Reforçando conceitos de SQL

Estudo de Caso:
Agenciamento de Corridas de Táxis

Algumas Entidades existentes Estudo de Caso

Cliente	**Corrida Agendada**	**Fila**
Logradouro	**Zona**	**Motorista**
Corrida Efetivada	**Táxi**	

Consulta Básica SQL

SELECT * FROM Cliente

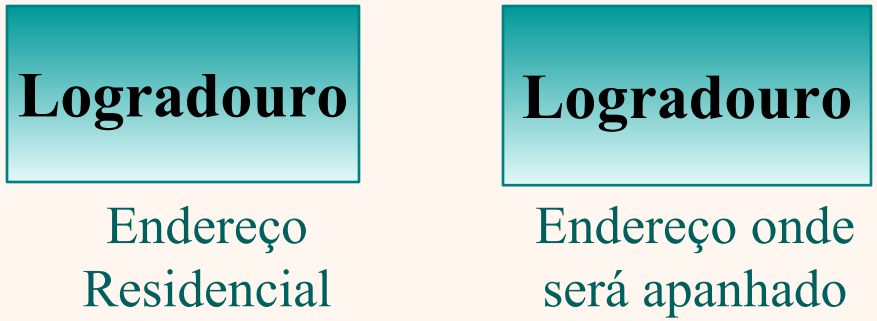
SELECT C.nome FROM Cliente C

Variável tupla

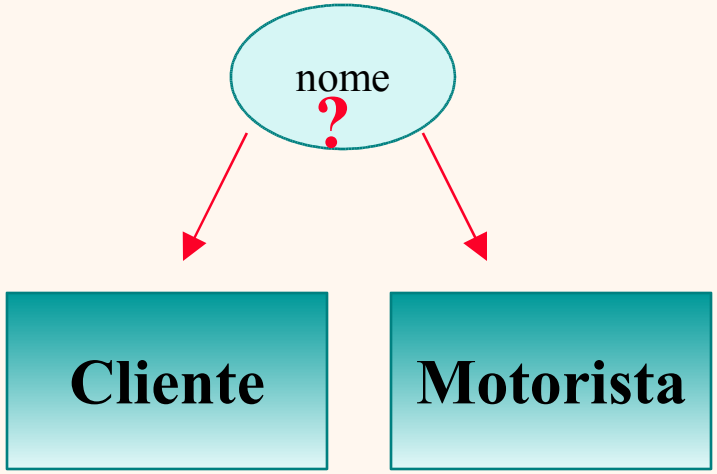
- Problema: é necessário conhecer o estrutura da tabela Cliente para saber quais os atributos o SELECT está se referindo.
- Quando SELECT * é um bom estilo:
  - COUNT(*)
  - Consultas Aninhadas Correlacionadas.

Variáveis Tupla ("apelidos")

- NECESSÁRIO quando:
  - Mesma relação aparece mais de uma vez na cláusula FROM.



  - Mesmo nome de atributos em relações diferentes.



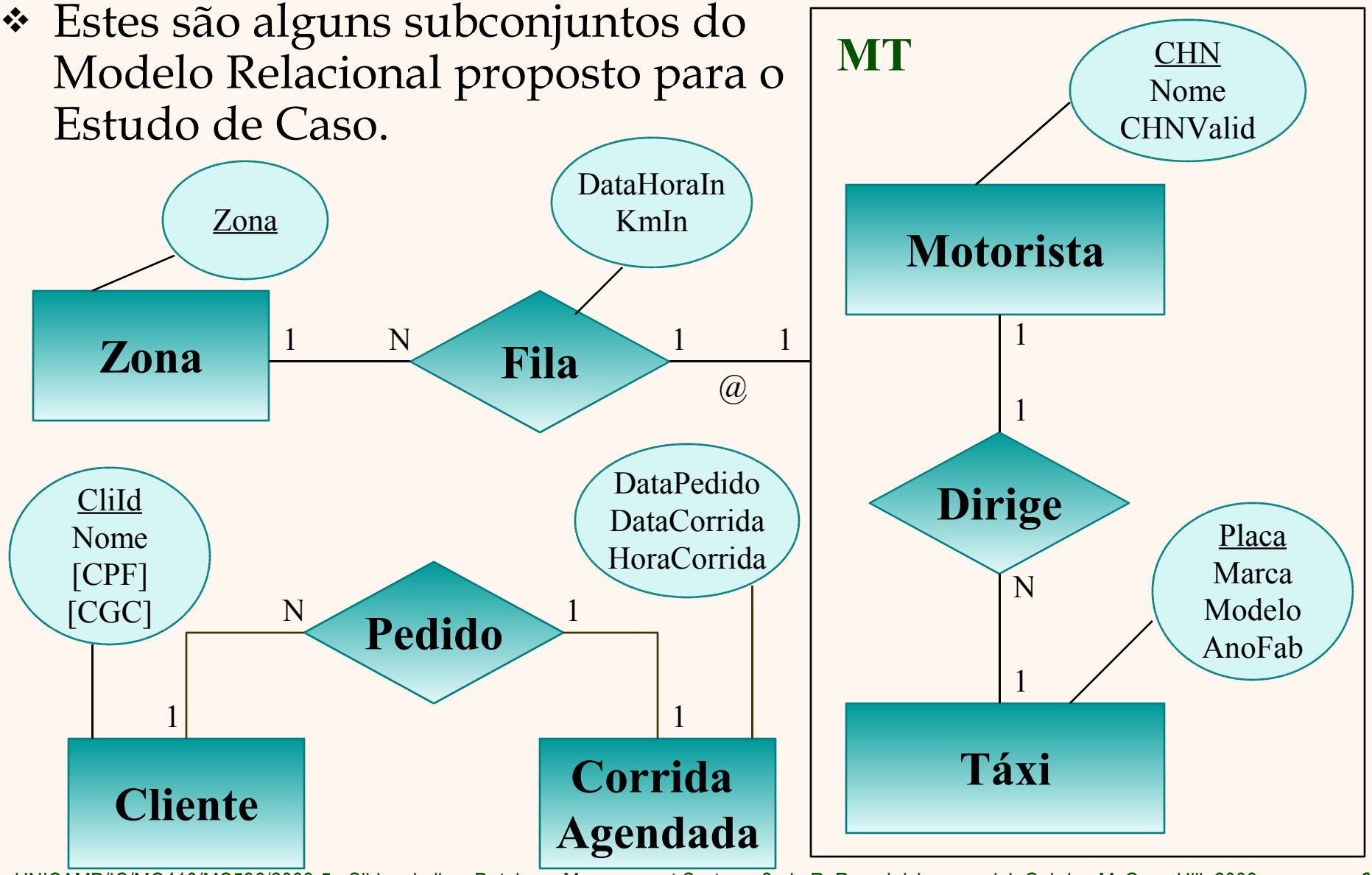
Variáveis Tupla ("apelidos")

- São RECOMENDADAS para:
  - Melhorar LEGIBILIDADE das consultas – através do SELECT sabe-se de qual relação o atributo pertence.
  - Facilitar escrita da consulta SQL (consultas muito grandes ou tabelas com nomes grandes)

Ex: Os motoristas-táxi que fizeram uma corrida no dia 15/01/2004, entre 10:00 e 12:00, na cidade de Campinas.

Modelo Relacional

- Estes são alguns subconjuntos do Modelo Relacional proposto para o Estudo de Caso.



Estratégia de Avaliação Conceitual

- Nomes dos clientes que agendaram pelo menos uma corrida no mês de Janeiro.

```
SELECT C.nome
FROM Cliente C, CorridaAgendada A
WHERE C.cliId = A.cliId AND
 MONTH(dataCorrida) = 1
```

Álgebra relacional

Projeção = SELECT

Produto cartesiano = FROM

Selecão = WHERE

Cliente

cliId	nome	CPF	CGC
1111	Juliana	21788878904	*Null*
2222	Maria	21455588899	*Null*
3333	Pedro	34567891000	*Null*
4444	Renato	45678567857	*Null*

Corrida Agendada

cliId	codLog	num	data Pedido	data Corrida	hora Corrida
1111	11121	123	14/01/2004	15/01/2004	15:00
1111	41231	77	25/01/2004	27/01/2004	18:15
2222	22212	12	10/02/2004	10/02/2004	08:00
3333	33121	65	10/01/2004	27/01/2004	13:00

Estratégia de Avaliação Conceitual

(1) Produto Cartesiano

clild	nome	CPF	CGC	clild	codLog	num	data Pedido	data Corrida	Hora Corrida
1111	Juliana	21788878904	*Null*	1111	11121	123	14/01/2004	15/01/2004	15:00
1111	Juliana	21788878904	*Null*	1111	41231	77	25/01/2004	27/01/2004	18:15
1111	Juliana	21788878904	*Null*	2222	22212	12	10/02/2004	10/02/2004	08:00
1111	Juliana	21788878904	*Null*	3333	33121	65	10/01/2004	27/01/2004	13:00
2222	Maria	21455588899	*Null*	1111	11121	123	14/01/2004	15/01/2004	15:00
2222	Maria	21455588899	*Null*	1111	41231	77	25/01/2004	27/01/2004	18:15
2222	Maria	21455588899	*Null*	2222	22212	12	10/02/2004	10/02/2004	08:00
2222	Maria	21455588899	*Null*	3333	33121	65	10/01/2004	27/01/2004	13:00
3333	Pedro	34567891000	*Null*	1111	11121	123	14/01/2004	15/01/2004	15:00
3333	Pedro	34567891000	*Null*	1111	41231	77	25/01/2004	27/01/2004	18:15
3333	Pedro	34567891000	*Null*	2222	22212	12	10/02/2004	10/02/2004	08:00
3333	Pedro	34567891000	*Null*	3333	33121	65	10/01/2004	27/01/2004	13:00
4444	Renato	45678567857	*Null*	1111	11121	123	14/01/2004	15/01/2004	15:00
4444	Renato	45678567857	*Null*	1111	41231	77	25/01/2004	27/01/2004	18:15
4444	Renato	45678567857	*Null*	2222	22212	12	10/02/2004	10/02/2004	08:00
4444	Renato	45678567857	*Null*	3333	33121	65	10/01/2004	27/01/20	13:00

(2) Condição (3) Seleção

Estratégia de Avaliação Conceitual

Resultado

nome
Juliana
Juliana
Pedro

(4) Eliminar Linhas duplicadas (DISTINCT)

nome
Juliana
Pedro

Expressões e String

- Expressões são compostas por operadores aritméticos, lógicos, de comparação e de string. Abaixo, exemplos de expressões com comparação.

```
SELECT T.Placa
FROM Taxi T
WHERE T.placa LIKE ‘ _ _ _ _ _ _ _ 5’
```

Táxi com placa final 5

```
SELECT A.DataCorrida
FROM CorridaAgendada A
WHERE A.HoraCorrida
BETWEEN ’08:00:00’ AND ’12:00:00’
```

Operadores de comparação

Para comparações entre strings, a ordem usual é determinada alfabeticamente. Porém o SQL suporta outras formas de ordenação de string (conceito de *collation* ou *sort order*).

Collation: maneira como são armazenados os tipos de string. Permite que você especifique o que é "menos que" outros.

Ex: se s*ort order* for *case-sensitive* e todos os modelos estivessem cadastrados com letra minúscula na tabela de Táxi, a consulta abaixo retornaria vazia.

```
SELECT T.placa
FROM Taxi T
WHERE T.modelo = 'PALIO'
```

Operadores

Operadores de comparação-conjunto

ALL = todos
ANY = pelo menos um
= ANY equivale ao IN
< > ALL equivale a NOT IN

```
SELECT cliId FROM CorridaAgendada A1
WHERE A1.horaCorrida > ANY
    (SELECT A2.horaCorrida FROM CorridaAgendada A2
    WHERE A2.cliId = 1111 AND dataPedido = '25/01/2004')
```

cliId	codLog	num	data Pedido	data Corrida	hora Corrida
1111	11121	123	25/01/2004	25/01/2004	13:00
1111	41231	77	25/01/2004	25/01/2004	18:15
3333	33121	65	24/01/2004	25/01/2004	16:00
4444	19621	14	23/01/2004	25/01/2004	20:45

Se fosse > ALL, retornaria somente cliId = 4444

UNION, INTERSECT e EXCEPT

- Motoristas que possuem táxi com ano de fabricação = 2000 ou 2001

UNION

Motorista-Táxi

CNH	Nome	CNHValid	Placa
01111111111	Paulo	S	AAA1111
02222222222	Lucas	S	CCC3333
03333333333	Cida	S	EEE5555
04444444444	Marcos	N	AAA1111
05555555555	Joana	S	BBB2222

O conjunto de operações estão disponíveis na álgebra relacional. A principal diferença é que, no SQL, elas são multiconjuntos de operações.

Táxi

Placa	Marca	Modelo	AnoFab
AAA1111	Fiat	Palio	2000
BBB2222	Volkswagen	Santana	1998
CCC3333	Volkswagen	Gol	2000
DDD4444	Volkswagen	Gol	1998
EEE5555	Ford	Fiesta	2001

O resultado não possuirá linhas repetidas a não ser que a opção ALL seja especificada.

UNION, INTERSECT e EXCEPT

- Motoristas que possuem táxi com ano de fabricação = 2000 e 2001

Motorista-Táxi

CNH	Nome	CNHValid	Placa
01111111111	Paulo	S	AAA1111
02222222222	Lucas	S	CCC3333
03333333333	Cida	S	EEE5555
04444444444	Marcos	N	AAA1111
05555555555	Joana	S	BBB2222

INTERSECT

Táxi

Placa	Marca	Modelo	AnoFab
AAA1111	Fiat	Palio	2000
BBB2222	Volkswagen	Santana	1998
CCC3333	Volkswagen	Gol	2000
DDD4444	Volkswagen	Gol	1998
EEE5555	Ford	Fiesta	2001

UNION, INTERSECT e EXCEPT

- Motoristas que possuem táxi com ano de fabricação = 2000 mas não possuem táxi com ano de fabricação = 2001

Motorista-Táxi

CNH	Nome	CNHValid	Placa
01111111111	Paulo	S	AAA1111
02222222222	Lucas	S	CCC3333
03333333333	Cida	S	EEE5555
04444444444	Marcos	N	AAA1111
05555555555	Joana	S	BBB2222

EXCEPT

Táxi

Placa	Marca	Modelo	AnoFab
AAA1111	Fiat	Palio	2000
BBB2222	Volkswagen	Santana	1998
CCC3333	Volkswagen	Gol	2000
DDD4444	Volkswagen	Gol	1998
EEE5555	Ford	Fiesta	2001

Motorista pode ter apenas um táxi.

Representa o "menos" da álgebra relacional.

Consulta Aninhada

- Nomes dos motoristas que não estão aguardando na fila de espera na zona Leste.

```
SELECT M.nome
FROM Motorista M
WHERE M.CHN NOT IN
        (SELECT F.CHN
        FROM Fila F
        WHERE F.zonaId IN
                (SELECT Z.zonaId
                FROM Zona Z
                WHERE Z.nome = 'Leste')
```

Trocar por:

IN	e	NOT IN
NOT IN		NOT IN

Sub-consulta será recomputada quantas vezes o número de linhas da consulta imediatamente superior.

Consultas Aninhadas Correlacionadas

- Nomes do motoristas que possuem um táxi da marca Fiat

```
SELECT M.nome
FROM Motorista M
WHERE EXISTS
    (SELECT * FROM Taxi T
    WHERE T.Marca = 'Fiat'
    AND T.placa = M.placa)
```

EXISTS: Compara sub-consulta com vazio!

Se sub-consulta não for vazia, retorna VERDADEIRO.

Quer checar se tupla existe e não realmente trazer todos os atributos.

Operações Agregadas

cliId	codLog	num	data Pedido	data Corrida	hora Corrida
1111	11121	123	25/01/2004	25/01/2004	13:00
1111	41231	77	25/01/2004	25/01/2004	18:15
3333	33121	65	24/01/2004	25/01/2004	16:00
4444	19621	14	23/01/2004	25/01/2004	20:45

```
SELECT COUNT(*)
FROM CorridaAgendada A
```

Resultado: 4

```
SELECT COUNT(DISTINCT A.cliId)
FROM CorridaAgendada A
```

Resultado: 3

GROUP BY

Esta consulta é inválida !

```
SELECT T.marca, MAX (T.anoFab) AS anoMaisNovo
FROM Taxi T
```

Correto:

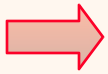
```
SELECT T.marca, MAX (T.anoFab)  AS anoMaisNovo
FROM Taxi T GROUP BY T.marca
```

Regra: Sempre que houver operações agregadas na cláusula SELECT, os dados (cujos atributos que não contêm operações) têm que ser agrupados.

GROUP BY e HAVING

```
SELECT T.marca, MAX (T.anoFab) AS anoMaisNovo
FROM Taxi T
WHERE T.anoFab BETWEEN 2000 AND 2002
GROUP BY T.marca
```

Placa	Marca	Modelo	AnoFab
FJC1234	Fiat	Palio	2000
GCG4321	Volkswagen	Santana	1997
HAH5775	Volkswagen	Gol	2001
IJI5765	Volkswagen	Gol	1998
JOP2888	Ford	Fiesta	2001
KLK9191	Ford	Escort	2002
LUI3433	Ford	Fiesta	1998



Marca	anoMaisNovo
Fiat	2000
Ford	2002
Volkswagen	2001

Adicionando HAVING COUNT(*) > 1 após cláusula GROUP BY

Marca	anoMaisNovo
Ford	2002

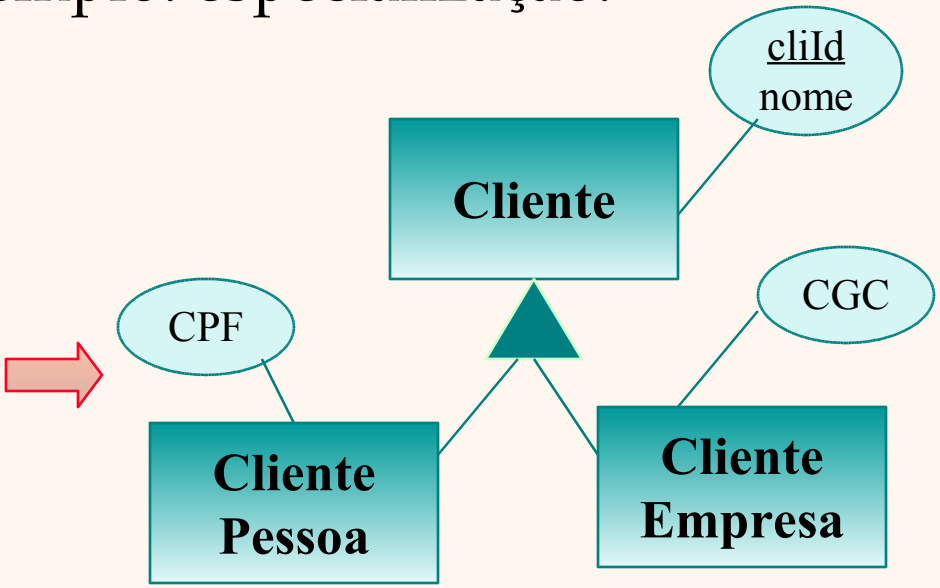
WHERE = age sobre cada tupla individualmente

HAVING = age sobre o grupo

Valores Nulos

- Exemplo: Existem dois tipos de clientes: Pessoa e Empresa. Se o cliente é uma pessoa física ela possui CPF, se for uma Empresa possui CGC. Ambos os campos são opcionais, isto é, aceitam nulos.
- Problema com valores nulos: valores desconhecidos! Considero valores nulos nas minhas pesquisas?
- Uma solução para este exemplo: especialização.

cliId	nome	CPF	CGC
1111	Juliana	21788878904	*Null*
2222	Maria	21455588899	*Null*
3333	Pedro	34567891000	*Null*
4444	Renato	45678567857	*Null*



Outer Join

```
SELECT C.cliId, A.dataPedido FROM Cliente C
NATURAL LEFT OUTER JOIN CorridaAgendada A
```

- Incluir TODAS as linhas de Cliente e somente as linhas da CorridaAgendada quando os campos associados forem iguais (cliId).
- RIGHT OUTER JOIN = ao contrário.
- NATURAL = cria relação pela combinação dos campos com os mesmos nomes (neste exemplo: C.cliId = A.cliId).

cliId	dataPedido
1111	14/01/2004
2222	25/01/2004
3333	10/02/2004
4444	*Null*

cliId	nome	CPF	CGC
1111	Juliana	21788878904	*Null*
2222	Maria	21455588899	*Null*
3333	Pedro	34567891000	*Null*
4444	Renato	45678567857	*Null*

cliId	codLog	num	data Pedido	data Corrida	hora Corrida
1111	11121	123	14/01/2004	15/01/2004	15:00
1111	41231	77	25/01/2004	27/01/2004	18:15
2222	22212	12	10/02/2004	10/02/2004	08:00
3333	33121	65	10/01/2004	27/01/2004	13:00

Restrições de Integridade

- Sobre uma única tabela: pode utilizar chaves primárias, estrangeiras e restrições de domínio.
- Restrições de Domínio
  - No Create Table CorridaAgendada, incluir CHECK (horaCorrida >= '05:00' AND horaCorrida <= '24:00')
- Sobre várias tabelas:
  - Assertions: cria restrições sem depender de uma tabela em particular. Exemplo fictício: um cliente não pode ser um motorista.

```
CREATE ASSERTION ClienteNaoMotorista CHECK
(NOT EXISTS (SELECT C.nome FROM Cliente C
        INTERSECT
        SELECT M.nome FROM Motorista M) )
```

Triggers (Gatilhos)

- Poderoso mecanismo para tratar alterações no Banco de Dados
- Deve ser utilizado com cuidado. Ex: num ponto de execução de uma *trigger*, pode ocorrer de outra *trigger* ser ativada ou até a mesma (*triggers* recursivas).
- Restrições x *Triggers*
  - O uso comum de *triggers* é manter a consistência do Banco de Dados. Em muitos casos, pode-se utilizar uma restrição para atingir o mesmo objetivo. Porém as *triggers* são mais flexíveis.
  - Vão além de manter integridade, podem ser utilizadas para estatísticas, log de eventos para auditoria, entre outros.

Triggers (Gatilhos)

Supondo que na tabela de Cliente exista uma nova coluna chamada idade. Toda vez que um cliente for cadastrado na tabela, queremos ter um controle de Estatística dos clientes com idade >= 60 anos, para oferecer promoções e atendimento diferenciado.

Sintaxe SQL:1999

```
CREATE TRIGGER controleEstatisticas
                    AFTER INSERT ON Cliente      /* Evento */
REFERENCING NEW TABLE TerceiraIdade
FOR EACH STATEMENT
    INSERT                                       /* Ação */
        INTO Estatisticas (nome,idade)
        SELECT nome, idade
        FROM TerceiraIdade T
        WHERE T.idade >= 60
```

Triggers (Gatilhos)

Sintaxe Oracle

```
CREATE TRIGGER init_count BEFORE INSERT ON Cliente      /* Evento */
DECLARE
      count  INTEGER;
BEGIN
      count := 0;                                       /* Ação */
END

CREATE TRIGGER incr_count AFTER INSERT ON Cliente       /* Evento */
      WHEN (new.idade >=60)                             /* Condição*/
      FOR EACH ROW
      BEGIN
                                                        /*Ação */
            count := count + 1;
      END
```

Order by

❖ Classificar atributos na ordem especificada (padrão é ordem crescente).

```
SELECT C.cliId,C.nome, c.CPF, C.CGC
FROM Cliente C
ORDER BY C.nome [ASC/DESC]
```

cliId	nome	CPF	CGC
1111	Juliana	21788878904	*Null*
2222	Maria	21455588899	*Null*
3333	Pedro	34567891000	*Null*
4444	Renato	45678567857	*Null*

Padrão (ASC)

cliId	nome	CPF	CGC
1111	Juliana	21788878904	*Null*
2222	Maria	21455588899	*Null*
3333	Pedro	34567891000	*Null*
4444	Renato	45678567857	*Null*

DESC

cliId	nome	CPF	CGC
4444	Renato	45678567857	*Null*
3333	Pedro	34567891000	*Null*
2222	Maria	21455588899	*Null*
1111	Juliana	21788878904	*Null*

Operação custosa ! Utilizar somente quando necessário.