

# Fundamentos de Banco de Dados

## SQL

Prof. MSc. Adriano Gomes

# SQL-Structured Query Language

- Originalmente desenvolvida nos laboratórios da IBM na década de 70;
- Primeira versão: **SEQUEL**-Structured English Query Language;
- Esforço para a padronização:
  - SQL1 (SQL-86)
  - SQL2 (SQL-92)
  - SQL3 (SQL-99)

# SQL

- Linguagem de Consulta Estruturada. Porém não abrange apenas consultas, mas **definição** (**DDL**) e **manipulação** (**DML**) dos dados;
- Fundamentada no modelo relacional (álgebra relacional) e padrão das bases relacionais;
- Utilizada tanto de forma interativa como incluída em linguagens hospedeiras
  - Java, C/C++, Cobol...

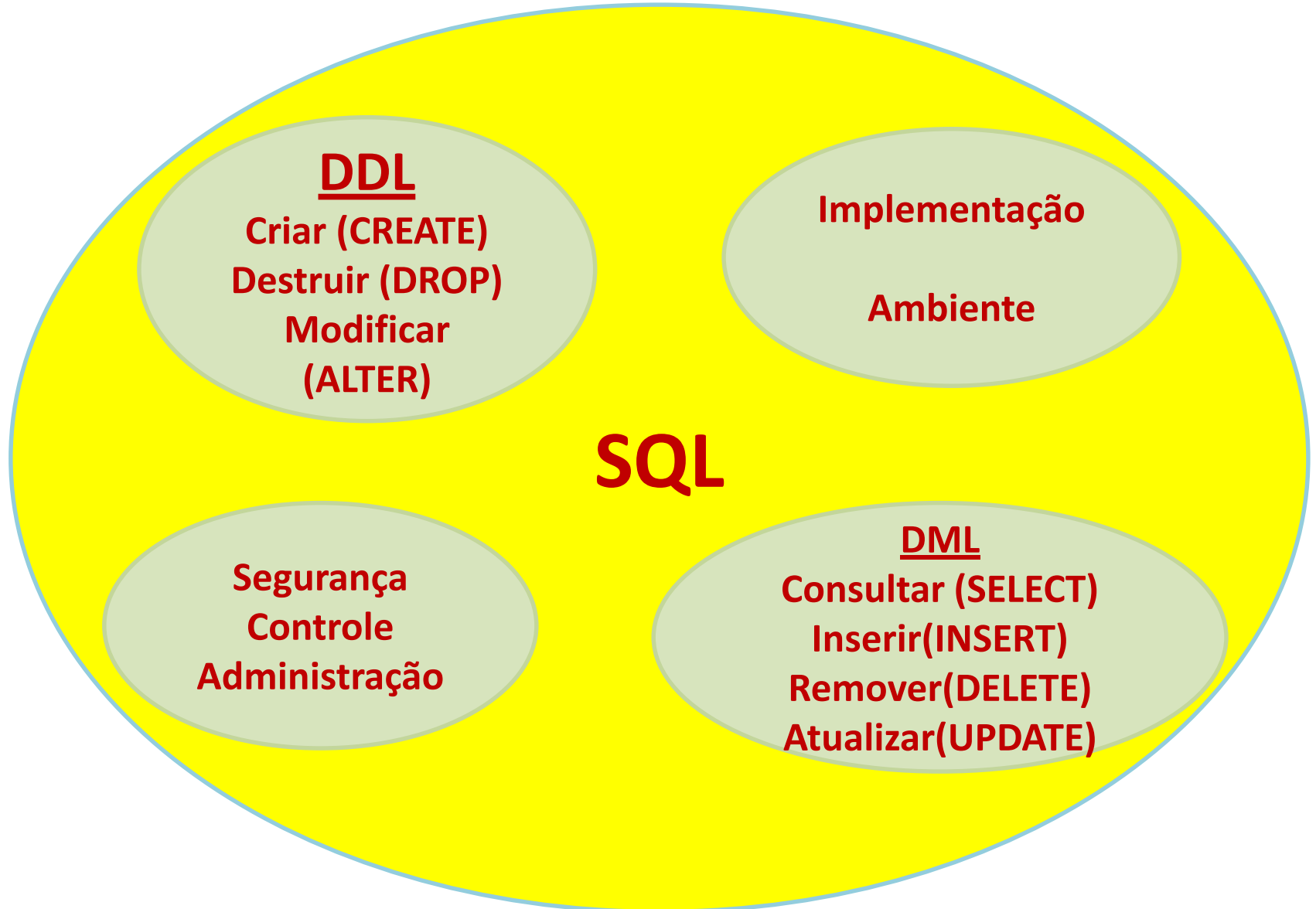
# Enfoques do SQL

- Linguagem interativa de consulta (ad-hoc): usuários podem definir consultas independente de programas;
- Linguagem de programação para acesso a banco de dados: comandos SQL embutidos em programas de aplicação;
- Linguagem de administração de dados: o DBA pode utilizar SQL para realizar suas tarefas;

# Enfoques do SQL

- **Linguagem cliente/servidor:** os programas clientes usam comandos SQL para se comunicarem e compartilharem dados com o servidor ;
- **Linguagem para banco de dados distribuídos:** auxilia na distribuição de dados por vários nós e na comunicação com outros sistemas;
- **Caminho de acesso a outros bancos de dados em diferentes máquinas:** auxilia na conversão entre diferentes produtos em diferentes máquinas;

# Usos de SQL



# SQL - Vantagens

- Independência de fabricante;
- Portabilidade entre sistemas;
- Redução de custos com treinamento;
- Comandos em inglês;
- Consulta interativa;
- Múltiplas visões de dados;
- Manipulação dinâmica dos dados;

# SQL - Desvantagens

- A padronização inibe a criatividade;
- Está longe de ser uma linguagem relacional ideal (C.J. Date);
- Algumas críticas:
  - Falta de ortogonalidade nas expressões, funções embutidas, variáveis indicadoras, referência a dados correntes, constantes NULL, conjuntos vazios...;
  - Discordância com as linguagens hospedeiras;
  - Não dá suporte a alguns aspectos do modelo relacional (atribuição de relação, join explícito, domínios, ...);



# Conteúdo

- Criação, alteração e destruição de tabelas;
- Extração de dados de uma tabela (Consultas);
- Definição de visões;
- Inserção, modificação e remoção de dados;

# Esquema Relacional Exemplo

Empregado(Matricula, Nome, Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

Projeto(Num\_Proj, Nome, lugar)

Trabalha(Mat Empr, Num\_Proj, Horas)

Dependente(Matricula, Nome, Grau\_Parentesco)

# Criação de Tabelas

- Comando CREATE TABLE

```
CREATE TABLE <nome tabela> (  
    <descrição dos atributos>  
    <descrição das chaves>  
    <descrição das restrições>)
```

- Descrição dos atributos -> <nome> <tipo>
- Os tipos de dados aceitos pelo **PostgreSQL**:  
varchar(n), character(n), char(n), text, smallint,  
integer, bigint, decimal, numeric, real, serial,  
bigserial, timestamp, interval, date, dentre  
outros.

# Criação de Tabelas

- Descrição das chaves

```
CONSTRAINT <nometabela_pkey>  
PRIMARY KEY(<nome dos atributos>)
```

- Só admite valor único

```
CONSTRAINT <nometabela_const>  
UNIQUE(<nome_atributo>)
```

# Criação de Tabelas

**Empregado(Matricula, Nome,  
Salario, Num\_Dept)**

```
CONSTRAINT Empregado_pkey  
PRIMARY KEY(matricula)
```

```
CONSTRAINT Empregado_const  
UNIQUE(nome)
```

# Criação de Tabelas

- Chave primária pode ser definida por auto-numeração

– Chave inteira cujo valor é atribuído pelo sistema, sendo incrementado de 1 a cada nova inserção;

```
CREATE TABLE <nome tabela> (<atributo> SERIAL ...)
```

– Chave inteira cujo valor é atribuído pelo sistema, sendo incrementado de 1 a cada nova inserção

```
CREATE SEQUENCE <nome sequencia>;
```

```
CREATE TABLE <nome tabela> (<atributo> tipo DEFAULT nextval('<nome sequencia>'), <lista de atributos>...);
```

# Criação de Tabelas

- Lista das chaves estrangeiras da forma

```
CONSTRAINT <nometabela_fkey>  
FOREIGN KEY (<atributo>  
REFERENCES <outra_tabela> (<chave primaria>)
```

**Empregado(Matricula, Nome,  
Salario, Num\_Dept)**

**Departamento(Num\_Dept,  
Nome )**

```
CONSTRAINT Empregado_fkey  
FOREIGN KEY(Num_Dept)  
REFERENCES Departamento (Num_Dept)
```

# Criação de Tabelas

- Descrição das restrições
- Salário não pode ser menor que o mínimo

```
CONSTRAINT <nometabela_check>  
CHECK (<restrição>)
```

**Empregado(Matricula, Nome,  
Salario, Num\_Dept)**

```
CONSTRAINT Empregado_check  
CHECK (salario > 678)
```



# Criação de Tabelas

```
CREATE TABLE Empregado
(Matricula integer,
 Nome varchar (20),
 Salario real,
 Num_Dept integer,
 CONSTRAINT empregado_pkey PRIMARY KEY (Matricula),
 CONSTRAINT empregado_fkey FOREIGN KEY (Num_Dept)
 REFERENCES Departamento (Num_Dept),
 CONSTRAINT Empregado_check CHECK (salario > 678) );
```

# Criação de Tabelas

- Criação de índices em uma tabela existente:

CREATE INDEX

- São estruturas que permitem agilizar a busca e ordenação de dados em tabelas

```
CREATE [UNIQUE] INDEX <nome>  
ON <tabela> (<atributo1> [, <atributo2>...]);
```

# Alteração das Tabelas

- Alterar definições de uma tabela existente ->

ALTER TABLE

```
ALTER TABLE <ação>;
```

- Ação pode ser: adicionar/remover uma coluna na tabela, adicionar/remover uma restrição de integridade.

# Alteração das Tabelas

- Acrescentar a coluna sexo na tabela Empregado;

```
ALTER TABLE EMPREGADO ADD Sexo char;
```

- Remover coluna da tabela Empregado

```
ALTER TABLE EMPREGADO DROP Sexo;
```

# Remoção das Tabelas

- Eliminar uma tabela já criada -> DROP TABLE

```
DROP TABLE <tabela>;
```

- Remover a tabela Empregado

```
DROP TABLE EMPREGADO;
```

- Os dados são também excluídos;

# Inserção de Dados em Tabelas

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Adicionar uma ou mais tuplas em uma tabela existente -> INSERT

```
INSERT INTO nome_da_tabela <lista de  
atributos> VALUES(<lista de valores>)
```

- Inserir uma tupla em Empregado

```
INSERT INTO Empregado (Matricula, Nome,  
Salario, Num_Dept) VALUES (015, 'José da  
Silva', 3000.00, 020)
```

# Inserção de Dados em Tabelas

- Inserir dados recuperados de uma tabela em outra tabela -> Uso do SELECT

```
INSERT INTO Empregado <lista de atributos>  
SELECT(<lista de valores>) FROM <tabela>  
WHERE <condição>;
```

- Armazenar em uma tabela para cada departamento com mais de 50 empregados, o número de empregado e a soma dos salários pagos

# Inserção de Dados em Tabelas

```
INSERT INTO Dept-Inf(Nome,Num_Emp,  
SomaSalario)  
SELECT D.Nome, COUNT(*),SUM(Salario)  
FROM Empregado E, Departamento D  
WHERE E.Num_Dept=D.Num_Dept  
GROUP BY D.nome  
HAVING COUNT(*) > 50
```



# Atualização de Dados em Tabelas

- Alterar os valores em uma tabela com base nas condições especificadas -> UPDATE

```
UPDATE <nome tabela>  
SET <nome atributo> = valor  
WHERE <condição>;
```

- Atualizar o salário do empregado de matrícula 015 para R\$1500,00

```
UPDATE Empregado SET Salario=1500.00 WHERE Matricula=015;
```

# Remoção de Dados em Tabelas

- Exclusão de dados de uma tabela já existente -> DELETE

```
DELETE FROM <nome tabela>  
WHERE <condição>;
```

- Remover empregados com salário superior a R\$2000,00

```
DELETE FROM Empregado  
WHERE Salario > 2000.00;
```

# Exercício – SQL para fazer em Casa

- Equipamento(cod\_equipamento, nm\_equipamento, valor, quantidade, tipo)
- Professor(cod\_professor, nm\_professor, telefone, cod\_curso, salario)
- Reserva(cod\_equipamento, cod\_professor, dt\_reserva, horário, cod\_sala)
- Sala (cod\_sala, nome\_sala)
- Curso(cod\_curso, nome\_curso)

# Resposta Exercício - SQL

- Equipamento(cod\_equipamento,  
nm\_equipamento, valor, quantidade, tipo)

```
CREATE TABLE Equipamento  
(cod_equipamento varchar(5),  
  nm_equipamento varchar (20),  
  valor real,  
  quantidade integer,  
  tipo varchar (10),  
  CONSTRAINT equipamento_pkey PRIMARY KEY (cod_equipamento));
```

# Resposta Exercício - SQL

- Sala (cod\_sala, nome\_sala)

```
CREATE TABLE Sala  
(cod_sala varchar(5),  
  nome_sala varchar (20),  
CONSTRAINT sala_pkey PRIMARY KEY (cod_sala));
```

- Curso(cod\_curso, nome\_curso)

```
CREATE TABLE Curso  
(cod_curso varchar(5),  
  nome_curso varchar (20),  
CONSTRAINT curso_pkey PRIMARY KEY (cod_curso));
```

# Resposta Exercício - SQL

- Professor(cod\_professor, nm\_professor, telefone, cod\_curso, salario)

```
CREATE TABLE Professor
(cod_professor varchar(5),
 nm_professor varchar (20),
 telefone varchar(8),
 cod_curso varchar(5),
 salario real,
 CONSTRAINT professor_pkey PRIMARY KEY (cod_professor),
 CONSTRAINT professor_fkey FOREIGN KEY (cod_curso)
 REFERENCES Curso (cod_curso)
);
```

# Resposta Exercício - SQL

• Reserva(cod Equipamento, cod Professor,  
dt reserva, horário, cod\_sala)

```
CREATE TABLE Reserva
(cod Equipamento varchar(5),
 cod Professor varchar (5),
 dt reserva date,
 horario varchar(5),
 cod_sala varchar(5),
 CONSTRAINT reserva_pkey PRIMARY KEY (cod Equipamento, cod Professor,
 dt reserva),
 CONSTRAINT reserva_fkey1 FOREIGN KEY (cod Equipamento)
 REFERENCES Equipamento(cod Equipamento),
 CONSTRAINT reserva_fkey2 FOREIGN KEY (cod Professor)
 REFERENCES Professor(cod Professor),
 CONSTRAINT reserva_fkey3 FOREIGN KEY (cod_sala)
 REFERENCES Sala(cod_sala));
```

# Extração de Dados de uma Tabela

- Consultar dados em uma tabela->SELECT
- Selecionar atributos (Projeção)

```
SELECT <lista de atributos> FROM <tabela>
```

- Exemplo: Listar matrícula, nome e salário de todos os empregados

**Empregado(Matricula, Nome,  
Salario, Num\_Dept)**

```
SELECT Matricula, Nome, Salario  
FROM Empregado
```



# Extração de Dados de uma Tabela

- Projetando todos os atributos:

```
SELECT * FROM <tabela>
```

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Exemplo: Listar todos os empregados

```
SELECT * FROM Empregado
```

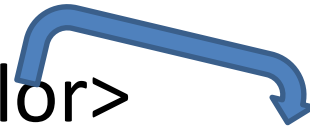
# Extração de Dados de uma Tabela

- Cláusula WHERE: Selecionando tuplas da tabela

```
SELECT <lista de atributos>  
FROM <tabela>  
WHERE <condição>;
```

- <condição>:

- <nome atributo> <operador> <valor>



Relacionais			
<> ou !=	Diferente	=	Igual a
>	Maior que	>=	Maior ou igual a
<	Menor que	<=	Menor ou igual a

Lógicos	
AND	E
OR	Ou
NOT	Não

Pode ser inclusive  
uma consulta

# Extração de Dados de uma Tabela

- Listar nome e salário dos empregados do departamento 020;
- Empregado(Matricula, Nome, Salario, Num\_Dept)**

```
SELECT Nome, Salario  
FROM Empregado WHERE Num_Dept=020;
```

- Listar nome e salário dos empregados do departamento 020 com salário > R\$2000,00

```
SELECT Nome, Salario  
FROM Empregado WHERE Num_Dept=020  
AND Salario > 2000;
```

# Operadores SQL

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- BETWEEN e NOT BETWEEN: substituem o uso dos operadores  $\leq$  e  $\geq$

```
... WHERE <nome atributo> BETWEEN  
      <valor1> and <valor2>;
```

- Listar o nome dos empregados com salário entre R\$1.000,00 e R\$2.000,00.

```
SELECT Nome, Salario  
FROM Empregado WHERE Salario  
      BETWEEN 1000 AND 2000;
```

# Operadores SQL

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- LIKE e NOT LIKE: só se aplicam sobre os atributos do tipo char. Operam como = e <> utilizando os símbolos % (substitui uma palavra) e \_ (substituí um caractere )

```
... WHERE <nome atributo> LIKE <valor>;
```

- Listar os empregados que tem o primeiro nome José

```
SELECT * FROM Empregado WHERE Nome  
LIKE 'José%';
```

# Operadores SQL

Dependente(Matricula, Nome,  
Grau\_Parentesco)

- IN e NOT IN: procuram dados que estão ou não contidos em um dado conjunto de valores

```
... WHERE <nome atributo> in <valores>;
```

- Listar os nomes dos dependentes com grau de parentesco 'M' ou 'P'.

```
SELECT Nome FROM Dependente WHERE  
Grau_Parentesco in ('M','P');
```

# Operadores SQL

Dependente(Matricula, Nome,  
Grau-Parentesco)

- IS NULL e IS NOT NULL: identificam se o atributo tem valor nulo (não informado) ou não;

```
... WHERE <nome atributo> IS NULL;
```

- Listar os nomes dos dependentes com grau de parentesco não definido.

```
SELECT Nome FROM Dependente WHERE  
Grau-Parentesco IS NULL;
```

# Cláusula ORDER BY

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Ordenação dos dados: crescente ou decrescente;

```
... [WHERE <condição>] ORDER BY <nome atributo>  
      <ASC|DESC>
```

- Listar todos os empregados ordenados por nome

```
SELECT * FROM Empregado ORDER BY Nome;
```

- Listar empregados ordenados decendentemente por salário

```
SELECT * FROM Empregado ORDER BY Salario DESC;
```



# Cálculo na cláusula SELECT

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Pode-se criar um campo que não pertença à tabela a partir de cálculos sobre atributos da tabela

Oper. Aritméticos	
+	Adição
-	subtração
*	Multiplicação
/	Divisão

- Mostrar o nome, salário dos empregados com ajuste de 60% para aqueles que ganham menos que R\$1.000,00

```
SELECT Nome, (Salario*1.6)
FROM Empregado WHERE Salario<1000;
```

# Funções de Agregação

Empregado(Matricula, Nome, Salario, Num\_Dept)

- Utilização de funções sobre conjuntos:
  - Disparadas a partir do SELECT
- Mostrar o valor do maior salário dos empregados e o nome do empregado que o recebe

Funções de Agregação	
AVG	Média
MIN	Mínimo
MAX	Máximo
COUNT	Contar
SUM	Somar

Consulta Aninhada

```
SELECT Nome, Salario FROM Empregado
WHERE Salario IN (SELECT MAX(SALARIO)
                  FROM EMPREGADO);
```

# Funções de Agregação

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Mostrar qual o salário médio dos empregados

```
SELECT AVG(Salario) FROM Empregado;
```

- Quantos empregados ganham mais de R\$1.000,00?

```
SELECT COUNT(*)  
FROM Empregado WHERE Salario > 1000;
```

# Cláusula DISTINCT

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Elimina tuplas duplicadas do resultado de uma consulta
- Quais os diferentes salários dos empregados?

```
SELECT DISTINCT(SALARIO)  
FROM Empregado;
```

# Cláusula GROUP BY

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Organiza a seleção de dados em grupos.
- Listar o número do departamento e a quantidade de empregados que nele trabalha;

```
SELECT Num_Dept, COUNT(*)  
FROM Empregado GROUP BY Num_Dept;
```

Atributos do Group By devem aparecer na  
cláusula SELECT

# Cláusula HAVING

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

- Agrupando informação de forma condicional
  - Vem depois do GROUP BY e antes do ORDER BY
- Listar o número total de empregados que recebem salários superior a R\$1.000,00 em cada departamento com mais de 5 empregados que ganham mais que R\$1.000,00

```
SELECT Num_Dept, COUNT(*)  
FROM Empregado  
WHERE Salario>1000  
GROUP BY Num_Dept  
HAVING COUNT(*) > 5;
```

# Uso de “Alias”

- Para substituir nomes de tabelas em comandos SQL
  - São definidos na cláusula FROM

```
SELECT E.nome FROM Empregado E WHERE  
E.Salario>1000;
```



Alias

# Consultas Dados em Várias Tabelas - Junção

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

- Junção de Tabelas (JOIN)
  - Citar as tabelas envolvidas na cláusula FROM;
  - Qualificadores de nomes – utilizados para evitar ambiguidades
- Listar o nome do empregado e do departamento no qual está alocado

```
SELECT E.Nome, D.Nome  
FROM Empregado E, Departamento D  
WHERE D.Num_Dept = E.Num_Dept;
```



## Consultas Dados em Várias Tabelas - Junção

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Dependente(Matricula, Nome,  
Grau\_Parentesco)

- Listar o nome do empregado que tem como dependente o José da Silva

```
SELECT E.*  
FROM Empregado E, Dependente D  
WHERE E.Matricula=D.Matricula AND D.nome='José da  
Silva';
```

- Pode-se utilizar as cláusulas (NOT) LIKE, (NOT) IN, IS (NOT) NULL misturadas aos operadores AND, OR e NOT nas equações de junção (cláusula WHERE)

## Consultas Dados em Várias Tabelas - Junção

**Empregado**(Matricula, Nome, Salario, Num\_Dept)

**Dependente**(Matricula, Nome, Grau\_Parentesco)

**Departamento**(Num\_Dept, Nome )

- Listar o nome do departamento e do empregado que tenha dependente cujo primeiro nome seja José ordenado pelo nome do departamento.

```
SELECT D1.Nome, E.Nome  
FROM Empregado E, Departamento D1, Dependente D2  
WHERE D2.Nome LIKE 'José%' AND D2.Matricula= E.Matricula  
AND D1.Num_Dept=E.Num_Dept  
ORDER BY D1.Nome;
```

## Consultas Dados em Várias Tabelas - Junção

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

- Para cada departamento, liste o nome do departamento, e para cada um deles, listar a matrícula, o nome e o salário de seus empregados, ordenando a resposta.

```
SELECT D.Nome, E.Matricula, E.Nome, E.Salario
FROM Empregado E, Departamento D
WHERE D.Num_Dept= E.Num_Dept
ORDER BY E.Salario DESC, D.Nome;
```

# Consultas Dados em Várias Tabelas - Junção

**Empregado**(Matricula, Nome, Salario, Num\_Dept)      **Departamento**(Num\_Dept, Nome )

**Trabalha**(Mat Empr, Num Proj, Horas)      **Projeto**(Num Proj, Nome, Local)

- Junção em mais de 2 tabelas;
- Listar o nome dos empregados, com seu respectivo departamento que trabalhem mais de 20 horas em algum projeto.

```
SELECT E.Nome, D.Nome
FROM Empregado E, Departamento D, Trabalha T
WHERE T.Horas > 20 AND E.Num_Dept=D. Num_Dept AND
T.Mat_Empr=E.Matricula;
```

## Consultas Dados em Várias Tabelas - Junção

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

Trabalha(Mat Empr, Num Proj,  
Horas)

- Agrupando por meio de mais de um atributo;
- Listar o número de horas trabalhadas em projetos de cada empregado por departamento, informando o nome do departamento e a matrícula do empregado.

```
SELECT D.Nome, E.Matricula, SUM(T.HORAS)
FROM Empregado E, Departamento D, Trabalha T
WHERE E.Num_Dept=D. Num_Dept AND T.Mat_Empr=E.Matricula
GROUP BY D.Nome, E.Matricula;
```

# Junção de Tabelas

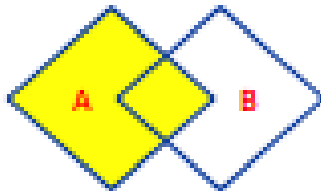
- Inner join (às vezes chamada de "junção simples")
  - É uma junção de duas ou mais tabelas que retorna somente as tuplas que satisfazem à condição de junção
  - Equivalente à junção natural

# Junção de Tabelas

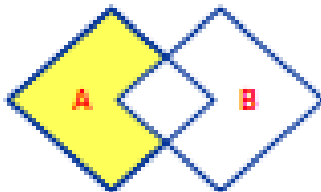
- Outer join

- Retorna todas as tuplas de uma tabela e somente as tuplas de uma tabela secundária onde os campos de junção são iguais (condição de junção é encontrada)
- Para todas as tuplas de uma das tabelas que não tenham tuplas correspondentes na outra, pela condição de junção, é retornado null para todos os campos da lista da cláusula SELECT que sejam colunas da outra tabela

# Junção de Tabelas

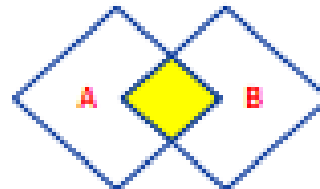


SELECT ...  
FROM A **LEFT JOIN** B  
ON A.key = B.key

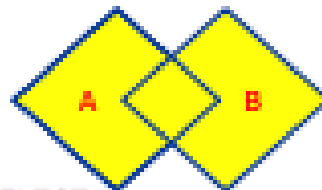


SELECT ...  
FROM A **LEFT JOIN** B  
ON A.key = B.key  
WHERE B.key IS NULL

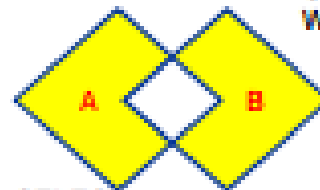
## SQL JOINS



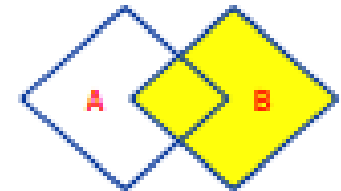
SELECT ...  
FROM A **INNER JOIN** B  
ON A.key = B.key



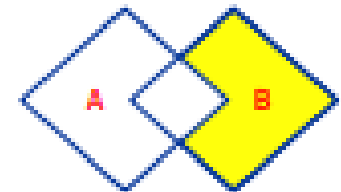
SELECT ...  
FROM A **FULL OUTER JOIN** B  
ON A.key = B.key



SELECT ...  
FROM A **FULL OUTER JOIN** B  
ON A.key = B.key  
WHERE A.key IS NULL  
OR B.key IS NULL



SELECT ...  
FROM A **RIGHT JOIN** B  
ON A.key = B.key



SELECT ...  
FROM A **RIGHT JOIN** B  
ON A.key = B.key  
WHERE A.key IS NULL



# Junção de Tabelas

Empregado(Matricula, Nome, Departamento(Num\_Dept, Nome )  
Salario, Num\_Dept)

- Listar os nomes de todos os departamentos e dos empregados que nele trabalham
- Obs: Pode haver departamentos em que não há empregados alocados

```
SELECT E.Nome, D.Nome  
FROM Departamento D LEFT OUTER JOIN Empregado E  
ON D.Num_Dept=E. Num_Dept;
```

```
SELECT E.Nome, D.Nome  
FROM Empregado E RIGHT OUTER JOIN Departamento D  
ON E.Num_Dept=D. Num_Dept;
```

# Consultas Aninhadas

- O resultado de uma consulta é utilizado por outra forma, de forma encadeada e no mesmo comando SQL;
- O resultado do SELECT mais interno (subselect) é usado por outro SELECT mais externo para obter o resultado final;
- O SELECT mais interno (subconsulta ou consulta aninhada) pode ser usado apenas nas cláusulas WHERE e HAVING do comando mais externo ou em cálculos;
- As subconsultas podem retornar um único valor, uma única linha ou uma tabela.

# Consultas Aninhadas

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

- Subconsulta com operador de igualdade. Listar os empregados que trabalham no departamento de Informática

```
SELECT *  
FROM Empregado  
WHERE Num_Dept in (SELECT Num_Dept  
                    FROM Departamento  
                    WHERE Nome='Informatica');
```

# Consultas Aninhadas

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

- Subconsulta com função agregada. Listar os empregados cujos salários são maiores do que salário médio, mostrando o quanto são maiores

```
SELECT Matricula, Nome, Num_Dept, Salario-(SELECT  
AVG(Salario) FROM Empregado) AS DifSal  
FROM Empregado  
WHERE Salario > (SELECT AVG(Salario)  
FROM Empregado);
```

# Consultas Aninhadas

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

Dependente(Matricula, Nome,  
Grau\_Parentesco)

- Mais de um nível de aninhamento. Listar os dependentes dos empregados que trabalham no departamento de Informática;

```
SELECT D.Matricula, D.Nome, D.Grau_Parentesco
FROM Dependente D
WHERE D.Matricula in
  (SELECT E.Matricula FROM Empregado E WHERE E.Num_Dept
= (SELECT D.Num_Dept FROM Departamento D WHERE
D.nome='Informatica')
);
```

# Cláusulas ANY/SOME

Empregado(Matricula, Nome, Departamento(Num\_Dept, Nome )  
Salario, Num\_Dept)

- Usados com subconsultas que produzem uma única coluna de números;
- ANY/SOME são equivalentes. Retornam True ou False;
- Listar os empregados que possuem salário maior que de pelo menos um empregado do departamento 02

```
SELECT *  
FROM Empregado  
WHERE Salario >= SOME (SELECT Salario  
                        FROM Empregado  
                        WHERE Num_Dept=02);
```

# Cláusulas ALL

**Empregado**(Matricula, Nome, Salario, Num\_Dept)      **Departamento**(Num\_Dept, Nome)

- Utilizado com subconsultas que produzem uma única coluna de número;
- Retorna True ou False;
- Listar os empregados que possuem salário maior que o salário de cada funcionário do departamento 02.

```
SELECT *  
FROM Empregado  
WHERE Salario >= ALL (SELECT Salario  
                      FROM Empregado  
                      WHERE Num_Dept=02);
```

# Cláusulas EXISTS e NOT EXISTS

- Usados APENAS com subconsultas;
- EXISTS:
  - Retorna TRUE SE E SOMENTE SE existe pelo menos uma linha resultado da subconsulta;
  - Retorna FALSE SE E SOMENTE SE a subconsulta produz uma tabela resultante vazia;



# Cláusulas EXISTS e NOT EXISTS

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Departamento(Num\_Dept, Nome )

- Liste todos os empregados do departamento de Informática

```
SELECT *  
FROM Empregado E  
WHERE EXISTS (SELECT D.Num_Dept  
               FROM Departamento D  
               WHERE E.Num_Dept=D.Num_Dept AND  
                     D.Nome='Informatica');
```

# Cláusulas CONTAINS

Empregado(Matricula, Nome,  
Salario, Num\_Dept)

Projeto(Num\_Proj, Nome, Local)  
Trabalha(Mat Empr, Num\_Proj,  
Horas)

- Não faz parte do Padrão SQL;
- Utiliza-se o Exists para simulá-lo;
- Liste os empregados que trabalham em projetos localizados em Quixadá.

```
SELECT * FROM Empregado E
WHERE (SELECT T.Num_Proj
      FROM Trabalha T
      WHERE T.Mat_Empr=E.Matricula)
CONTAINS
(SELECT P.Num_Proj
 FROM Projeto P
 WHERE P.Local='Quixada');
```

# Regras Relativas a Subconsultas

- A cláusula ORDER BY não pode ser usada em uma subconsulta;
- A lista de atributos especificados no SELECT de uma subconsulta deve conter um único elemento (exceto para EXISTS);
- Nomes de atributos especificados na cláusula SELECT da subconsulta estão associados às tabelas listadas na cláusula FROM da mesma;
- É possível referir-se a uma tabela da cláusula FROM da consulta mais externa utilizando qualificadores de atributos;
- Quando a subconsulta é um dos operandos envolvidos em uma comparação, ela deve aparecer no lado direito da comparação;

# Operações de Conjunto

- UNIÃO/INTERSEÇÃO/DIFERENÇA
- As tuplas duplicadas retornadas são removidas do resultado final;
- Tipos de retorno compatíveis;

```
(SELECT <atributo(s)> FROM table A)  
{UNION| INTERSECT| MINUS}  
(SELECT <atributo(s)>FROM table B);
```

# Visões

- Podem ser virtual ou materializada;
- Quando virtual : tabelas virtuais que não ocupam espaço físico;
- Operações:
  - Criação e utilização;
  - Inserção e modificação (semântica depende da definição/natureza da visão)

```
CREATE VIEW <nome da visão> (<lista de atributos>) AS SELECT....
```

# Visões

Empregado(Matricula, Nome, Salario, Num\_Dept)  
Departamento(Num\_Dept, Nome )  
Trabalha(Mat Empr, Num Proj, Horas)

- Construa uma visão dos empregados do que ganham acima de R\$3.000,00, informando os dados do empregado e do departamento

```
CREATE VIEW Dep_02 (Mat, Nom, Salar, Nome-Dept) AS  
SELECT E.Matricula, E.Nome, E.Salario, D.Nome  
FROM Empregado E, Departamento D  
WHERE E.Salario > 3000 AND E.Num-Dept=D.Num-Dept;
```