulos em um valor computável. Compare os resultados dos dois primeiros exemplos e tente identificar a diferença.

Agrupamento por GROUP BY

SELECT nome\_da\_coluna [, ...], função\_agregada [, ...]

FROM nome\_da\_tabela [, ...]

GROUP BY [ALL] nome\_da\_coluna [,...]

ORDER BY colunas

* O operador ALL inclui no resultado todos os grupos, incluindo aqueles que não atendem às condições de busca;
* A(s) coluna(s) contidas na cláusula SELECT deve(m) estar todas obrigatoriamente na cláusula GROUP BY;
* As colunas da cláusula GROUP BY não precisam estar na cláusula SELECT;
* Pode-se agrupar também por mais de uma coluna;
* A cláusula ORDER BY não é obrigatória, mas é bastante comum nesses casos porque organiza o resultado da consulta.

Agrupamento por GROUP BY e HAVING

A cláusula HAVING é usada em conjunto com a cláusula GROUP BY. Ela

determina as condições sobre as quais será realizada a composição dos grupos . Em outras palavras, a cláusula HAVING serve para decidir quais dos grupos gerados farão parte do resultado final. Os grupos que não satisfizerem as condições da cláusula HAVING são descartados.

SELECT nome\_da\_coluna [, ...], função\_agregada [, ...]

FROM nome\_da\_tabela [, ...]

GROUP BY [ALL] nome\_da\_coluna [,...]

HAVING condições

ORDER BY colunas

* As novas colunas geradas pelo cálculo das funções agregadas podem ser referidas na cláusula HAVING.