

# SISTEMAS DE BANCO DE DADOS 2

## AULA 4

SQL - Restrições

MySQL e ORACLE

Vandor Roberto Vilardi Rissoli



# APRESENTAÇÃO

- *Alter Table*
- *Primary, Foreign Key, Unique*
- *Check e Assertion*
- *Select com algumas cláusulas*
  - *Distinct e Alias*
  - *Group by e Order by*
- *Referências*



# SQL

## Instrução ALTER TABLE

Por meio desta instrução é possível :

- adicionar um novo atributo
- modificar uma atributo existente
- eliminar um atributo da relação

**ALTER TABLE <tabela> [tarefa] (informações da tabela);**

ALTER TABLE ESTADOS

ADD (cidade varchar2(10) ); -- inclui novo atributo na tabela

ALTER TABLE ESTADOS -- altera atributo

MODIFY (cidade varchar2(30) );

ALTER TABLE ESTADOS -- apaga atributo

DROP COLUMN cidade); 

Eliminação de um atributo por vez e a relação se mantém sempre com um atributo

# SQL

## Restrições de Integridade

As instruções de integridade são criadas junto com a tabela ou podem ser incluídas depois da sua criação, por meio da instrução **ALTER TABLE**.

Um tipo de restrição, que já foi estudado, é a criação de um atributo obrigatório (*NOT NULL*), em que os atributos devem possuir um valor, não podendo ser NULOS para serem inseridos na tabela.



# SQL

A identificação e a criação de uma chave primária consiste também em uma restrição. Esta restrição (chave primária) cria um recurso relevante na relação para se estabelecer o relacionamento seguro entre relações.



Cada relação só pode possuir uma chave primária, no qual esta chave é formada por um atributo ou conjunto de atributos da relação (chave simples ou composta).



# SQL

Esta restrição impõe a **exclusividade** do atributo (ou conjunto de atributos), além de assegurar que **nenhum** atributo desta chave possua valor NULO.

CONSTRAINT <nome> **PRIMARY KEY** (<atributos>);

Cria-se a chave primária junto com a criação da tabela:

```
CREATE TABLE ESTADO (  
    sigla varchar(2) NOT NULL ,  
    nome varchar(20) ,  
    CONSTRAINT ESTADO_PK PRIMARY KEY(sigla)) ;
```

Tabela criada.

← resultado

Por meio desta instrução foi criada uma relação de nome ESTADO, com uma chave primária de nome ESTADO\_PK, que sempre terá um valor único na relação.

# SQL

A criação pode ser feita para uma relação que já exista, como:

```
DROP TABLE ESTADO;           -- apaga a relação já criada
CREATE TABLE ESTADO (
    sigla varchar(2) NOT NULL,
    nome varchar(20) ) ;
```

Na instrução a seguir é criada a chave primária para a relação ESTADO, por meio de um **ALTER TABLE**.

```
ALTER TABLE ESTADO
    ADD CONSTRAINT ESTADO_PK PRIMARY KEY (sigla) ;
```

Com estas instruções será criada uma relação de nome ESTADO que possui a chave primária **sigla**, em que esta chave primária é responsável pela identificação única de uma tupla na relação ESTADO. Veja o exemplo acima:

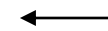
# SQL

Após a criação da relação ESTADO, inicia-se o processo de construção do BD, em que os dados estarão sendo inseridos.

```
INSERT INTO ESTADO VALUES('AM', 'AMAZONAS');  
INSERT INTO ESTADO VALUES('SP','SAO PAULO');  
INSERT INTO ESTADO VALUES('SC','SANTA CATARINA');  
INSERT INTO ESTADO VALUES('AM', 'AMAPA');
```

\*ERRO na linha 1:

Restrição exclusiva (SYS.ESTADO\_PK) violada



resultado

```
INSERT INTO ESTADO VALUES('AM', 'AMAPA');
```

Como pode ser observado acima, a inserção do estado AMAPA não pode ser feita, pois por algum motivo tentou-se inserir, equivocadamente, este estado com a SIGLA de 'AM'. Este atributo é chave primária da relação ESTADO e já apresenta o estado AMAZONAS com esta SIGLA.



# SQL

Realmente a SIGLA AM representa o estado do Amazonas, por isso pode-se imaginar que a inserção iria ser feita de forma errônea, por erro de digitação ou por erro no processo de coleta, ou ainda formação dos dados que compõe esta relação. Com o estabelecimento da restrição na relação, ela identificou o erro que se estaria cometendo e não permitiu o armazenamento, até então solicitado.

Outra restrição importante que utiliza as chamadas chaves candidatas, atributo que possui valor único, mas que não é chave primária, é identificado como atributo de valor único (**UNIQUE**).



# SQL

A restrição **UNIQUE** requer que cada valor do atributo (ou conjunto de atributos) seja exclusivo, ou seja, não se repita (duplicado), como na restrição de chave primária (PRIMARY KEY).

Essa restrição também é chamada de chave exclusiva, quando envolve um único atributo, ou chave exclusiva composta quando envolve mais que um atributo.

Os valores nulos são aceitos nesta restrição, a não ser que o atributo envolvido na restrição (ou os atributos) esteja com a restrição **NOT NULL** descrita.



# SQL

A restrição **UNIQUE** pode ser criada em nível de coluna ou tabela, porém uma chave exclusiva composta só pode ser feita no nível de tabela.

Essa restrição pode ser criada juntamente com a relação ou depois da mesma, por exemplo:

```
DROP TABLE ESTADO;           -- apagando a relação já criada
```

```
CREATE TABLE ESTADO (  
    sigla varchar(2) NOT NULL,  
    nome varchar(20));
```

-- Inserindo a restrição em uma relação já existente

```
ALTER TABLE ESTADO  
    ADD CONSTRAINT ESTADO_UK UNIQUE (nome);
```

→ O **ORACLE** cria um **índice** exclusivo sobre a chave exclusiva para este tipo de restrição.

# SQL

Outra restrição de integridade relacionada a criação e manipulação de chaves é a restrição de integridade referencial, que cria uma chave estrangeira em uma outra relação, ou na própria relação, fazendo o auto-relacionamento.



Por meio de uma CHAVE ESTRANGEIRA, que é formada por um ou mais atributos, é criado um relacionamento com a chave primária de uma outra relação, ou com a mesma relação (auto-relacionamento).

# SQL

A relação que possui a chave primária é chamada de relação pai (ou *master*), enquanto a relação com a chave estrangeira é chamada de filho (ou *detail*). Quando existir um **auto-relacionamento** em uma relação, esta relação é pai e filho ao mesmo tempo.



Suponha que seja necessário conhecer as cidades da Federação e a qual Unidade da Federação (estado) cada uma destas cidades pertencem. Para isso será necessário armazenar dados de cada uma das cidades que formam o BD.

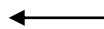


# SQL

Elaborando o projeto do BD que será necessário para atender a necessidade do sistema desejado, constatou-se que os dados relacionados a relação CIDADE deverão ser formados pelos atributos de nome da cidade, quantidade de habitantes, estado que ela pertence, além de um código que a identifique unicamente dentro da relação (sua chave primária), formando assim a relação CIDADE.

```
CREATE TABLE CIDADE (  
    idCidade int                NOT NULL,  -- número inteiro  
    cidade varchar(40)          NOT NULL,  
    qtdeHabitante bigint,  
    estado varchar(2),  
    CONSTRAINT CIDADE_PK PRIMARY KEY (idCidade));
```

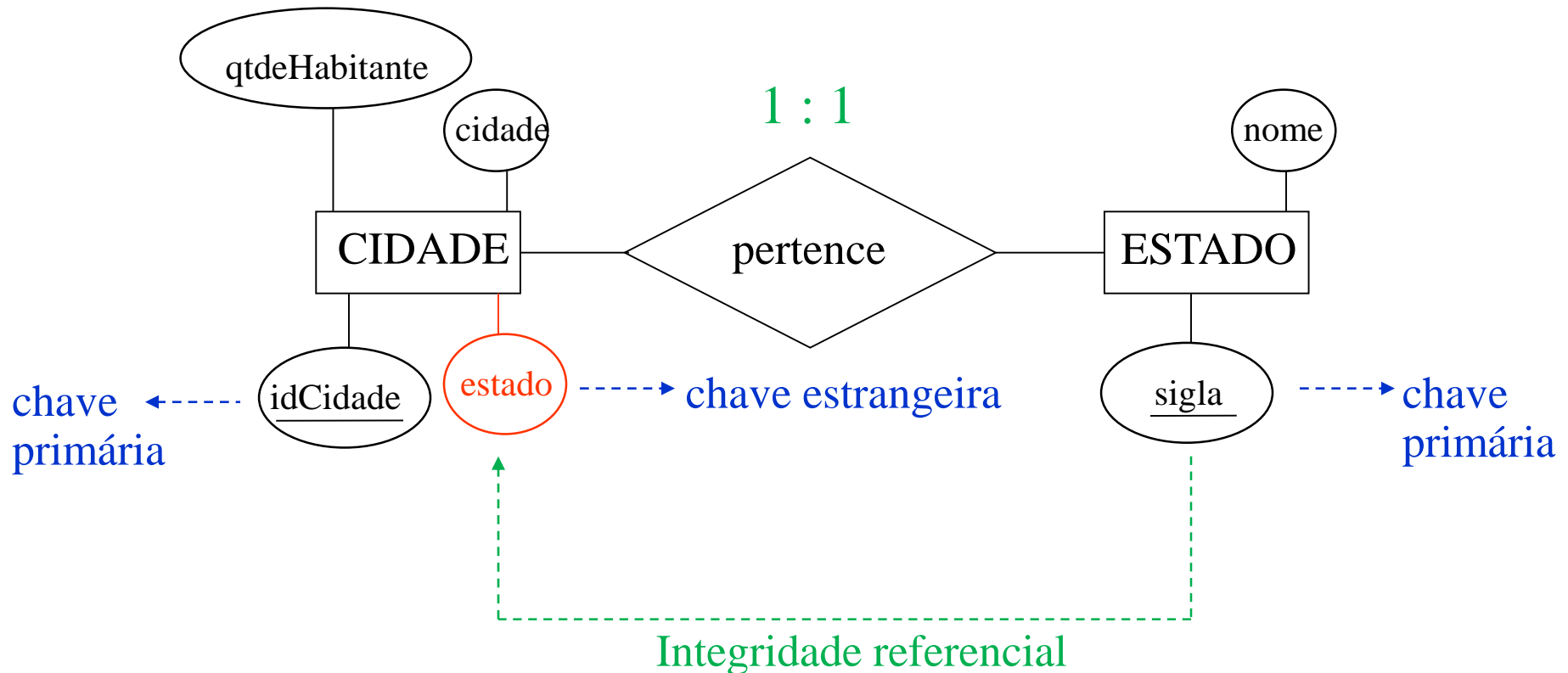
Tabela criada.



resultado

# SQL

Elaborando o ME-R e o seu respectivo DE-R para esta situação tem-se:



O mapeamento deste modelo gerará as seguintes relações, contendo algumas restrições, tais como:

# SQL

A exemplo da chave primária, a chave estrangeira também pode ser criada juntamente com a relação ou separadamente para uma relação que já exista.

```
CREATE TABLE CIDADE (  
    idCidade int NOT NULL,  
    cidade varchar(40) NOT NULL,  
    qtdeHabitante bigint,  
    estado varchar(2),  
    CONSTRAINT CIDADE_PK PRIMARY KEY (idCidade),  
    CONSTRAINT CIDADE_ESTADO_FK FOREIGN KEY (estado)  
        REFERENCES ESTADO (sigla));
```

OU

```
ALTER TABLE CIDADE  
    ADD CONSTRAINT CIDADE_ESTADO_FK  
    FOREIGN KEY (estado) REFERENCES ESTADO (sigla);
```



# SQL

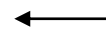
Com esta especificação de restrição uma cidade deverá possuir uma sigla de um estado já cadastrado na relação ESTADO (relação pai). Caso seja informada uma sigla não existente na relação pai a inserção não será realizada.

Ou a chave primária esta especificada corretamente, criando a relação entre as relações, ou ela deve ser nula na relação filho.

```
INSERT INTO CIDADE VALUES(1,'MARÍLIA',300000,'SP');  
INSERT INTO CIDADE VALUES(1,'MARÍLIA',300000,'RR');
```

\* ERRO na linha 1:

Restrição de integridade (SYS.CIDADE\_ESTADO\_FK)  
violada - chave-pai não localizada



resultado



# SQL

```
INSERT INTO CIDADE VALUES(1,'MARÍLIA',300000, 'SC');
```

**\*ERRO na linha 1:**

← resultado

Restrição exclusiva (SYS.CIDADE\_PK) violada

```
INSERT INTO CIDADE VALUES(2,'MARÍLIA',300000, null);
```

```
INSERT INTO CIDADE VALUES(3,'MARÍLIA',300000, 'SC');
```

```
SELECT * FROM CIDADE; -- consultando os dados inseridos
```

IDCIDADE	NOME	QTDE_HABITANTE	SIGLA
----------	------	----------------	-------

← resultado

1	MARÍLIA	300000	SP
2	MARÍLIA	300000	
3	MARÍLIA	300000	SC

```
INSERT INTO ESTADO VALUES('PR', 'PARANA');
```

```
INSERT INTO CIDADE VALUES(4,'MARINGA',400000, 'PR');
```

```
DELETE FROM CIDADE WHERE IDCIDADE = 2;
```

```
DELETE FROM CIDADE WHERE ESTADO = 'SC';
```

# SQL

A chave estrangeira é sempre definida na relação filho e a relação contendo o atributo referenciado (ou atributos) é a relação pai.

Para a definição desta restrição as palavras reservadas **FOREIGN KEY**, **REFERENCES** e **ON DELETE CASCADE** são empregadas, por exemplo:

```
CREATE TABLE CIDADE (  
    idCidade int NOT NULL,  
    cidade varchar(40) NOT NULL,  
    estado varchar(2),  
    CONSTRAINT CIDADE_PK PRIMARY KEY (idCidade),  
    CONSTRAINT CIDADE_ESTADO_FK FOREIGN KEY (estado)  
    REFERENCES ESTADO (sigla) ON DELETE CASCADE);
```

Identifica o(s) atributo(s)  
na relação filho

Identifica a relação pai  
e seu(s) atributo(s)

Identifica que as tuplas da relação filho  
também serão apagadas quando a tupla da  
Relação pai for apagada

# SQL

As opções de restrição relacionadas as tabelas relacionadas por uma chave estrangeira no **MySQL** variam conforme a necessidade da implementação, sendo possíveis:

- **CASCADE**: Atualiza ou exclui os registros da tabela filha automaticamente, ao atualizar ou excluir uma tupla da pai;
- **RESTRICT**: Rejeita a atualização ou exclusão de um registro da tabela pai, se houver registros na tabela filha;
- **SET NULL**: Define como NULL o valor do campo na tabela filha, ao atualizar ou excluir o registro da tabela pai;
- **NO ACTION**: Equivale à opção RESTRICT, mas a verificação de integridade referencial é executada após a tentativa de alterar a tabela. É a opção PADRÃO se nenhuma opção for definida na criação de chave estrangeira;
- **SET DEFAULT**: Define um valor padrão para coluna da tabela filha, aplicado quando um registro da tabela pai for atualizado ou excluído.

# SQL

Uma restrição **CHECK** especifica uma condição que deverá ser satisfeita para cada tupla da relação, podendo ela ser definida a nível de atributo ou relação.

```
CREATE TABLE CIDADE (  
    idCidade int          NOT NULL,  
    cidade varchar(40) NOT NULL,  
    qtddeHabitante bigint,  
    estado varchar(2),  
    CONSTRAINT CIDADE_PK PRIMARY KEY (idCidade),  
    CONSTRAINT HABITANTE_CK CHECK (qtddeHabitante > 0 ) );
```

Um único atributo pode possuir várias restrições **CHECK**, não havendo limite no número destas restrições que podem ser definidas em um atributo (nível só de tupla).



# SQL

A restrição de **CHECK** pode usar as mesmas construções condicionais elaboradas nas consultas (**SELECT**), existindo algumas exceções, por exemplo em **ORACLE**:

- Não são permitidas referências às pseudocolunas CURRVAL, NEXTVAL, LEVEL e ROWNUM;

identificador inteiro do usuário corrente ←

- Nem as chamadas as funções SYSDATE, UID,   
 retorna o nome do usuário corrente no BD ← USER e USERENV; → retorna informações de uma sessão corrente

- Nem as consultas que se referem a outros valores em outras tuplas... mas para isso existem alternativas, entre elas trabalhar com

**ASSERTION** (conteúdo estudado mais a frente)



# SQL

Uma outra expressão que pode ser usada na criação de relações é a **DEFAULT** que insere um valor no atributo quando este não for informado. Observe o exemplo:

```
CREATE TABLE CIDADE (  
    id                int      NOT NULL,  
    cidade            varchar(40) NOT NULL,  
    aniversario       date DEFAULT SYSDATE NOT NULL ,  
    habitante         bigint,  
    CONSTRAINT CIDADE_PK PRIMARY KEY (id),  
    CONSTRAINT HABITANTE_CK CHECK (habitante > 0 ));
```

Não precisa ser  
**NOT NULL**



**SYSDATE** → função que “captura” a data do sistema (SO) em **ORACLE**.

```
INSERT INTO CIDADE VALUES(1,'Brasília','20-apr-2001', 2000000);
```

```
INSERT INTO CIDADE(ID,CIDADE,HABITANTE)  
VALUES (2,'São Paulo',4000000);
```

# SQL

O **MySQL** ainda não suporta **Check Constraint**, mas vários outros servidores de banco de dados (**ORACLE**, Postgresql, SQL Server, etc.) tem este recurso disponível. Veja um exemplo de implementação o endereço <https://youtu.be/t5M0sOo5rsY> seguir que mostra mais uma possibilidade.

Verificar na versão 8  
do MySQL

<https://youtu.be/t5M0sOo5rsY>

Para o **MySQL** podem ser observadas algumas funcionalidades interessantes para captura da data atual do servidor.

**CURDATE()** => fornece só a data no padrão AAAA-MM-DD

**NOW()** => fornece data e horário no padrão

AAAA-MM-DD HH:MM:SS





# Consultando o Banco de Dados

Elabore uma consulta que permita visualizar todos os dados do esquema ESTADO e CIDADE.

Para que todos os dados das duas relações sejam apresentados é necessário realizar uma operação de junção entre elas.

Apesar de existir o relacionamento entre as duas relações, este relacionamento só será usado na apresentação dos dados (via SELECT por exemplo), somente se for especificada a expressão de junção na consulta, por exemplo:

```
SELECT nome, sigla FROM ESTADO;
```

```
SELECT idCidade, cidade, habitante, aniversario  
FROM CIDADE;
```

Estas consultas apresentarão todos os dados de todas as tuplas de cada relação. Porém a solicitação feita acima deseja apresentar todos estes dados já relacionados.

# SQL

Para isso utiliza-se a junção entre as relações, que podem ser feitas para mais que duas relações, caso exista esta necessidade.

```
SELECT nome, sigla, idCidade, cidade, qtdeHabitante, estado
FROM ESTADO, CIDADE                -- consulta sobre duas relações
WHERE sigla = estado;              -- realizando a junção entre as duas
```

Apresentando os dados de maneira mais organizada, pode-se classifica-los como for mais interessante, por exemplo, separar por **sigla** primeiro e dentro das siglas por uma ordenação alfabética do nome da cidade.

```
SELECT nome, sigla, idCidade, cidade, qtdeHabitante, estado
FROM ESTADO, CIDADE                -- realizando a junção entre as duas
WHERE sigla = estado               -- ordenando de forma crescente a
ORDER BY sigla, cidade;            -- apresentação do resultado
```

# SQL

No intuito de evitar a compreensão equivocada do compilador SQL e a incorreta execução, ou mesmo o erro de compilação da declaração SQL desejada, será abordada algumas regras de sintaxe importantes para a escrita correta de uma instrução SQL.

→ Crie as 2 relações a seguir, respeitando as descrições:

## RELAÇÃO ESTADO

sigla CHARACTER(2)

nome CHARACTER(20)

## RELAÇÃO CIDADE

codigo NUMÉRICO(5)

nome CHARACTER(50)

sigla CHARACTER(2)

habitantes NUMÉRICO

→ Agora, insira **3 tuplas** válidas na relação ESTADO e mais **7 tuplas** na relação CIDADE, onde ao menos uma tupla seja de cada estado cadastrado na relação ESTADO.

# SQL

- Garanta que sua base de dados NÃO ACEITARÁ nome de cidades iguais para um mesmo Estado no projeto.
- Realize algumas CONSULTAS sobre estes dados, sendo um SELECT para cada solicitação abaixo:
  - A) Projeção de sigla e nome do estado de MS e DF, sendo mostrado uma vez só se tiverem repetidas na tabela de cidades;
  - B) Selecione só o nome das cidades e a sigla dos estados de DF, GO e MS sobre as cidades em ordem alfabética do estado e em cada um por nome de cidade;
  - C) Selecione todas as cidades do **primeiro** estado cadastrado mostrando só o nome da cidade, nome e sigla do estado;
  - D) Consulte o nome e a sigla do estado cadastrado por **último** e todas as cidades cadastradas dele, mostrando seu nome e a quantidade de habitantes em ordem alfabética estado de nome e de cidade.

# SQL

O resultado padrão a ser apresentado por uma instrução **SELECT** exibi todos os atributos solicitados em todas as tuplas selecionadas, o que pode incluir tuplas duplicadas.

Caso isso não seja desejável por uma determinada consulta, o qualificador **DISTINCT** deve ser inserido na instrução **SELECT** que será executada. Este qualificador vem logo após a palavra-reservada **SELECT**, onde em seguida podem ser especificados todos os atributos que se deseja apresentar como resultado desta consulta.

Observe o exemplo: suponha que a relação cidade possua 6 tuplas, sendo três cidades do estado de São Paulo.

```
SELECT sigla  
FROM JOSE.CIDADE  
WHERE sigla = 'SP';
```

**Esta consulta selecionará somente as cidades com a sigla do estado de São Paulo (SP), mostrando assim três tuplas idênticas.**

# SQL

Este tipo de consulta também poderia incluir o qualificador **ALL** no início do SELECT, porém esta qualificação já é padrão quando um qualificador não é especificado, pois ela estabelece que todas as tuplas selecionadas devem ser apresentadas.

```
SELECT ALL sigla  
FROM JOSE.CIDADE  
WHERE sigla = 'SP';
```

Para que os valores selecionados não sejam apresentados em duplicidade o qualificador **DISTINCT** deve ser colocado no local do **ALL**.

```
SELECT DISTINCT sigla  
FROM JOSE.CIDADE  
WHERE sigla = 'SP';
```

Esta consulta apresentará uma única tupla com o valor da sigla SP, apesar de existirem mais que uma.

# SQL

Quando os atributos solicitados não formarem uma tupla **totalmente** única eles serão apresentados normalmente.

```
SELECT DISTINCT sigla, codigo  
FROM JOSE.CIDADE  
WHERE sigla = 'SP';
```

O resultado apresentado consistirá de todas as tuplas que possuem a **sigla** SP, com códigos diferentes, pois se existir alguma sigla igual com código igual o qualificador **DISTINCT** não permitirá a sua apresentação.



# SQL

## FUNÇÕES DE GRUPO

As funções de grupo trabalham com conjunto(s) de tuplas, ao invés de uma única tupla por vez, fornecendo o resultado do SELECT também avaliado por grupo.

As funções de grupo são:

**AVG** - valor médio do grupo, **ignorando os atributos nulos**

**COUNT** - conta quantidade de tuplas (\* - inclui nulos e duplicat.)

**MAX** - valor máximo **ignorando o nulo**

**MIN** - valor mínimo **ignorando o nulo**

**STDDEV** - desvio padrão dos valores selecionados, **sem nulos**

**SUM** - valores somados **ignorando os nulos**

**VARIANCE** - variação dos valores selecionados, **sem nulos**





# SQL

As funções de grupo **ignoram valores nulos** quando o atributo for usado para agrupar valores.

A exceção é a função COUNT que pode ser usada com o parâmetro pré-definido \* (asterisco) que solicitará a quantidade de tuplas que existe na relação referenciada.

```
SELECT COUNT(*)  
FROM JOSE.CIDADE;
```

Todas as demais funções de grupo **ignoram os valores nulos**, porém estas podem fazer uso também da função **NVL( )**, caso seja necessário trabalhar com nulos no ambiente **ORACLE**.




# SQL

## FUNÇÃO NVL()

Esta função permite o tratamento de atributos que **possuam o valor nulo armazenado**.

Ela pode ser aplicada sobre qualquer tipo de dado (numérico, caracter, data), onde o atributo que possuir seu valor nulo armazenado retornará um outro dado definido para este tipo específico.

**NVL**(**<expr1>**,**<expre2>**)

valor de origem ou expressão que possa conter nulo

valor de conversão para nulo que será retornado pela função

Exemplo de conversão:

NUMÉRICO = **NVL**(**<coluna numérica>** , 0)

DATA = **NVL**(**<coluna de data>** , '01-JAN-95')

CARACTER = **NVL**(**<coluna caracter>** , 'indisponível')

# SQL

## CRIANDO GRUPOS DE DADOS

Tudo que foi feito até aqui tratava todos os dados de uma relação como um único grupo de dados, mas as vezes é necessário dividir estas relações em grupos menores.

Procuram realizar estas divisões nos dados é que se pode incluir a cláusula **GROUP BY** em uma instrução SELECT.

É possível aplicar uma cláusula **GROUP BY** em uma seleção e usar uma função de grupo para **sumarizar** os valores de cada um dos grupos formados.



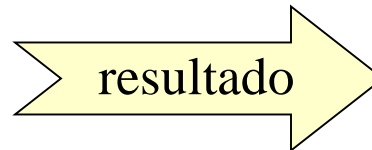
# SQL

Suponha que a consulta abaixo seja executada e o resultado seja o apresentado a seguir.

O que isso significa?

O que isso quer dizer?

```
SELECT SIGLA, COUNT(sigla)
FROM CIDADE
GROUP BY sigla;
```



SIGLA COUNT(SIGLA)	
----	-----
CE	1
DF	2
SP	3

A relação CIDADE foi dividida em três grupos por meio da sigla, onde é considerada somente as siglas que estão cadastradas e são apresentadas por meio da seleção do atributo **sigla**, enquanto que a função **COUNT**, sobre a sigla, mostra a quantidade de tuplas que existe em cada sigla (alguns grupos).

# SQL

Sempre que existir um ou mais atributos individuais na seleção, juntamente com a aplicação de funções de grupo, será necessário incluí estes atributos no grupo por meio da cláusula **GROUP BY**.

```
SELECT sigla, COUNT(sigla)
FROM CIDADE
GROUP BY sigla;
```

→ um atributo simples ou comum

A seleção sem especificação de nenhum atributo individual também é possível, mas será realizada por toda a relação se não for especificada a cláusula **GROUP BY**. Observe os resultados em seu BD aplicando as duas sugestões a seguir.

```
SELECT COUNT(sigla)
FROM CIDADE
GROUP BY sigla;
```

≠

```
SELECT COUNT(sigla)
FROM CIDADE;
```

# SQL

O atributo especificado no GROUP BY não precisa estar no SELECT, mas normalmente seu resultado fica sem sentido devido a dificuldade de compreensão dos dados agrupados.

```
SELECT COUNT(sigla)
FROM CIDADE
GROUP BY sigla;
```

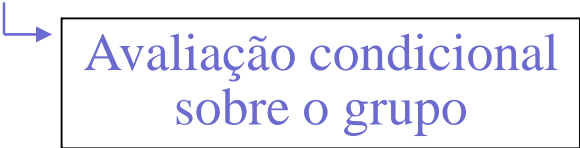
Pode também existir a necessidade de agrupar mais que um atributo, onde uma relação é agrupada primeiro por um atributo e dentro deste também é realizado o agrupamento por um outro atributo. Imagine o exemplo:

```
SELECT departamento, empregado, SUM(salario)
FROM EMPREGADO
GROUP BY departamento, empregado;
```

# SQL

A utilização da cláusula WHERE não pode ser aplicada na restrição de grupos, mas a inclusão da cláusula HAVING pode avaliar cada grupo e permitir a sua apresentação ou não.

```
SELECT departamento, AVG(salario)
FROM EMPREGADO
GROUP BY departamento
HAVING AVG(salario > 2000);
```



Avaliação condicional  
sobre o grupo



cláusula de avaliação onde os valores  
que o satisfação são apresentados

→ Como é usada a cláusula WHERE na restrição de tuplas sobre atributos individuais, a cláusula **HAVING** também pode ser empregada, mas sobre funções de grupo da consulta.

# SQL

Pode ser usada a cláusula **GROUP BY** sem uma função de grupo no SELECT. Porém, a apresentação dos resultados sempre acontecerão em ordem crescente quando um cláusula de grupo estiver na instrução.

Um **ORDER BY** pode ser inserido, mas por padrão a apresentação sempre será em ordem crescente. A aplicação do **ORDER BY** seria coerente para a alteração desta apresentação, onde por exemplo o resultado desejado seria melhor empregado em uma ordem decrescente, ou mesmo sobre outros atributos da consulta.

```
SELECT departamento, AVG(salario)
FROM EMPREGADO
GROUP BY departamento
HAVING AVG(salario > 2000)
ORDER BY departamento;
```



# Exercícios de Fixação

- 1.) Elabore um conjunto de relações que permita armazenar e consultar os dados de alunos matriculados em um curso. Para isso, identifique quais são os dados necessários sobre o aluno e o curso, elaborando o ME-R e o respectivo DE-R do modelo proposto. Em seguida realize o mapeamento do DE-R para as relações que formarão o BD com as restrições estudadas até o momento. Implemente-os no ambiente SQL e insira alguns dados. Por fim, elabore as 3 consultas:
- a)** todos os alunos matriculados;
  - b)** todos os cursos disponíveis na instituição;
  - c)** todos os alunos com a identificação completa do curso que ele esta cursando.

# Exercícios de Fixação

2.) Usando o ME-R elabore um projeto que implemente um BD para o controle de funcionários de uma empresa. Este controle visa gerenciar as atividades de cada funcionário que trabalha em um departamento específico dentro de uma única empresa. O motivo deste controle é reconhecer o envolvimento de cada departamento nos diferentes projetos que a empresa realiza, identificando com isso os departamentos envolvidos em cada projeto e os funcionários destes departamentos que fazem parte do projeto pesquisado. Os dados disponíveis dos projetos são sempre o nome, a data prevista de início e fim do projeto, além do funcionário responsável pelo projeto. Os departamentos possuem nome, além da identificação dos funcionários que o compõe.

# Exercícios de Fixação

... continuação do exercício 2

Os funcionários são contratados após a coleta dos dados pessoais (nome, CPF, carteira profissional, telefones e endereço) de cada um, além da identificação do cargo que ele estará ocupando, seu salário mensal e o departamento ao qual ele será alocado. Elabore também as seguintes consultas para atender a necessidade desta empresa:

- a) Identificação nominal do funcionário e seu endereço para uso do serviço de mala direta (etiqueta para correio);
- b) Os funcionários que pertencem a um departamento;
- c) Relatório de custos separado por departamento identificando os salários dos funcionários que formam tal departamento;
- d) Os departamentos envolvidos em um projeto;
- e) Os funcionários envolvidos em um projeto.

# SQL – Conhecendo SEQUENCE

## Criando Sequências

Alguns SGBDs usando de recursos diferentes para geração de valores sequências que podem ser empregados com segurança, por exemplo, como chaves primárias. Uma sequência (*sequence*) é um objeto do SGBD que gera números sequenciais e respeita as características fornecidas por parâmetros no momento de sua criação.

**CREATE SEQUENCE** <nome\_da\_sequência>

Parâmetros (**ORACLE**)

increment by    /    start with

maxvalue ou nomaxvalue    /    minvalue ou nominvalue

cycle ou nocycle    /    cache ou nocache

# SQL – Conhecendo SEQUENCE

OPÇÃO	Descrição
nome_da_sequência	Nome da sequencia, não podendo ser o mesmo de uma tabela
Increment by <i>n</i>	Especifica de quanto será o incremento ou decremento. O padrão é 1
Start with <i>n</i>	Especifica o primeiro número a ser gerado. O padrão é 1.
Maxvalue <i>n</i>	Especifica o valor máximo que a sequência gerada pode atingir. O padrão é nomaxvalue, indo até 1027
Minvalue <i>n</i>	Especifica o valor mínimo para as sequências que estiverem sendo decrementadas. É mutuamente exclusiva ao maxvalue.
Cycle   nocycle	Indica que ao atingir o valor máximo a numeração continuará a partir do valor inicial. O default é nocycle.
Cache <i>n</i>   nocache	Especifica quantos valores o bando de dados pré-aloca e mantém em memória. O padrão é 20.



# SQL – Conhecendo SEQUENCE

Exemplo:

```
CREATE SEQUENCE SEQ_PESSOA  
    minvalue 1  
    maxvalue 9999999999  
    start with 1  
    increment by 1  
    nocache  
    cycle ;
```



# SQL – Conhecendo SEQUENCE

Utilizando a sequence

```
SELECT SEQ_PESSOA.nextval
```

Caso for testar no banco de dados ou outro programa acrescente um **from dual** para obter o resultado.

```
SELECT SEQ.nextval FROM DUAL;
```

A recuperação do valor atual acontece com a substituição do **nextval** por **currval**

```
SELECT SEQ.currval FROM DUAL;
```



# SQL – Conhecendo SEQUENCE

Usando como uma chave para inserir novo registro em uma tabela:

```
INSERT INTO pessoa  
  (SEQUENCIA, NOME, IDADE) VALUES (  
    SEQ_PESSOA.Nextval, 'José Silva', 25);
```

Para listar as *sequences* do usuário pode se usar a instrução:

```
SELECT * FROM user_sequences;
```





# Referência de Criação e Apoio ao Estudo

## Material para Consulta e Apoio ao Conteúdo

- ELMASRI, R. e Navathe, S. B., Fundamentals of Database Systems, Addison-Wesley, 3rd edition, 2000
  - Capítulo 8
- SILBERSCHATZ, A. & Korth, H. F., Sistemas de Banco de Dados.
  - Capítulo 4
- SUNDERRAMAN, R., Oracle Programming: A Primer, Addison Wesley, 1999.
  - Capítulo 2
- Universidade de Brasília (UnB Gama)
  - <https://sae.unb.br/cae/conteudo/unbfga>  
(escolha a disciplina **Sistemas de Banco de Dados 1**)
- Oracle: SQL Assertions/Declarative multi-row constraints
  - <https://community.oracle.com/ideas/13028>