



BANCO DE DADOS RELACIONAL

Azriel Majdenbaum – Aula 03

Professores

CLAUDIO BONEL

Professor Convidado

Claudio Bonel é doutorando e mestre em Educação, especialista em Sistemas de Informação, licenciado em Informática e tecnólogo em Marketing. Atua no mercado de TI desde 1996, especificamente, na área de Engenharia, Ciência e Análise de Dados com participação em projetos relevantes como a implantação do bilhete eletrônico na malha ferroviária do Rio de Janeiro e o IPO da Empresa Magazine Luíza, além de projetos com grandes empresas do Brasil, como Coca-cola, Icatu Seguros, Petrobrás e Transpetro. No exterior, trabalhou com a BAT na África do Sul, Canadá e Argentina. Atualmente, é diretor de tecnologias e treinamentos de uma consultoria, professor de pós-graduação da Escola Gestão em Políticas Públicas do Estado Rio de Janeiro, do Centro de Tecnologia da Informação e Comunicação e da Faculdade de Tecnologia SENAC/RJ. É autor de livros sobre Análise de Dados, Microsoft Most Valuable Professional, na Plataforma de Dados (premiação Microsoft), palestrante e fundador do projeto social Dado Humanizado que atua na capacitação e mentoria de jovens das favelas do Rio de Janeiro em tecnologias essenciais ao mercado de trabalho e apoio às famílias.

AZRIEL MAJDENBAUM

Professor PUCRS

Doutor em Administração pela Pontifícia Universidade Católica do Rio Grande do Sul e Mestre em Administração pela Universidade Universidade Federal do Rio Grande do Sul; Especialização em Administração Hospitalar pela PUCRS; Bacharel em Informática pela PUCRS. Profissional com 25 anos de experiência, atuou em posições executivas e técnicas. Experiência em gestão de Tecnologia, desenvolvimento, testes e implantação de sistemas corporativos, formação e coordenação de equipes, administração de recursos e orçamentos de Informática. Liderou diversos projetos de otimização empresarial, através de iniciativas de Redesenho de Processos / Organização e Implantação de Sistemas de Gestão.

Ementa da disciplina

Visão geral da abordagem de banco de dados. Estudo sobre modelagem conceitual (E/R). Estudo sob mapeamento objeto relacional (ORM). Desenvolvimento com SQL padrão (DDL e DML).

Pós graduação em Desenvolvimento Full Stack

Por Azriel Majdenbaum

Banco de Dados Relacional

Por Azriel Majdenbaum

Organização da aula

- Introdução
 - O ambiente de trabalho
 - Modelo Relacional
- Manipulação Básica de Dados Parte I
- Manipulação Básica de Dados Parte II
- Manipulação Básica de Dados Parte III
- Manipulação Básica de Dados Parte IV
- Manipulação Avançada de Dados
- Considerações finais

Introdução

- Ambiente de desenvolvimento
- Modelo Relacional
- **Manipulação Básica de Dados Parte I**
 - Criando a sua primeira tabela
 - Inserindo dados na tabela VEICULOS
 - Consultando dados a partir da tabela VEICULOS
 - Alterando dados na tabela VEICULOS
 - Excluindo dados da tabela VEICULOS
 - Ordenando dados
 - Contando registros
 - Evitando duplicatas
- **Manipulação Básica de Dados Parte II**
- **Manipulação Básica de Dados Parte III**
- **Manipulação Básica de Dados Parte IV**
- **Manipulação Avançada de Dados**
- **Considerações finais**

Introdução

- Ambiente de desenvolvimento
- Modelo Relacional
- Manipulação Básica de Dados Parte I
- Manipulação Básica de Dados Parte II
 - Valores nulos
 - Operadores Like e IN
 - Manipulação de datas
- Manipulação Básica de Dados Parte III
- Manipulação Básica de Dados Parte IV
- Manipulação Avançada de Dados
- Considerações finais

Introdução

- Ambiente de desenvolvimento
- Modelo Relacional
- Manipulação Básica de Dados Parte I
- Manipulação Básica de Dados Parte II
- Manipulação Básica de Dados Parte III
 - Restrições de integridade (Entidade, Domínio e Referencial)
 - Relacionamento muitos para muitos
- Manipulação Básica de Dados Parte IV
- Manipulação Avançada de Dados
- Considerações finais

Introdução

- Ambiente de desenvolvimento
- Modelo Relacional
- Manipulação Básica de Dados Parte I
- Manipulação Básica de Dados Parte II
- Manipulação Básica de Dados Parte III
- Manipulação Básica de Dados Parte IV
 - Junções de Tabelas:
 - Inner join
 - Outer join
- Manipulação Avançada de Dados
- Considerações finais

Introdução

- Ambiente de desenvolvimento
- Modelo Relacional
- Manipulação Básica de Dados Parte I
- Manipulação Básica de Dados Parte II
- Manipulação Básica de Dados Parte III
- Manipulação Básica de Dados Parte IV
- **Manipulação Avançada de Dados**
 - Funções de agregação
 - Group by
 - Having
 - Subconsulta
 - Indexação
 - Auto incremento
- **Considerações finais**

Introdução

- Ambiente de desenvolvimento

- Soluções online

- Reduzem a complexidade de instalações
 - Garantem ambiente estável
 - Normalmente possuem exemplos base

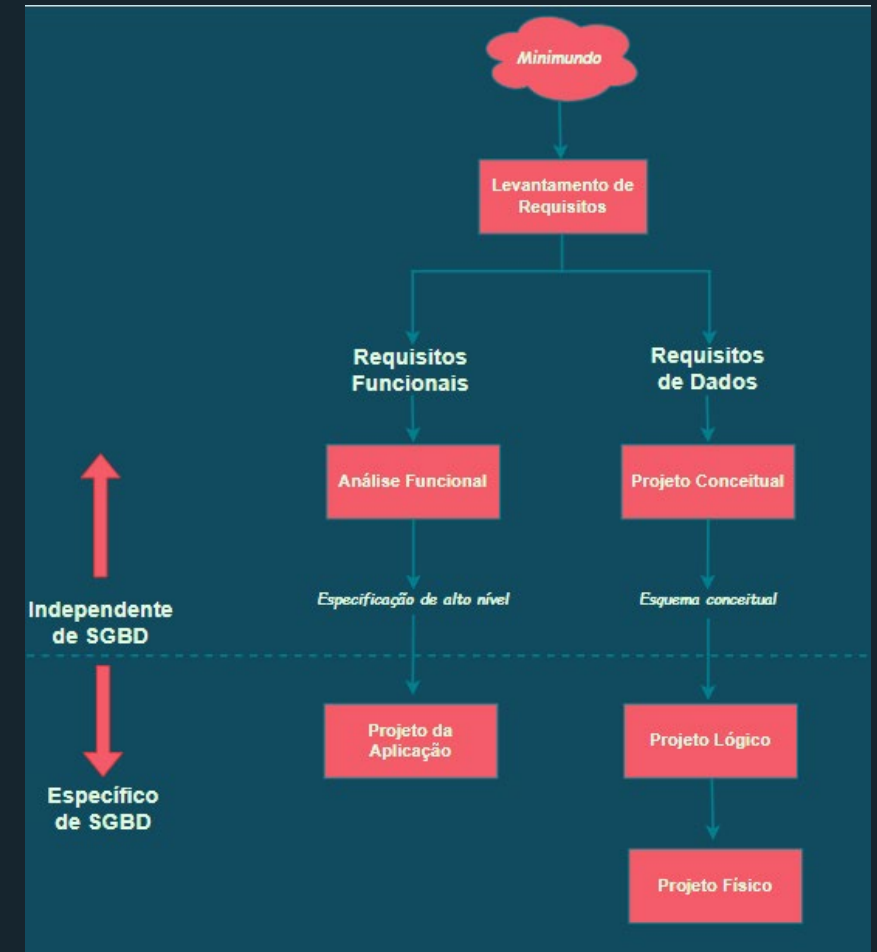
- Nossa proposta

- livesql.oracle.com
 - <https://www.brmodeloweb.com/lang/pt-br/index.html>
 - Download brModelo 3.2 (<https://sourceforge.net/projects/brmodelo/>)



Modelo relacional

- É um Modelo Lógico de Dados, originalmente proposto por Edgar Frank Codd, caracterizado por:
 - Independência de Dados: mudanças no esquema interno não afetam o esquema conceitual e mudanças no esquema conceitual não afetam as aplicações;
 - Linguagem Estruturada de Consulta (**SQL**);
 - Redução da Redundância;
 - Simplicidade na representação dos dados;
 - Simplicidade na representação dos resultados das consultas sobre os dados.



(Baseado em ELMASRI, NAVATHE, 2012, p. 133)

Modelo relacional – Conceitos Formais

- **Domínio**: dado uma característica de um objeto da realidade, por exemplo, o número de matrícula de um aluno, o domínio deste elemento de dado é o conjunto de todos os números de matrícula possíveis;
- **Relação**: um objeto da realidade pode ser representado, de forma abstrata, por um conjunto de elementos de dados definidos, cada um, sobre um domínio.

Exemplo: Suponha uma relação chamada LIVROS, composta pelos seguintes elementos de dados, ou ATRIBUTOS, definidos por domínios específicos:

Código do livro	Numérico, 5 posições, obrigatório
Título	Texto, tam.200, obrigatório
Ano de lançamento	Data, obrigatória
Importado?	Booleano, S/N, obrigatório
Preço	Numérico, 10 posições, 2 decimais
Prazo de entrega	Numérico, 3 posições, obrigatório

Modelo relacional – Conceitos Formais

Assim, cada relação possui:

- **Um Cabeçalho**: composto pelo conjunto de ATRIBUTOS que a compõe;
- **Um Corpo**: composto por um conjunto de TUPLAS, cada qual correspondendo à representação abstrata de um objeto da realidade.

Adicionalmente:

- Uma TABELA armazena os objetos de uma classe da realidade;
- Uma tabela é composta por COLUNAS, as quais armazenam determinado atributo dos objetos da classe;
- Cada instância de uma classe (objeto) é armazenada como uma LINHA da tabela.



Tabela Estados

UF	NOME	REGIAO
AC	Acre	N
AL	Alagoas	NE
AP	Amapá	N
AM	Amazonas	N
BA	Bahia	NE
CE	Ceará	NE
DF	Distrito Federal	CO
ES	Espírito Santo	SE
GO	Goiás	CO
MA	Maranhão	NE
MT	Mato Grosso	CO
MS	Mato Grosso do Sul	CO

Modelo relacional – Conceitos Formais

A Linguagem SQL

O Modelo Relacional prevê, desde sua concepção, a existência de uma linguagem baseada em caracteres (interpretada) que suporte a definição do esquema físico (tabelas, restrições, etc.), e sua manipulação (inserção, consulta, atualização e remoção).

A Linguagem SQL (Structured Query Language) é padrão para os SGBDs Relacionais que seguem o padrão ANSI (American National Standards Institute).

A Linguagem SQL pode ser dividida em cinco conjuntos de comandos:

Recuperação de dados	Comando SELECT
Linguagem de definição de dados: DDL - Data Definition Language	Comandos para criação e manutenção de objetos do banco de dados: CREATE , ALTER , DROP , RENAME e TRUNCATE
Linguagem de manipulação de dados: DML - Data Manipulation Language	Comandos para inserções (INSERT), atualizações (UPDATE) e exclusões (DELETE)
Linguagem para controle de transações	Comandos COMMIT , ROLLBACK e SAVEPOINT
Linguagem para controle de acesso a dados	Comandos GRANT e REVOKE



Observações gerais sobre os comandos SQL:

- Não importa se são escritos em maiúsculas ou minúsculas;
- Podem ser escritos em uma só linha ou com quebras após qualquer palavra;
- Usualmente coloca-se cada cláusula de um comando em uma linha separada;
- Sugere-se usar tabulações e espaços para melhorar a clareza.

Manipulação Básica de Dados Parte I

Manipulação Básica de Dados – Create Table

Criando a sua primeira tabela no banco de dados

Para criar uma tabela no banco de dados, usamos o comando CREATE TABLE:

```
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 ...
);
```

Observação: os elementos entre colchetes são opcionais. Neste caso servem para dizer, para cada coluna, se o seu dado é obrigatório (**NOT NULL**) ou opcional (**NULL**). Esse conceito, assim como as restrições, será explorado mais adiante.

Manipulação Básica de Dados – Create Table

Para exemplificar, vamos criar uma tabela para armazenar informações sobre alguns **veículos**.

Experimente o seguinte comando no SGBD:

```
CREATE TABLE VEICULOS
(
  placa    CHAR(8) ,
  ano      NUMBER(4) ,
  km       NUMBER(6) ,
  marca    VARCHAR(50) ,
  modelo   VARCHAR(50) -- repare que não há virgula aqui
) ;
```

Tipos de dados básicos em SQL

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

Observação: os elementos entre colchetes são opcionais. Neste caso servem para dizer, para cada coluna, se o seu dado é obrigatório (**NOT NULL**) ou opcional (**NULL**). Esse conceito, assim como as restrições, será explorado mais adiante.

Manipulação Básica de Dados – Insert

Inserindo alguns dados na tabela VEICULOS

Agora vamos inserir alguns veículos no banco de dados. Para isso, usaremos o comando INSERT INTO.

```
INSERT INTO nome_da_tabela [(colunas)]  
VALUES (valores);
```

Exemplos.

```
INSERT INTO VEICULOS  
VALUES ('IJK-1212', 2022, 0, 'Chevrolet', 'Onix');
```

```
INSERT INTO VEICULOS (placa, ano, km, marca, modelo)  
VALUES ('IJM-1556', 2015, 72045, 'Volkswagen', 'Gol');
```

```
-- Isso é um comentário em SQL.  
-- Repare o uso de aspas simples para textos.  
-- Repare também na especificação das colunas no segundo caso.  
-- Insira pelo menos mais cinco veículos no BD.
```

Manipulação Básica de Dados – Select

Consultando dados a partir da tabela VEICULOS

Para consultar os veículos armazenados no banco de dados, usamos o comando SELECT:


```
SELECT coluna1, coluna2, ...  
FROM nome_da_tabela  
[WHERE condicao];
```

Para consultar todos os dados de uma tabela usamos o comando a seguir:

```
SELECT *  
FROM nome_da_tabela ;
```

Podemos também aplicar um filtro nas linhas e nas colunas retornadas pelo comando SELECT. Vamos consultar a placa e o ano dos veículos novos (zero km). Para isso, usamos a cláusula WHERE:

```
SELECT placa, ano  
FROM VEICULOS  
WHERE km = 0;
```

 Escreva o comando para selecionar a placa, o ano e o modelo dos veículos anteriores ao ano 2022. Experimente também variar a ordem das colunas.

Manipulação Básica de Dados – Update

Alterando dados na tabela VEICULOS

Para alterar uma informação no banco de dados, usamos o comando UPDATE:

```
UPDATE nome_da_tabela  
SET campo1 = valor1  
[WHERE condicao];
```

Por exemplo, se quiséssemos alterar a quilometragem dos veículos para 0 (zero), poderíamos usar o comando:

```
UPDATE VEICULOS  
SET km = 0;
```

Cuidado! O problema de usar o comando acima é que ele se aplica a toda a tabela, ou seja, a todos os registros armazenados. Como consequência, todos os veículos teriam o seu atributo km igual a 0!

Manipulação Básica de Dados – Update

Para que o comando UPDATE seja aplicado a somente um registro ou a alguns dos registros, também usamos a cláusula WHERE.

Por exemplo, poderíamos alterar o modelo do veículo cuja placa é IJK-1212 para “Cruze”:

```
UPDATE VEICULOS  
SET modelo = 'Cruze'  
WHERE placa = 'IJK-1212';
```

Verifique como ficou:

```
SELECT *  
FROM VEICULOS;
```

Manipulação Básica de Dados – Update

Também podemos alterar diversos registros de uma só vez. Vamos, por exemplo, estabelecer que todos os veículos com menos de 10 quilômetros rodados devam apresentar a sua quilometragem igual a zero:

```
UPDATE VEICULOS  
SET KM = 0  
WHERE KM < 10;
```

Verifique como ficou:


```
SELECT *  
FROM VEICULOS;
```


Manipulação Básica de Dados

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

Exemplos:

- `SELECT com preco * 2`
- `UPDATE com x = x + 10`
- `UPDATE com preco = preco * 1.1`
- `SELECT e WHERE com idade > 18`
- `SELECT e WHERE com idade > 18 AND peso < 80`
- `SELECT e WHERE com idade <> 18`
- *(escreva outras expressões aritméticas)*
- *(escreva outras expressões lógicas. Use AND, OR, NOT)*

 Experimente escrever o comando UPDATE para somar 100 quilômetros a todos os veículos cujos anos estão entre 2015 e 2021 (inclusive).

Manipulação Básica de Dados – Delete

Excluindo registros da tabela VEICULOS

Para excluir um ou mais registros de uma tabela, usamos o comando DELETE.

```
DELETE FROM nome_da_tabela  
[WHERE condicao];
```

Repare novamente na cláusula WHERE, que especifica quais registros deverão ser excluídos. **Cuidado:** se você omitir a cláusula WHERE, todos os registros da tabela serão excluídos!

(Note: Isso não irá excluir a tabela em si. A estrutura da tabela permanecerá intacta. Você inclusive poderá inserir outros registros posteriormente).


Manipulação Básica de Dados – Delete

Vamos excluir o veículo Gol, de 2015, cuja placa é IJM-1556:


```
DELETE FROM VEICULOS  
WHERE placa = 'IJM-1556';
```

Verifique como ficou:

```
SELECT *  
FROM VEICULOS;
```

 Experimente agora excluir todos os veículos da marca Chevrolet que possuem mais de 50.000 KM.

```
DELETE FROM VEICULOS  
WHERE (marca = 'Chevrolet') AND (km > 50000);
```

 O uso dos parênteses não é obrigatório, mas facilita a leitura do comando.

Manipulação Básica de Dados – Order BY

A ordem dos registros de um comando SELECT não é garantida. No entanto, se quisermos colocá-los em alguma ordem específica, podemos usar a cláusula ORDER BY no comando de seleção:

```
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**.

```
SELECT placa, km  
FROM VEICULOS  
ORDER BY km;  
  
SELECT placa, km  
FROM VEICULOS  
WHERE ano > 2000  
ORDER BY km DESC;
```

É possível também ordenar por mais de uma coluna:

```
SELECT marca, modelo  
FROM VEICULOS  
ORDER BY marca DESC, modelo ASC;
```

Manipulação Básica de Dados – COUNT(*)

Embora seja uma palavra-chave que pertence a outra categoria na linguagem SQL, é bastante conveniente conhecer nesse momento a “função de agregação” denominada COUNT.


Em rápidas palavras, uma função de agregação é uma função que opera sobre um conjunto de linhas de uma tabela. A função COUNT retorna o número de linhas envolvidas em uma consulta.

Para saber quantas linhas no total existem na tabela VEICULOS, podemos usar:

```
SELECT COUNT (*)  
FROM VEICULOS ;
```

Para saber quantos veículos zero-KM temos registrados, podemos usar:

```
SELECT COUNT (*)  
FROM VEICULOS  
WHERE km = 0 ;
```

 Agora tente descobrir quantos veículos da marca Ford estão cadastrados no banco de dados.

Manipulação Básica de Dados – DISTINCT

Evitando duplicatas

Depois de inserir diversos registros na tabela de veículos, experimente executar o seguinte comando:

```
SELECT marca  
FROM VEICULOS;
```

Se existir mais de um veículo da mesma marca, a listagem desse comando exibe o nome daquela marca repetidamente – afinal de contas, estamos solicitando a coluna que contém a marca de cada um dos registros armazenados.

Se, no entanto, quisermos saber o nome de cada uma das marcas do nosso estoque de veículos, mas sem repeti-los (ou seja, eliminando as ocorrências duplicadas), devemos usar o seguinte comando:

```
SELECT DISTINCT marca  
FROM VEICULOS;
```



Para pensar: Faz sentido usar a cláusula DISTINCT para a coluna “placa” da tabela VEICULOS?

Manipulação Básica de Dados Parte II

Manipulação Básica de Dados – NULL

Valores Nulos

Quando criamos uma tabela, podemos especificar, para cada campo, se ele é obrigatório ou não. Para especificar que um campo é obrigatório indicamos **NOT NULL**, e para especificar que um campo é de conteúdo opcional indicamos **NULL**. O padrão da linguagem SQL é **NULL** (campo opcional).

Por exemplo:

```
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
);
```


Manipulação Básica de Dados – NULL

Depois, para inserir registros podemos especificar cada um dos campos da tabela (e inclusive a sua ordem)...

```
-- ordem normal das colunas:  
INSERT INTO PESSOAS (cpf, nome, idade, endereco)  
VALUES ('32809', 'Maria', 25, 'Rua A, 20');  
  
-- outra ordem qualquer das colunas:  
INSERT INTO PESSOAS (idade, endereco, cpf, nome)  
VALUES (25, 'Rua A, 20', '30599', 'Pedro');  
  
-- valores nulos:  
INSERT INTO PESSOAS (cpf, nome, idade, endereco)  
VALUES ('29385', 'Carlos', NULL, NULL);  
  
INSERT INTO PESSOAS (cpf, nome, idade, endereco)  
VALUES ('39582', 'Alice', 80, NULL);  
  
INSERT INTO PESSOAS (cpf, nome, idade, endereco)  
VALUES ('78838', 'Antonio', NULL, 'Rua B, 80');
```

Manipulação Básica de Dados – NULL

... ou podemos omitir alguns dos campos.


Neste caso, o valor **NULL** é implícito.

Mas, para isso, temos que omitir os nomes dos campos também no comando de inserção:

```
INSERT INTO PESSOAS (cpf, nome)
VALUES ('90038', 'Ana Paula');

INSERT INTO PESSOAS (cpf, nome, idade)
VALUES ('23487', 'Patricia', 18);

INSERT INTO PESSOAS (cpf, nome, endereco)
VALUES ('23363', 'Jose', 'Rua C, 50');
```

 Experimente alguns comandos SELECT para ver como os dados aparecem. Por exemplo, descubra quantas pessoas omitiram alguma informação para o cadastro.

Para fazer busca por valores nulos, não devemos usar o operador de igualdade convencional (=). É necessário usar os operadores especiais **IS NULL** e **IS NOT NULL**:

```
-- Pessoas sem especificação de idade:
SELECT *
FROM PESSOAS
WHERE idade IS NULL;
```

```
-- Pessoas que forneceram algum endereço:
SELECT *
FROM PESSOAS
WHERE endereco IS NOT NULL;
```

Manipulação Básica de Dados – LIKE e IN

Operadores LIKE e IN

O operador **LIKE** é usado para localizar textos. O símbolo '%' substitui zero ou mais caracteres.

```
-- 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%';  
  
-- Pessoas com nomes que terminam com 'Silva':  
SELECT *  
FROM PESSOAS  
WHERE nome LIKE '%Silva';  
  
-- Pessoas com nomes que contenham 'Carlos':  
SELECT *  
FROM PESSOAS  
WHERE nome LIKE '%Carlos%';
```

Manipulação Básica de Dados – LIKE e IN

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);
```

Manipulação Básica de Dados – ALTER TABLE

Removendo e adicionando novas colunas em tabelas

Remover e adicionar colunas em uma tabela significa alterar a sua estrutura. Para isso usamos o comando `ALTER TABLE`.

Para remover uma coluna de uma tabela usamos o comando `ALTER TABLE` com a cláusula `DROP COLUMN`:

```
ALTER TABLE PESSOAS  
DROP COLUMN idade;
```

Para adicionar uma nova coluna em uma tabela usamos o comando `ALTER TABLE` com a cláusula `ADD`:

```
ALTER TABLE PESSOAS  
ADD sexo CHAR(1);
```

Manipulação Básica de Dados – DATE

Manipulando Datas

Nossa tabela **PESSOAS** armazena a idade de cada pessoa, mas essa não foi uma boa escolha de projeto porque precisaríamos atualizar as idades ano após ano.

Uma escolha melhor teria sido:

```
CREATE TABLE PESSOAS
(
  cpf          VARCHAR(20)    NOT NULL,
  nome         VARCHAR(150)   NOT NULL,
  datanasc     DATE           NULL,  -- alterado!
  endereco     VARCHAR(150)   NULL
);
```

Observação: A manipulação de datas é muito dependente do produto de banco de dados que se está usando.

As informações que constam aqui se referem ao SGBD da Oracle.

O tipo DATE armazena o século, todos os quatro dígitos de um ano, o mês, o dia, a hora (formato 24h), os minutos e os segundos.

Usando os comandos para alteração da tabela, vamos modificar a sua estrutura para substituir idade por data de nascimento:

```
ALTER TABLE PESSOAS
DROP COLUMN idade;
```

```
ALTER TABLE PESSOAS
ADD datanasc DATE NULL;
```

Manipulação Básica de Dados – DATE

Inserindo datas em um campo DATE

O formato padrão varia conforme a instalação do SGBD. Supondo que a sua instalação apresenta o formato DD-MON-YYYY, temos um dia com 2 dígitos, as três primeiras letras do mês (inglês) e um ano com 4 dígitos. Exemplo:

```
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');
```

Manipulação Básica de Dados – DATE

Funções de conversão de datas

As funções `TO_CHAR()` e `TO_DATE()` convertem uma data/horário em uma string e vice-versa.

A sintaxe geral é `TO_CHAR(x [, formato])` e `TO_DATE(x [, formato])`.

O elemento `SYSDATE` captura data e horas atuais.



Experimente os seguintes comandos:

```
SELECT TO_CHAR(SYSDATE, 'MONTH, DD, YYYY  
HH24:MI:SS')  
FROM PESSOAS;
```

-- DUAL é uma tabela interna do Oracle usada com o comando SELECT quando não precisamos de uma tabela real do banco de dados.

```
SELECT nome, TO_CHAR(datanasc, 'MONTH, DD, YYYY')  
FROM PESSOAS;
```

```
INSERT INTO PESSOAS (cpf, nome,  
datanasc, endereco)  
VALUES (  
    '29920',  
    'Beto',  
    TO_DATE('25-FEB-1979 21:36:28',  
    'DD-MON-YYYY HH24:MI:SS'),  
    'Rua E, 80'  
);
```


Manipulação Básica de Dados – DATE

Aritmética de Datas

Em SQL é possível realizar as seguintes operações sobre datas:

- $\text{DATE} + \text{NUMBER} = \text{DATE}$
- $\text{DATE} - \text{NUMBER} = \text{DATE}$
- $\text{DATE} - \text{DATE} = \text{número de dias entre as datas}$

Por exemplo:

```
SELECT SYSDATE + 1  
FROM DUAL
```

Observação: A tabela **DUAL** é uma tabela “dummy” (no SGBD Oracle). Ela contém somente uma coluna chamada “dummy” e apenas uma linha que contém o valor 'X'.

Ela é utilizada sempre que se deseja retornar uma única linha em uma consulta e também porque todo comando **SELECT** deve possuir uma cláusula **FROM**.

Manipulação Básica de Dados Parte III

Manipulação Básica de Dados – INTEGRIDADE

Restrição de Integridade de Entidade

Para manter bancos de dados com qualidade, é necessário garantir a integridade dos dados armazenados. A **integridade de entidade** define uma linha como entidade exclusiva de uma determinada tabela.

Por exemplo, não podemos permitir que exista mais de uma pessoa com o mesmo CPF ou mais de um veículo com uma mesma placa. Para isso, devemos escolher, para cada tabela, o conjunto mínimo de colunas (atributos) que unicamente identifica cada registro. Na tabela **PESSOAS**, a coluna '**cpf**' deve ter valores únicos. Na tabela **VEICULOS**, a coluna '**placa**' deve ter valores únicos. Em alguns casos será necessário usar duas ou mais colunas para garantir unicidade (isso será visto mais adiante). Chamamos esses conjuntos de atributos de “chave”.

Às vezes uma tabela pode apresentar mais de uma alternativa para essa escolha. Considere, por exemplo, uma tabela de **ALUNOS** contendo os campos '**nroMatricula**' e '**cpf**'. Nesse caso os dois campos, individualmente, devem ser únicos. Nos bancos de dados, chamamos cada um desses casos de **chaves candidatas**.

Depois que definimos todas as chaves candidatas para uma determinada tabela, escolhemos apenas uma delas como sendo a principal. Por exemplo, se uma tabela apresenta três chaves candidatas, escolhemos uma delas como sendo a principal (arbitrariamente, já que qualquer uma delas serviria para esse propósito).

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**).

Manipulação Básica de Dados – INTEGRIDADE

Em SQL usamos as restrições **PRIMARY KEY** (para a principal) e **UNIQUE** (para as demais AKs). Naturalmente, toda PK é também NOT NULL (por quê?). Por exemplo:


```
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
);
```

Essas declarações de restrições **PRIMARY KEY** e **UNIQUE** estão na forma inline, ou seja, são especificadas junto com a criação de cada campo no comando CREATE TABLE.

```
CREATE TABLE ALUNOS
(
  nroMatricula  VARCHAR(10)  NOT NULL,
  cpf           VARCHAR(20)  NOT NULL,
  email         VARCHAR(100) NOT NULL,
  nome          VARCHAR(150) NOT NULL,
  anoIngresso   NUMBER(4)    NOT NULL,
  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)
);
```

A forma alternativa que usa a palavra **CONSTRAINT** (“restrição”) é preferível por ser mais flexível.

 Experimente agora tentar inserir registros diferentes, mas com o mesmo número de matrícula ou com um mesmo número de CPF para ver como o SGBD impede a operação e relata um erro de duplicidade de chave.

Manipulação Básica de Dados – INTEGRIDADE

Restrição de Integridade de Domínio

A integridade de domínio visa a 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 sexo somente aceite os valores 'M' ou 'F'.

Esses casos podem ser garantidos usando a restrição **CHECK**. Por exemplo, vamos garantir que o campo 'anoIngresso' possua sempre um valor superior a 2000 e que o campo sexo permita apenas os valores 'M' ou 'F' (em maiúsculas):

```
ALTER TABLE ALUNOS
ADD CONSTRAINT CK_AnoIngr CHECK (anoIngresso > 2000);

ALTER TABLE ALUNOS
ADD CONSTRAINT CK_sexo CHECK (sexo IN ('M', 'F'));
```

Manipulação Básica de Dados – INTEGRIDADE

Restrição de Integridade Referencial

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

Por exemplo, suponha uma tabela que registra os estados (unidades federativas) do país:

```
CREATE TABLE ESTADOS
(
  uf      CHAR(2)      NOT NULL,
  nome    VARCHAR2(40) NOT NULL,
  regioao CHAR(2)      NOT NULL,
  CONSTRAINT PK_ESTADOS PRIMARY KEY (uf)
);
```

Suponha agora outra tabela que registra as cidades, vinculando-as aos estados:

```
CREATE TABLE CIDADES
(
  cod_cidade NUMBER(4)      NOT NULL,
  nome       VARCHAR2(60) NOT NULL,
  uf         CHAR(2)       NOT NULL,
  CONSTRAINT PK_CIDADES PRIMARY KEY (cod_cidade)
);
```

Como não há verificação de consistência (integridade) entre os dados das duas tabelas, seria perfeitamente possível cadastrar uma cidade especificando sua 'uf' como 'XX', ou ainda especificando sua 'uf' com uma sigla de estado ainda não cadastrada na tabela de estados.

Manipulação Básica de Dados – INTEGRIDADE

Então, como podemos garantir que, ao incluir uma nova cidade, o seu campo 'uf' se refere a um estado que realmente existe (ou seja, já tenha sido anteriormente cadastrado)?

A resposta é: com uma chave estrangeira (em inglês, **FOREIGN KEY – FK**)

O comando a seguir cria um vínculo entre as duas tabelas, definindo uma regra de integridade relacional do campo 'uf' da tabela de cidades para o campo 'uf' da tabela de estados (ou seja, para a chave primária da tabela 'pai'):

```
ALTER TABLE CIDADES
ADD
(
    CONSTRAINT FK_EST_CID
        FOREIGN KEY (uf)
        REFERENCES ESTADOS (uf)
);
```

Observação: se houver algum registro que não atenda à restrição que estamos tentando adicionar, o SGBD retorna um erro (por quê?).

 Experimente agora inserir dados nas duas tabelas depois de enviar o comando ALTER TABLE para o banco de dados.

 Tente também forçar uma inconsistência de dados, como por exemplo, inserir uma cidade para um estado que ainda não esteja cadastrado.

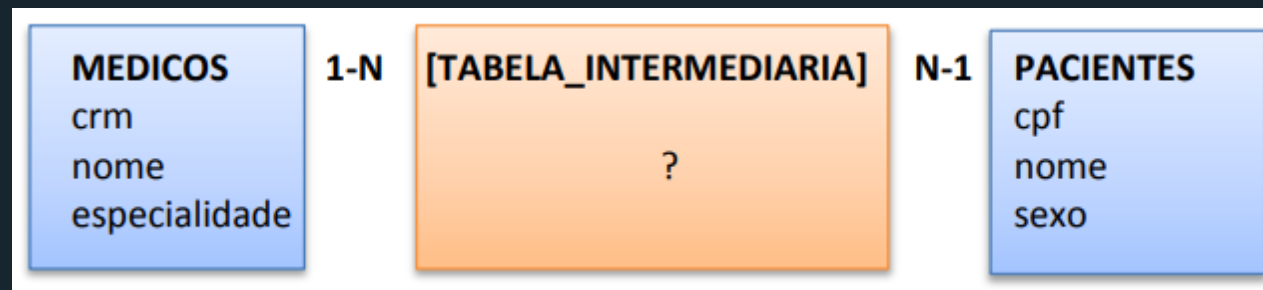
Manipulação Básica de Dados – INTEGRIDADE

Relacionamentos do tipo muitos-para-muitos

Dizemos que o relacionamento existente entre as tabelas ESTADOS e CIDADES é do tipo um-para-muitos (ou 1-N). Em outras palavras, estamos dizendo que cada estado pode ter várias cidades, mas que cada cidade pertence a apenas um estado.

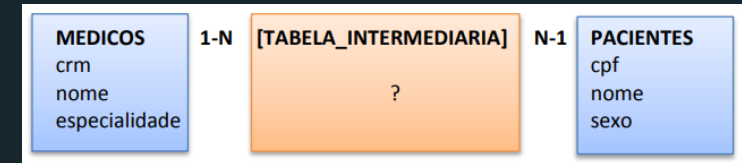
No entanto, em alguns casos precisamos representar relacionamentos do tipo muitospara-muitos (ou N-N). Esse é o caso dos relacionamentos entre MEDICOS e PACIENTES ou então entre PESSOAS e PROJETOS, por exemplo.

Infelizmente, não há como representar relacionamentos N-N entre (usando apenas) duas tabelas em um banco de dados relacional. No entanto, podemos resolver esse problema criando uma terceira tabela (ou seja, uma tabela intermediária entre as duas tabelas originais) e criando dois relacionamentos 1-N entre as tabelas:



Manipulação Básica de Dados – INTEGRIDADE

Relacionamentos do tipo muitos-para-muitos



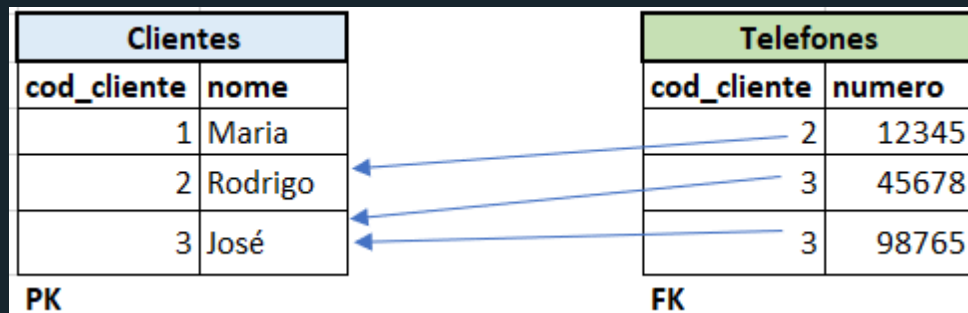
Prática:

- ✎ Crie uma tabela MEDICOS com os campos 'crm', 'nome' e 'especialidade'.
- ✎ Crie uma tabela PACIENTES com os campos 'cpf', 'nome' e 'sexo'.
- ✎ Crie uma tabela chamada MEDICOSPACIENTES com os campos 'crm' e 'cpf'. Esta tabela servirá para saber quais médicos tratam de quais pacientes (e, por consequência, quais pacientes são atendidos por quais médicos).
- ✎ Defina uma FK entre MEDICOSPACIENTES.crm e MEDICOS.crm.
- ✎ Defina outra FK entre MEDICOSPACIENTES.cpf e PACIENTES.cpf.
- ✎ Qual será a chave primária da tabela MEDICOSPACIENTES?
- ✎ E se quiséssemos armazenar o histórico de consultas entre médicos e pacientes? Crie uma tabela de consultas, que contenha a data de cada consulta.

Manipulação Básica de Dados Parte IV

Manipulação Básica de Dados – JOIN

Até o momento, consultamos dados a partir de uma única tabela. Esta parte do material introduz o conceito de **JUNÇÕES**, que permitem a consulta de dados de mais de uma tabela, pela relação entre **chaves estrangeiras** e **chaves primárias**.



1) Clientes que têm telefone

inner join

2	Rodrigo	2	12345
3	José	3	45678
3	José	3	98765

```
SELECT EST.uf, EST.nome, CID.uf, CID.nome
FROM ESTADOS EST INNER JOIN CIDADES CID
ON EST.uf = CID.uf;
```

2) Todos os clientes, mesmo os que não têm telefone

outer join

1	Maria	null	null
2	Rodrigo	2	12345
3	José	3	45678
3	José	3	98765

```
SELECT EST.uf, EST.nome, CID.uf, CID.nome
FROM ESTADOS EST LEFT OUTER JOIN CIDADES CID
ON EST.uf = CID.uf;
```

Manipulação Básica de Dados – CROSS JOIN

O produto cartesiano é uma operação da teoria de conjuntos. Executar um produto cartesiano entre duas tabelas resulta na combinação de todas as linhas (registros) da primeira tabela com todas as linhas da segunda tabela.

Clientes		Telefones	
cod_cliente	nome	cod_cliente	numero
1	Maria	2	12345
2	Rodrigo	3	45678
3	José	3	98765

PK FK

Produto Cartesiano = cross join				junção = inner join	
cod_cliente	nome	cod_cliente	numero	clientes.cod_cliente = telefones.cod_cliente	
1	Maria	2	12345	FALSO	
1	Maria	3	45678	FALSO	
1	Maria	3	98765	FALSO	
2	Rodrigo	2	12345	VERDADEIRO	
2	Rodrigo	3	45678	FALSO	
2	Rodrigo	3	98765	FALSO	
3	José	2	12345	FALSO	
3	José	3	45678	VERDADEIRO	
3	José	3	98765	VERDADEIRO	

Observação:

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.

Manipulação Básica de Dados – OUTER JOIN

O produto cartesiano é uma operação da teoria de conjuntos. Executar um produto cartesiano entre duas tabelas resulta na combinação de todas as linhas (registros) da primeira tabela com todas as linhas da segunda tabela.

Clientes			Telefones	
cod_cliente	nome		cod_cliente	numero
1	Maria		2	12345
2	Rodrigo	←	3	45678
3	José	←	3	98765
PK			FK	

junção = inner join

clientes.cod_cliente = telefones.cod_cliente

cod_cliente	nome	cod_cliente	numero
2	Rodrigo	2	12345
3	José	3	45678
3	José	3	98765

left outer join

clientes.cod_cliente = telefones.cod_cliente

cod_cliente	nome	cod_cliente	numero
1	Maria	null	null
2	Rodrigo	2	12345
3	José	3	45678
3	José	3	98765

```
SELECT EST.uf, EST.nome, CID.uf, CID.nome
FROM ESTADOS EST LEFT OUTER JOIN CIDADES CID
ON EST.uf = CID.uf;
```

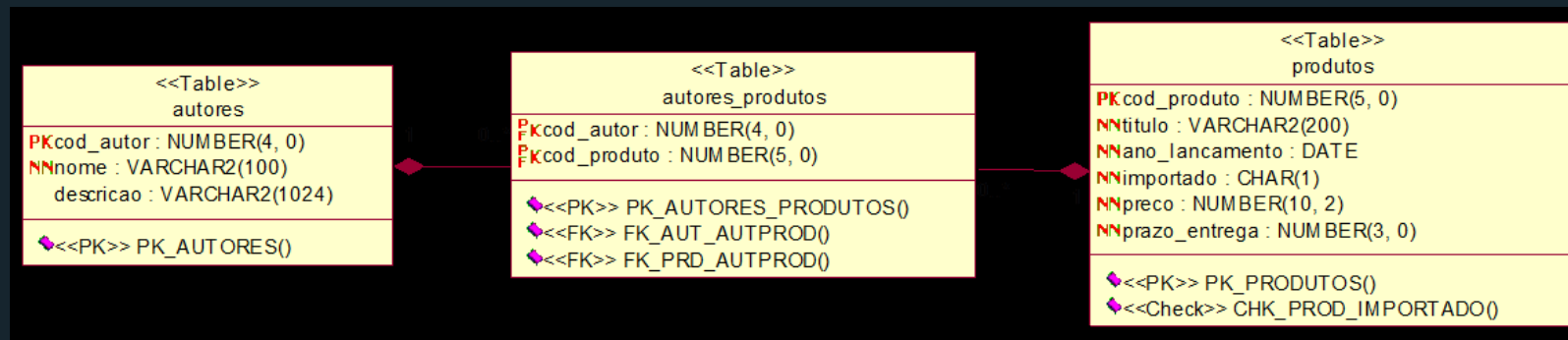
```
SELECT EST.uf, EST.nome, CID.uf, CID.nome
FROM ESTADOS EST RIGHT OUTER JOIN CIDADES CID
ON EST.uf = CID.uf;
```

```
SELECT EST.uf, EST.nome, CID.uf, CID.nome
FROM ESTADOS EST FULL OUTER JOIN CIDADES CID
ON EST.uf = CID.uf;
```

Manipulação Básica de Dados – JOIN

JOINS encadeados

Há ocasiões em que precisamos buscar dados de mais de duas tabelas. Em outros casos, os dados de que precisamos encontram-se em tabelas mais “distantes” no esquema do banco de dados. Para poder obter esses dados, precisamos usar **JOINS encadeados**.



```
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);
```

Observações:

- Lembre-se de usar apelidos (*alias*) para tabelas ao usar junções para facilitar a redação do comando.
- O tipo de JOIN mais usado é o EQUI-JOIN.
- Para buscar dados de N tabelas relacionadas, usamos N-1 JOINS.
- As junções externas também incluem resultados quando não há correspondência entre os dados das tabelas.

Manipulação Avançada de Dados

Manipulação Avançada de Dados – Funções

Funções

Em SQL podemos aplicar dois tipos de funções sobre linhas de uma tabela:

1. Funções sobre linhas, as quais operam sobre cada linha do resultado individualmente; e
2. 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).

As funções numéricas mais comuns são:

ABS(n), ACOS(n), ASIN(n), ATAN(n), ATAN2(n), CEIL(n), COS(n), COSH(n), EXP(n), FLOOR(n), LN(n), LOG(n), MOD(n,m), POWER(n,m), ROUND(n,m), SIGN(n), SIN(n), SINH(n), SQRT(n), TAN(n), TANH(n), TRUNC(n,m)

As funções sobre caracteres mais comuns são:

LOWER(s), UPPER(s), INITCAP(s), LTRIM(s1,s2), RTRIM(s1,s2), CONCAT(s1,s2), LPAD(s1,n,s2), RPAD(s1,n,s2), LENGTH(s), SUBSTR(s,n,m), REPLACE(s1,s2,s3), CHR(n), SOUNDEX(s), TRANSLATE(s1,s2,s3).

Nota: Não veremos cada uma das funções, mas você pode pesquisar a documentação da linguagem SQL para obter mais informações.

Manipulação Avançada de Dados – Funções

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

A forma geral é:

```
SELECT função_agregada  
FROM nome_da_tabela  
[...]
```

Funções agregadas mais comuns:

COUNT	(*)	
COUNT	([ALL DISTINCT]	nome_da_coluna)
SUM	([ALL DISTINCT]	nome_da_coluna)
AVG	([ALL DISTINCT]	nome_da_coluna)
MAX	([ALL DISTINCT]	nome_da_coluna)
MIN	([ALL DISTINCT]	nome_da_coluna)
STDDEV	([ALL DISTINCT]	nome_da_coluna)
VARIANCE	([ALL DISTINCT]	nome_da_coluna)

Alguns exemplos:

```
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;
```

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

Manipulação Avançada de Dados – GROUP BY

Agrupamento via cláusula GROUP BY

A cláusula **GROUP BY** pode ser utilizada em um comando **SELECT** para agrupar os resultados de uma consulta, gerando subtotais por grupos em novas colunas.

A forma geral é:

```
SELECT nome_da_coluna [, ...], função_agregada [, ...]  
FROM nome_da_tabela [, ...]  
GROUP BY [ALL] nome_da_coluna [, ...]  
ORDER BY colunas
```

Observações sobre a cláusula **GROUP BY**:

- 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.

Manipulação Avançada de Dados – GROUP BY

Exemplos:

```
CREATE TABLE PRODS
(
  codigo NUMERIC(3) NOT NULL,
  nome VARCHAR(50) NOT NULL,
  preco NUMERIC (5,2) NOT NULL,
  tipo CHAR(1) NULL, -- [S]uprimento, [C]omponente, [P]eriférico
  CONSTRAINT PK1 PRIMARY KEY (codigo)
);
```

```
INSERT INTO PRODS VALUES( 10, 'HD' ,200 , 'C');
INSERT INTO PRODS VALUES( 11, 'Memoria' ,250 , 'C');
INSERT INTO PRODS VALUES( 12, 'Impressora' ,680 , 'P');
INSERT INTO PRODS VALUES( 13, 'Processador' ,600 , 'C');
INSERT INTO PRODS VALUES( 14, 'DVD-RW' ,2 , 'S');
INSERT INTO PRODS VALUES( 15, 'Papel A4' ,19 , 'S');
INSERT INTO PRODS VALUES( 16, 'Scanner' ,199 , 'P');
```

 Escreva comandos SELECT para as seguintes consultas:

- a) Quantos produtos existem na tabela PRODS?
- b) Quantos tipos de produtos existem na tabela PRODS?
- c) Quantos produtos existem de cada tipo?
- d) Qual a média de preço de todos os produtos?
- e) Qual a média de preço dos suprimentos (tipo 'S')?
- f) Qual a média de preço dos produtos de cada tipo?

Manipulação Avançada de Dados – GROUP BY

Exemplos:

```
ALTER TABLE PRODS ADD (usuario NUMBER(1) NULL);

UPDATE PRODS
SET usuario = 1
WHERE codigo IN (10,12,13,14);

UPDATE PRODS
SET usuario = 2
WHERE usuario IS NULL;

SELECT tipo, usuario, AVG(preco)
FROM PRODS
GROUP BY tipo, usuario
ORDER BY tipo, usuario;

UPDATE PRODS
SET usuario = 2
WHERE codigo = 14;
```

```
-- Deixando um produto sem usuario associado

UPDATE PRODS
SET usuario = NULL
WHERE codigo = 13;

-- Executando novamente

SELECT tipo, usuario, AVG(preco)
FROM PRODS
GROUP BY tipo, usuario
ORDER BY tipo, usuario;
```

Manipulação Avançada de Dados – HAVING

GROUP BY com 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.

A forma geral é:

```
SELECT nome_da_coluna [, ...], função_agregada [, ...]
FROM nome_da_tabela [, ...]
GROUP BY [ALL] nome_da_coluna [,...]
HAVING condições
ORDER BY colunas
```

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

Exemplo:

```
SELECT CID.nome, COUNT(*) QTD
FROM CIDADES CID JOIN ENDERECOS END
      ON CID.cod_cidade = END.cod_cidade
GROUP BY CID.nome
HAVING COUNT(*) > 10;
```

Manipulação Avançada de Dados – Subconsultas

Na linguagem SQL, subconsultas são comandos `SELECT` aninhados dentro de outros comandos `SELECT`, `INSERT`, `UPDATE` ou `DELETE`. Podemos ter, inclusive, subconsultas dentro de outras subconsultas. O número de níveis permitido depende do SGBD.

Existem basicamente dois tipos de subconsultas:

1. Subconsultas que retornam um único valor; e
2. Subconsultas que retornam um conjunto de valores (ou registros).

Manipulação Avançada de Dados – Subconsultas

Subconsultas que retornam um único valor:

Quando uma subconsulta retorna um único valor, usamos os operadores básicos de comparação.

Por exemplo, como podemos obter os títulos dos produtos que são mais caros do que o produto cujo código é 9?

Primeiro seria necessário descobrir o preço de tal produto:

```
SELECT preco  
FROM produtos  
WHERE cod_produto = 9
```

Esse comando retorna um único valor (179,00). A partir desse resultado, podemos montar o seguinte comando:

```
SELECT titulo  
FROM produtos  
WHERE preco > 179
```

Manipulação Avançada de Dados – Subconsultas

Subconsultas que retornam um único valor:

Mas isso exige duas consultas independentes (e duas trocas de dados entre o aplicativo cliente e o SGBD). Podemos unir os dois comandos SELECT usando o conceito de subconsultas.

Usando uma subconsulta ficaria assim:

```
SELECT titulo
FROM PRODUTOS
WHERE preco >
      (SELECT preco
       FROM PRODUTOS
       WHERE cod_produto = 9) ;
```

Podemos usar todos os operadores relacionais típicos entre as duas consultas:

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

Observação: Repare que a subconsulta é colocada do lado direito do operador relacional. Coloque a subconsulta entre parênteses.

Manipulação Avançada de Dados – Subconsultas

Subconsultas que retornam um único valor:

✎ Execute os seguintes comandos e identifique que dados eles obtêm.

```
SELECT titulo
FROM PRODUTOS
WHERE importado = 'N' AND preco >
      (SELECT MAX(preco)
       FROM PRODUTOS
       WHERE importado = 'S');
```

```
SELECT ano_lancamento, AVG(preco)
FROM PRODUTOS
GROUP BY ano_lancamento
HAVING AVG(preco) >
      (SELECT AVG(preco)
       FROM PRODUTOS
       WHERE ano_lancamento = trunc(sysdate, 'YYYY');
```

Manipulação Avançada de Dados – Subconsultas

Atualizações com subconsultas:

É possível utilizar uma subconsulta dentro dos comandos INSERT, UPDATE e DELETE.

Exemplo com INSERT

```
INSERT INTO PRODS (codigo, nome, preco, tipo)
```

```
SELECT
```

```
    cod_produto
```

```
    SUBSTR(titulo, 1, 15),
```

```
    preco,
```

```
    'L' -- coluna constante para todos os registros
```

```
FROM produtos
```

```
WHERE
```

```
    importado = 'N'
```

```
    AND titulo LIKE 'A%'
```

```
    AND cod_produto > 2;
```

Manipulação Avançada de Dados – Subconsultas

Atualizações com subconsultas:

Exemplo com UPDATE

```
UPDATE PRODUTOS
SET preco = preco - (10/100 * preco)
WHERE cod_produto IN
    ( SELECT cod_produto
      FROM PRODUTOS
      WHERE prazo_entrega > 30 ) ;
```

Exemplo com DELETE

```
DELETE FROM PRODS
WHERE codigo IN
    ( SELECT cod_produto
      FROM PRODUTOS
      WHERE
        importado = 'N'
        AND titulo LIKE 'A%'
        AND cod_produto > 100 ) ;
```

Manipulação Avançada de Dados – Indexação

Exemplo com a tabela Cidades:

```
SELECT nome, uf FROM CIDADES;
```

Por que o Oracle está me mostrando os nomes das cidades ordenados?

- Porque casualmente ao rodar o script de INSERÇÃO estava já em ordem.
- Então o SGBD foi preenchendo cada DATABLOCK na ordem que o script foi sendo executado.
- Então quando consulto ele lista nessa ordem.
- E os dados estão ordenados porque casualmente os dados foram inseridos de forma ordenada.
- É apenas uma coincidência sobre a forma como eu inseri os dados aqui.

Manipulação Avançada de Dados – Indexação

Mas o fato é que **os SGBD não garantem ordenação nas consultas!!!**

Por exemplo, Autores não estão ordenados:

```
SELECT nome FROM AUTORES;
```

Como posso procurar um autor específico?

Que tipo de algoritmo o SGBD poderia usar?

- Uma busca sequencial? Tem um custo muito alto. Mas pode dar sorte, se estiver no início.
- No pior caso será o último da lista ou nem estar no Banco de dados.

Manipulação Avançada de Dados – Indexação

O SGBD precisa usar um algoritmo otimizado para agilizar as pesquisas!!!

Por exemplo, **Pesquisa Binária!!!**

Mas para isso **os dados precisam estar ordenados!!!**

Quantos registros tem a tabela Autores?

```
SELECT Count(*) FROM AUTORES;
```

- Como acharíamos o autor 'Emílio F. Moran' usando pesquisa binária?

```
SELECT nome FROM AUTORES ORDER BY nome;
```

Manipulação Avançada de Dados – Indexação

Então, não seria mais fácil guardar o arquivo ordenado?

Não, porque para pesquisar pela coluna ordenada seria fácil, mas para pesquisar pelas outras colunas não!!!

E o SGBD foi feito para que façamos as consultas que quisermos !!!

Então, como o SGBD não pode guardar de forma ordenada por uma coluna porque restringiria a capacidade dele de fazer consultas por outras colunas, QUAL FOI A ESTRATÉGIA ADOTADA PELOS SGBDS?

Criar um outro arquivo chamado de arquivo de índice que vai conter a **Chave de Busca** e um ponteiro físico para aquela linha.

Manipulação Avançada de Dados – Indexação

O rowid, é uma pseudo coluna que me dá o endereço físico de uma coluna. Dentro dessa estrutura está o endereço completo para o SGBD encontrar o registro lá no disco. Ele faz um acesso direto ao DATABLOCK no disco. É a forma mais eficiente de trazer do disco a informação.

```
SELECT cod_autor, nome, rowid FROM Autores;
```

Então o que a gente precisaria ter para indexar uma tabela?

Um outro arquivo com a **Chave de Busca ordenada**, mais o **rowid**.

Manipulação Avançada de Dados – Indexação

Então, se eu quiser um índice por NOME. Vai ser um arquivo separado com todos os NOMES ordenados e mais o **rowid** para poder localizar a linha nessa tabela.

Se quiser pesquisar por outra coluna como o COD_AUTOR então cria outro índice.

1. O SGBD vai lá no índice de código de autor, porque está ordenado, faz a busca eficiente nesse índice, vai encontrar o **rowid**.
2. Se não encontrar nada nem vai na tabela porque ele sabe que não existe aquele dado na tabela.
3. Se encontrar, ele tem o **rowid**, então ele consegue recuperar aquela informação com um acesso direto ao DATABLOCK.

Resumindo:

- O índice é um arquivo separado.
- Tem duas informações, a Chave de Busca ordenada e o **rowid**.

Manipulação Avançada de Dados – Indexação

O que precisamos observar no nosso modelo físico:

- O Oracle cria automaticamente o índice da chave primaria (PK).
 - ❖ Por que ele faz isso?
 - Para garantir que não haja repetição de dados. Quando vou inserir um novo registro ele tem que procurar se já não existe e a busca precisa ser eficiente.
 - Traz vantagem também nas nossas consultas.
- **Desvantagem:**
 - ❖ Cada vez que insiro um novo registro ele precisa atualizar os índices.
 - Então equilibrar a quantidade de índices X inserções.
 - Então vamos criar só os índices extremamente necessários.
 - O Oracle não cria índice para Chave Estrangeira (Foreign Key).

Manipulação Avançada de Dados – Indexação

Sintaxe de criação de índices:

```
CREATE INDEX <nome do índice> ON <nome da tabela> (<nome da coluna>);
```

Tabela de AUTORES

```
CREATE INDEX idx_autores_nome ON AUTORES (nome);
```

Manipulação Avançada de Dados – Sequence

Auto incremento:

```
CREATE SEQUENCE seq_titulacoes START WITH 6;
```

Existe uma propriedade chamada **Nextval**.

```
SELECT seq_titulações.nextval FROM DUAL; -- rodar várias vezes para observar a variação
```

Para inserir um novo registro com **Nextval**:

```
INSERT INTO TITULACOES ( cod_titulação, titulo) VALUES (seq_titulações.nextval, 'Tecnnologo');
```

Para saber o número atual, ou corrente:

```
SELECT seq_titulações.currval FROM DUAL;
```

Manipulação Avançada de Dados – Sequence

Colocando a Sequence na criação da tabela:

```
CREATE TABLE TITULACOES  
(  
  cod_titulacao  NUMBER(4)          DEFAULT seq_titulacoes.nextval NOT NULL,  
  titulo         VARCHAR (20)      NOT NULL,  
  
  CONSTRAINT pk_titulacoes  PRIMARY KEY (cod_titulacao)  
);
```

Agora, como tenho o **DEFAULT** com a **Sequence**, para fazer **INSERT** posso fazer apenas assim:

```
INSERT INTO TITULACOES ( titulo)  VALUES ('Bacharel' );
```

```
INSERT INTO TITULACOES ( titulo)  VALUES ('Especialista' );
```

```
INSERT INTO TITULACOES ( titulo)  VALUES ('Mestre' );
```

- Introdução
- Funções
- Trabalhando com classes
- Exceções
- Funções assíncronas

Considerações finais

Considerações finais

- Explore as referências da disciplina

- VICCI, Claudia. Banco de Dados. São Paulo: Pearson, 2016.
- ELMASRI, R.; NAVATHE, S. B. Sistemas de banco de dados. 7a. Ed. São Paulo: Pearson Education do Brasil, 2018.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Sistema de bancos de dados. 7a. Ed. Rio de Janeiro: LTC, 2020.
- MACHADO, Felipe Nery Rodrigues. Banco de Dados – Projeto e Implementação. 4a. Ed. Porto Alegre: Saraiva, 2020.
- PICHETTI, Roni; et al. Banco de Dados. Porto Alegre: Sagah, 2020.
- PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. Banco de dados implementação em SQL PLSQL e Oracle 11g. São Paulo: Pearson, 2013.
- RAMAKRISHNAN, R.; GEHRKE, J. Sistemas de gerenciamento de banco de dados. 3a. Ed. Porto Alegre: AMGH, 2011.
- TEOREY, T. J.; et al. Projeto e modelagem de banco de dados. 2a Ed. Rio de Janeiro: Elsevier, 2014.

PUCRS online  **UOL** edtech.

PUCRS online  **uol**edtech.