Bases de Dados

Parte V
A Linguagem SQL

SQL – Structured Query Language

- O SQL é uma poderosa linguagem declarativa que permite definir, questionar e manipular bases de dados.
- Para além das funcionalidades básicas, o SQL incorpora uma série de facilidades adicionais que permitem:
  - Definir restrições de integridade.
  - Definir visões sobre os dados.
  - Especificar transacções.
  - Especificar permissões de segurança e de acesso.
  - Criar índices de forma a optimizar o acesso.
  - Ligar-se a outras linguagens de programação.
  - ...
- O SQL é simultaneamente uma DDL (Data Definition Language) e uma DML (Data Manipulation Language).

SQL – Structured Query Language

- Originalmente, o SQL foi desenhado e desenvolvido pela IBM Research e era chamado de SEQUEL (Structured English QUEry Language).
- Actualmente, o SQL é a linguagem standard para todos os sistemas comerciais de gestão de bases de dados relacionais.
- A 1ª versão standard do SQL foi definida em 1986, o SQL1 ou SQL-86.
- A 2ª versão standard foi definida em 1992, o SQL2 ou SQL-92.
- A versão mais recente é o SQL3 ou SQL-99.

CREATE TABLE

- Permite criar uma nova relação (tabela) indicando o seu nome, atributos (nome, domínio, restrições e valores por defeito) e restrições de integridade (integridade da chave e integridade referencial).

  CREATE TABLE <TABELA> (
  <ATRIB_1> <DOMÍNIO_1> [<OPÇÕES_1>],
  ...,
  <ATRIB_N> <DOMÍNIO_N> [<OPÇÕES_N>],
  [<RESTR_1>, ..., <RESTR_M>]);

- Uma diferença importante entre o SQL e a álgebra e o cálculo relacional é que o SQL permite que uma tabela possua **tuplos repetidos**.

Domínio dos Atributos

- Valores numéricos
  - **TINYINT**: 1 byte
  - **SMALLINT**: 2 bytes
  - **INT**: 4 bytes
  - **BIGINT**: 8 bytes
  - **FLOAT**: 4 bytes
  - **DOUBLE**: 8 bytes
  - **DECIMAL(N, D)**: N dígitos com D dígitos depois do ponto decimal
- Valores temporais
  - **DATE**: formato 'YYYY-MM-DD' ('2004-01-30')
  - **TIME**: formato 'HH:MM:SS' ('09:12:47')
  - **DATETIME**: formato 'YYYY-MM-DD HH:MM:SS' ('2004-01-30 09:12:47')
  - **TIMESTAMP**: formato YYYYMMDDHHMMSS (20040130091247)

Domínio dos Atributos

- Valores lógicos
  - **BOOLEAN**: TRUE e FALSE
- Sequências de texto (*strings*)
  - **CHAR(N)**: *string* de comprimento fixo de N caracteres, $0 \leq N \leq 255$
  - **VARCHAR(N)**: *string* de comprimento variável até N caracteres, $0 \leq N \leq 255$
  - **TEXT**: *string* de comprimento variável até 65 Kbytes
  - **LONGTEXT**: *string* de comprimento variável até 4.3 Gbytes

- Criar a tabela para a relação DEPARTAMENTO.

```
CREATE TABLE DEPARTAMENTO (
    Nome        VARCHAR(50),
    Num         INT,
    GerenteBI   CHAR(9),
    GerenteData DATE );
```

Valores por Defeito e Restrições sobre os Atributos

- Definir o valor por defeito para um atributo.
  <ATRIB> <DOMÍNIO> **DEFAULT** <VAL>
- Não permitir que um atributo possua valores NULL.
  <ATRIB> <DOMÍNIO> **NOT NULL**
- Restringir os valores que um atributo pode tomar.
  <ATRIB> <DOMÍNIO> **CHECK** (<COND>)

- Criar a tabela para a relação DEPARTAMENTO.

```
CREATE TABLE DEPARTAMENTO (
    Nome        VARCHAR(50)  NOT NULL,
    Num         INT          NOT NULL CHECK (Num > 0),
    GerenteBI   CHAR(9)      DEFAULT '000000000',
    GerenteData DATE );
```


Restrições de Integridade da Chave

- Definir a chave primária da tabela.
  PRIMARY KEY (<ATRIB_1>, ..., <ATRIB_N>)
- Definir chaves secundárias da tabela.
  UNIQUE (<ATRIB_1>, ..., <ATRIB_N>)

- Criar a tabela para a relação DEPARTAMENTO.

```
CREATE TABLE DEPARTAMENTO (
    Nome        VARCHAR(50)  NOT NULL,
    Num         INT          NOT NULL CHECK (Num > 0),
    GerenteBI   CHAR(9)      DEFAULT '000000000',
    GerenteData DATE,
    PRIMARY KEY (Num),
    UNIQUE (Nome) );
```

Restrições de Integridade Referencial

- Definir uma chave externa da tabela.
  FOREIGN KEY (<ATRIB_1>, ..., <ATRIB_N>)
  REFERENCES <TABELA>(<CHAVE_1>, ..., <CHAVE_N>)

- Criar a tabela para a relação DEPARTAMENTO.

```
CREATE TABLE DEPARTAMENTO (
    Nome        VARCHAR(50)  NOT NULL,
    Num         INT          NOT NULL CHECK (Num > 0),
    GerenteBI   CHAR(9)      DEFAULT '000000000',
    GerenteData DATE,
    PRIMARY KEY (Num),
    UNIQUE (Nome),
    FOREIGN KEY (GerenteBI) REFERENCES EMPREGADO(NumBI));
```


Manutenção da Integridade Referencial

- As restrições de integridade referencial podem ser violadas quando inserimos ou removemos tuplos ou quando alteramos o valor de uma chave primária ou chave externa. Quando isso acontece, o SQL por defeito rejeita essas operações.
- No entanto, é possível modificar esse comportamento para as operações de remoção (ON DELETE) e alteração (ON UPDATE) que violem a integridade referencial sobre tuplos que são referenciados pela chave externa de outras tabelas:
  - **ON DELETE SET NULL / ON UPDATE SET NULL**: coloca o valor NULL na chave externa dos tuplos que referenciam o tuplo removido/alterado.
  - **ON DELETE SET DEFAULT / ON UPDATE SET DEFAULT**: coloca o valor por defeito na chave externa dos tuplos que referenciam o tuplo removido/alterado.
  - **ON DELETE CASCADE**: remove todos os tuplos que referenciam o tuplo removido.
  - **ON UPDATE CASCADE**: actualiza com o novo valor a chave externa dos tuplos que referenciam o tuplo alterado.

Manutenção da Integridade Referencial

- Criar a tabela para a relação DEPARTAMENTO.

```
CREATE TABLE DEPARTAMENTO (
    ...
    GerenteBI     CHAR(9)          DEFAULT '000000000',
    ...
    FOREIGN KEY (GerenteBI) REFERENCES EMPREGADO(NumBI)
        ON DELETE SET DEFAULT ON UPDATE CASCADE );
```

- Criar a tabela para a relação LOCALIZAÇÕES_DEP.

```
CREATE TABLE LOCALIZAÇÕES_DEP (
    NumDep        INT              NOT NULL,
    Localização   VARCHAR(50)      NOT NULL,
    PRIMARY KEY (NumDep, Localização),
    FOREIGN KEY (NumDep) REFERENCES DEPARTAMENTO(Num)
        ON DELETE CASCADE ON UPDATE CASCADE );
```


Simplificações de Notação

- Se a chave primária for definida por um só atributo.

  <ATRIB> <DOMÍNIO> **PRIMARY KEY**

- Se uma chave externa for definida por um só atributo.

  <ATRIB> <DOMÍNIO> **REFERENCES** <TABELA>(<CHAVE>)

- Criar a tabela para a relação DEPARTAMENTO.

```
CREATE TABLE DEPARTAMENTO (
    Nome          VARCHAR(50),
    Num           INT           PRIMARY KEY,
    GerenteBI     CHAR(9)       REFERENCES EMPREGADO(NumBI),
    GerenteData   DATE );
```

CREATE DOMAIN

- Permite declarar um novo domínio para ser usado na definição de atributos.
  CREATE DOMAIN <NOME> **AS** <DOMÍNIO> [**CHECK** (<COND>)];

- Criar a tabela para a relação DEPARTAMENTO.

```
CREATE DOMAIN CHARNOME AS VARCHAR(50);
CREATE DOMAIN INTPOS AS INT CHECK (INTPOS > 0);
CREATE TABLE DEPARTAMENTO (
    Nome         CHARNOME    NOT NULL,
    Num          INTPOS      NOT NULL,
    GerenteBI    CHAR(9)     DEFAULT '000000000',
    GerenteData  DATE );
```


DROP TABLE

- Permite remover uma tabela (definição + dados).
  DROP TABLE <TABELA> [**CASCADE** | **RESTRICT**];
- Opção **CASCADE**
  - Remove também todas as restrições (chaves externas em outras tabelas) e visões que referenciam a tabela a remover.
- Opção **RESTRICT**
  - Só remove a tabela se esta não for referenciada por nenhuma restrição ou visão.

- Remover a tabela para a relação LOCALIZAÇÕES_DEP.
  DROP TABLE LOCALIZAÇÕES_DEP **CASCADE**;

ALTER TABLE

- Permite alterar os atributos ou restrições de uma tabela.
  ALTER TABLE <TABELA> [**ADD | DROP | ALTER**] <OPÇÕES>;
- Opção **ADD** <ATRIB> <DOMÍNIO>
  - Permite adicionar um novo atributo à tabela. O novo atributo terá valores NULL em todos os tuplos. A restrição NOT NULL não é permitida com esta opção.
- Opção **DROP** <ATRIB> [**CASCADE | RESTRICT**]
  - Permite remover um atributo da tabela. As opções CASCADE e RESTRICT têm o mesmo significado que em DROP TABLE.
- Opção **ALTER** <ATRIB> [**SET | DROP**] <OPÇÕES>
  - Permite alterar as restrições de um atributo da tabela.

ALTER TABLE

- Adicionar um novo atributo à tabela DEPARTAMENTO.
  ALTER TABLE DEPARTAMENTO **ADD** DataCriação DATE;

- Remover um atributo da tabela DEPARTAMENTO.
  ALTER TABLE DEPARTAMENTO **DROP** DataCriação **CASCADE**;

- Remover uma restrição de um atributo da tabela DEPARTAMENTO.
  ALTER TABLE DEPARTAMENTO **ALTER** GerenteBI
  DROP DEFAULT;

- Adicionar uma nova restrição a um atributo da tabela DEPARTAMENTO.
  ALTER TABLE DEPARTAMENTO **ALTER** GerenteBI
  SET DEFAULT '000000000';

SELECT-FROM-WHERE

- Permite consultar a base de dados para obter informação.

  SELECT <ATRIB_1>, ..., <ATRIB_N>

  FROM <TABELA_1>, ..., <TABELA_M>

  [**WHERE** <COND>];
- A operação de SELECT-FROM-WHERE não é a mesma que a operação de selecção da álgebra relacional.
- Uma diferença importante entre o SQL e a álgebra e o cálculo relacional é que o SQL permite que uma tabela possua tuplos repetidos (não é um conjunto).
- Para que uma tabela seja um conjunto é necessário especificar restrições do tipo PRIMARY KEY ou UNIQUE sobre os atributos da tabela ou utilizar a opção DISTINCT nas consultas.
- Em SQL, os operadores de comparação são {=, <, <=, >, >=, <>}.

SELECT-FROM-WHERE

- Obtenha o número do BI, primeiro e último nome de todos os empregados.

  SELECT NumBI, NomeP, NomeF

  FROM EMPREGADO;

  Na álgebra relacional seria:

  $\pi_{NumBI,\ NomeP,\ NomeF}(EMPREGADO)$

  No cálculo relacional por tuplos seria:

  {e.NumBI, e.NomeP, e.NomeF | EMPREGADO(e)}

SELECT-FROM-WHERE

- Obtenha o número do BI dos empregados que trabalham no departamento 4 e cujo salário é superior a 2000 euros.

  SELECT NumBI
  FROM EMPREGADO
  WHERE NumDep = 4 **AND** Salário > 2000;

  Na álgebra relacional seria:

  $\pi_{NumBI}(\sigma_{NumDep = 4 \text{ AND } Salário > 2000}(EMPREGADO))$

  No cálculo relacional por tuplos seria:

  {e.NumBI | EMPREGADO(e) AND e.NumDep = 4 AND e.Salário > 2000}

SELECT-FROM-WHERE

- Obtenha o nome dos empregados que trabalham no departamento de Produção.

  SELECT NomeP, NomeF
  FROM EMPREGADO, DEPARTAMENTO
  WHERE Nome = 'Produção' **AND** NumDep = Num;

  Na álgebra relacional seria:

  EMP_DEP $\leftarrow$ EMPREGADO $\bowtie_{NumDep = Num}$ DEPARTAMENTO

  RESULT $\leftarrow \pi_{NomeP, NomeF}(\sigma_{Nome = \text{'Produção'}}(EMP\_DEP))$

  No cálculo relacional por tuplos seria:

  {e.NomeP, e.NomeF | EMPREGADO(e) AND ($\exists$ d)(DEPARTAMENTO(d)
  AND d.Nome = 'Produção' AND d.Num = e.NumDep)}

SELECT *

- Permite substituir a lista de atributos a seleccionar, significando que se pretende seleccionar todos os atributos sem os mencionar explicitamente.

  SELECT *
  FROM <TABELA_1>, ..., <TABELA_M>
  [**WHERE** <COND>];

- Obtenha os empregados que trabalham no departamento 4 e cujo salário é superior a 2000 euros.

  SELECT *
  FROM EMPREGADO
  WHERE NumDep = 4 AND Salário > 2000;

SELECT DISTINCT

- Permite remover os tuplos em duplicado do resultado da consulta.

  SELECT DISTINCT <ATRIB_1>, ..., <ATRIB_N>
  FROM <TABELA_1>, ..., <TABELA_M>
  [**WHERE** <COND>];

- O SQL não remove automaticamente os tuplos em duplicado porque:
  - É uma operação dispendiosa pois requer ordenar tuplos e remover duplicados.
  - Quando se aplicam funções de agregação não é usual excluir as repetições do resultado.
  - O utilizador pode querer ver os tuplos em duplicado no resultado da consulta.

- Obtenha o número do BI dos empregados que têm dependentes.

  SELECT DISTINCT EmpBI
  FROM DEPENDENTE;

Operadores Aritméticos

- Os operadores aritméticos +, –, * e / podem ser utilizados para fazer cálculos com atributos do tipo numérico.

  <ATRIB> [+ | – | * | /] <VAL>

- Para todos os projectos localizados no Porto, obtenha o nome dos empregados e os respectivos salários se estes fossem aumentados em 5%.

  SELECT NomeP, NomeF, **Salário * 1.05**
  FROM EMPREGADO, TRABALHA_EM, PROJECTO
  WHERE Nome = 'Porto' AND Num = NumProj AND EmpBI = NumBI;

Ambiguidade no Nome dos Atributos

- Se as relações envolvidas numa consulta tiverem atributos com o mesmo nome pode ocorrer uma situação ambígua. Por exemplo, as relações

  DEPARTAMENTO(Nome, Num, GerenteBI, GerenteData)
  PROJECTO(Nome, Num, Localização, NumDep)

  têm atributos com o mesmo nome.

- A solução é **qualificar** os atributos com o prefixo do nome da relação a que pertencem:

  DEPARTAMENTO. Nome
  DEPARTAMENTO. Num
  PROJECTO. Nome
  PROJECTO. Num

Ambiguidade no Nome dos Atributos

- Para todos os projectos localizados no Porto, obtenha o nome do projecto e o último nome do respectivo gerente.

```
SELECT PROJECTO.Nome, NomeF
FROM EMPREGADO, DEPARTAMENTO, PROJECTO
WHERE Localização = 'Porto'
    AND PROJECTO.NumDep = DEPARTAMENTO.Num
    AND GerenteBI = NumBI;
```

Na álgebra relacional seria:

$PROJ\_PORTO \leftarrow \sigma_{PLocal = \text{'Porto'}}(\rho_{(PNome, PNum, PLocal, PDep)}(PROJECTO))$

$PORTO\_DEP \leftarrow PROJ\_PORTO \bowtie_{PDep = Num} DEPARTAMENTO$

$RESULT \leftarrow \pi_{PNome, NomeF}(PORTO\_DEP \bowtie_{GerenteBI = NumBI} EMPREGADO)$

Ambiguidade no Nome das Relações

- Se uma consulta referenciar uma mesma relação mais do que uma vez pode ocorrer uma situação ambígua. Por exemplo, o atributo SuperBI da relação EMPREGADO referencia a própria relação.
- A solução é renomear cada uma das duas relações:

```
EMPREGADO AS E
EMPREGADO AS S
```

- Obter para cada empregado, o seu último nome e o último nome do respectivo supervisor.

```
SELECT E.NomeF, S.NomeF
FROM EMPREGADO AS E, EMPREGADO AS S
WHERE E.SuperBI = S.NumBI;
```

Ambiguidade no Nome das Relações

- Uma relação pode ser renomeada mesmo que não seja referenciada mais do que uma vez na consulta. Ao renomear uma relação também é possível renomear o nome dos seus atributos.

  PROJECTO AS P(PNome, PNum, PLocal, PDep)

- Para todos os projectos localizados no Porto, obtenha o nome do projecto e o último nome do respectivo gerente.

  SELECT **PNome**, NomeF
  FROM EMPREGADO, DEPARTAMENTO,
  PROJECTO AS P(PNome, PNum, PLocal, PDep)
  WHERE **PLocal** = 'Porto' AND **PDep** = Num AND GerenteBI = NumBI;

Renomear Atributos

- O SQL permite que os atributos que aparecem no resultado de uma consulta sejam renomeados de modo a fazerem mais sentido.

  <ATRIB> **AS** <NOVO_NOME>

- Para todos os projectos localizados no Porto, obtenha o nome dos empregados e os respectivos salários se estes fossem aumentados em 5%.

  SELECT NomeP, NomeF, **Salário * 1.05 AS Aumento_Salário**
  FROM EMPREGADO, TRABALHA_EM, PROJECTO
  WHERE Nome = 'Porto' AND Num = NumProj AND EmpBI = NumBI;

- Obter para cada empregado, o seu último nome e o último nome do respectivo supervisor.

  SELECT **E.NomeF AS Nome_Empregado**, **S.NomeF AS Nome_Supervisor**
  FROM **EMPREGADO AS E**, **EMPREGADO AS S**
  WHERE E.SuperBI = S.NumBI;

BETWEEN

- Permite definir um intervalo numérico de comparação.
  <ATRIB> **BETWEEN** <VAL_1> **AND** <VAL_2>

- Obtenha os empregados que trabalham no departamento 4 e cujo salário é superior a 2000 euros e inferior a 4000 euros.

```
SELECT *
FROM EMPREGADO
WHERE NumDep = 4 AND (Salário BETWEEN 2000 AND 4000);
```


LIKE

- Permite comparar atributos do tipo string com sequências de texto padrão.
  <ATRIB> [**NOT**] **LIKE** <PADRÃO>
- Existem dois caracteres com significado especial:
  - **%** representa um número arbitrário de caracteres.
  - _ representa um qualquer caracter.

- Obtenha os empregados que vivem no Porto.

```
SELECT *
FROM EMPREGADO
WHERE Endereço LIKE '%Porto';
```

IS NULL

- Permite verificar se os valores de um atributo são NULL (não conhecido, em falta ou não aplicável).

  <ATRIB> **IS** [**NOT**] **NULL**

- Obtenha o nome dos empregados que não têm supervisor.

  SELECT NomeP, NomeF
  FROM EMPREGADO
  WHERE SuperBI **IS NULL**;

ORDER BY

- Permite definir a ordem dos tuplos do resultado.

  ORDER BY <ATRIB_1> [**ASC** | **DESC**], ..., <ATRIB_N> [**ASC** | **DESC**]

- Obtenha por ordem alfabética o nome dos empregados que trabalham no departamento 4 e cujo salário é superior a 2000 euros.

  SELECT NomeP, NomeF
  FROM EMPREGADO
  WHERE NumDep = 4 AND Salário > 2000
  ORDER BY NomeP, NomeF;

LIMIT

- Permite limitar o número de tuplos do resultado.

  LIMIT <VAL>
- É utilizado normalmente com o ORDER BY como forma de seleccionar os primeiros <VAL> valores que verificam um determinado critério de ordenação.

- Obtenha o valor do salário máximo dos empregados que trabalham no departamento 4.

```
SELECT Salário
FROM EMPREGADO
WHERE NumDep = 4
ORDER BY Salário DESC LIMIT 1;
```


Operações sobre Conjuntos

- O SQL incorpora algumas das operações sobre conjuntos da álgebra relacional em que o resultado é um conjunto de tuplos, ou seja, os tuplos em duplicado são removidos:
  - Reunião: (<CONSULTA_1>) **UNION** (<CONSULTA_2>)
  - Intersecção: (<CONSULTA_1>) **INTERSECT** (<CONSULTA_2>)
  - Diferença: (<CONSULTA_1>) **EXCEPT** (<CONSULTA_2>)
- Existem também versões destes operadores que não removem os tuplos em duplicado:
  - Reunião: (<CONSULTA_1>) **UNION ALL** (<CONSULTA_2>)
  - Intersecção: (<CONSULTA_1>) **INTERSECT ALL** (<CONSULTA_2>)
  - Diferença: (<CONSULTA_1>) **EXCEPT ALL** (<CONSULTA_2>)
- Estas operações só se aplicam a relações **compatíveis para a reunião**, ou seja, ambas as relações devem ter os mesmos atributos e na mesma ordem.

Operações sobre Conjuntos

- Obtenha o número do BI dos empregados que trabalham no departamento 4 ou que supervisionam um empregado que trabalha no departamento 4.

```
(SELECT NumBI
FROM EMPREGADO
WHERE NumDep = 4)
UNION
(SELECT SuperBI
FROM EMPREGADO
WHERE NumDep = 4);
```


Consultas Encadeadas

- O SQL permite que uma consulta, chamada de **consulta encadeada**, seja utilizada como parte da condição WHERE de outra consulta, chamada de **consulta externa**, sendo possível ter vários níveis de consultas encadeadas.

  [<ATRIB> | <VAL>] [**NOT**] **IN** <CONSULTA>

  [<ATRIB> | <VAL>] [= | < | <= | > | >= | <>] <CONSULTA>

  [<ATRIB> | <VAL>] [**NOT**] [= | < | <= | > | >= | <>]**ALL** <CONSULTA>

  [<ATRIB> | <VAL>] [**NOT**] [= | < | <= | > | >= | <>]**ANY** <CONSULTA>
- (A **IN** CONSULTA) é verdadeiro se A $\in$ CONSULTA. Caso contrário é falso.
- (A **θ** CONSULTA) é verdadeiro se CONSULTA tiver um único tuplo com um só atributo de valor C tal que A θ C. Caso contrário é falso.
- (A **θALL** CONSULTA) é verdadeiro se A θ C para todo o C $\in$ CONSULTA. Caso contrário é falso.
- (A **θANY** CONSULTA) é verdadeiro se A θ C para algum C $\in$ CONSULTA. Caso contrário é falso.

Consultas Encadeadas

- Obtenha o nome dos empregados que trabalham no departamento de Produção.

```
SELECT NomeP, NomeF
FROM EMPREGADO
WHERE NumDep = (SELECT Num
                FROM DEPARTAMENTO
                WHERE Nome = 'Produção');
```

- Obtenha o nome dos empregados cujo salário é superior ao salário de todos os empregados do departamento 4.

```
SELECT NomeP, NomeF
FROM EMPREGADO
WHERE Salário >ALL (SELECT Salário
                    FROM EMPREGADO
                    WHERE NumDep = 4);
```


Consultas Encadeadas

- Obtenha o nome dos projectos nos quais o empregado Rui Silva (NumBI= '487563546') trabalha ou é o gerente.

```
SELECT Nome
FROM PROJECTO
WHERE Num IN ((SELECT NumProj
               FROM TRABALHA_EM
               WHERE EmpBI = '487563546')
               UNION
               (SELECT Num
               FROM PROJECTO
               WHERE NumDep IN (SELECT Num
                                FROM DEPARTAMENTO
                                WHERE GerenteBI = '487563546')));
```

Consultas Encadeadas

- Obtenha o número do BI dos empregados que trabalham as mesmas horas que o empregado Rui Silva (NumBI='487563546') trabalha num determinado projecto.

```
SELECT DISTINCT EmpBI
FROM TRABALHA_EM
WHERE (NumProj, Horas) IN (SELECT NumProj, Horas
                           FROM TRABALHA_EM
                           WHERE EmpBI = '487563546');
```

- Obtenha o número do BI dos empregados que trabalham nos projectos 4, 5 ou 6.

```
SELECT DISTINCT EmpBI
FROM TRABALHA_EM
WHERE NumProj IN (4, 5, 6);
```


Consultas Encadeadas e Correlacionadas

- Se uma consulta encadeada tiver atributos não qualificados com o mesmo nome dos atributos da consulta externa pode ocorrer uma situação ambígua.
- Por regra os atributos não qualificados de uma consulta encadeada referem-se sempre às tabelas da consulta encadeada (a ideia é semelhante ao âmbito das variáveis em linguagens de programação).
- Quando uma consulta encadeada referencia atributos (qualificados) da consulta externa, as consultas dizem-se **correlacionadas**. Nestes casos, a consulta encadeada é avaliada repetidamente contra cada tuplo da consulta externa.

- Obtenha o nome dos empregados cujo supervisor e gerente são o mesmo.

```
SELECT E.NomeP, E.NomeF
FROM EMPREGADO AS E
WHERE E.NumDep IN (SELECT Num
                   FROM DEPARTAMENTO AS D
                   WHERE E.SuperBI = D.GerenteBI);
```

EXISTS

- Permite verificar se o resultado de uma consulta encadeada e correlacionada é vazio (não possui nenhum tuplo) ou não. A consulta encadeada é avaliada repetidamente contra cada tuplo da consulta externa.

  [**NOT**] **EXISTS** <CONSULTA>

- (**EXISTS** CONSULTA) é verdadeiro se CONSULTA não for vazio. Caso contrário é falso.

- Obtenha o nome dos empregados que têm um dependente do mesmo sexo.

```
SELECT E.NomeP, E.NomeF
FROM EMPREGADO AS E
WHERE EXISTS (SELECT *
              FROM DEPENDENTE AS D
              WHERE E.NumBI = D.EmpBI AND E.Sexo = D.Sexo);
```


EXISTS

- Obtenha o nome dos empregados que não têm dependentes.

```
SELECT NomeP, NomeF
FROM EMPREGADO AS E
WHERE NOT EXISTS (SELECT *
                  FROM DEPENDENTE
                  WHERE E.NumBI = EmpBI);
```

EXISTS

- Obtenha o nome dos empregados que trabalham em TODOS os projectos controlados pelo departamento 4. Ou dito de outro modo, obtenha o nome dos empregados para os quais NÃO EXISTE um projecto controlado pelo departamento 4 para o qual eles NÃO trabalhem.

```
SELECT NomeP, NomeF
FROM EMPREGADO AS E
WHERE NOT EXISTS (SELECT *
                  FROM PROJECTO AS P
                  WHERE P.NumDep = 4
                        AND NOT EXISTS (SELECT *
                                        FROM TRABALHA_EM AS T
                                        WHERE E.NumBI = T.EmpBI
                                        AND P.Num = T.NumProj));
```


EXISTS

- Obtenha o nome dos empregados para os quais NÃO EXISTE um projecto controlado pelo departamento 4 para o qual eles NÃO trabalhem (resolução alternativa utilizando o operador EXCEPT).

```
SELECT NomeP, NomeF
FROM EMPREGADO AS E
WHERE NOT EXISTS ((SELECT Num
                   FROM PROJECTO
                   WHERE NumDep = 4)
                   EXCEPT
                   (SELECT NumProj
                   FROM TRABALHA_EM
                   WHERE E.NumBI = EmpBI));
```

Operações de Junção

- O SQL permite que a junção de tabelas também seja feita de forma explícita.

  <TABELA_1> [**INNER**] **JOIN** <TABELA_2> **ON** <COND>
  <TABELA_1> **LEFT** [**OUTER**] **JOIN** <TABELA_2> **ON** <COND>
  <TABELA_1> **RIGHT** [**OUTER**] **JOIN** <TABELA_2> **ON** <COND>
  <TABELA_1> **FULL** [**OUTER**] **JOIN** <TABELA_2> **ON** <COND>
  <TABELA_1> **NATURAL JOIN** <TABELA_2>
  <TABELA_1> **NATURAL LEFT OUTER JOIN** <TABELA_2>
  <TABELA_1> **NATURAL RIGHT OUTER JOIN** <TABELA_2>
  <TABELA_1> **NATURAL FULL OUTER JOIN** <TABELA_2>

Operações de Junção

- Obtenha o nome dos empregados que trabalham no departamento de Produção.

```
SELECT NomeP, NomeF
FROM (EMPREGADO JOIN DEPARTAMENTO ON NumDep = Num)
WHERE Nome = 'Produção';
```

ou em alternativa:

```
SELECT NomeP, NomeF
FROM (EMPREGADO NATURAL JOIN
    (DEPARTAMENTO AS DEP(Nome, NumDep, GBi, GData)))
WHERE Nome = 'Produção';
```

Operações de Junção

- Para todos os projectos localizados no Porto, obtenha o nome do projecto e o último nome do respectivo gerente.

```
SELECT PNome, NomeF
FROM ((EMPREGADO JOIN DEPARTAMENTO ON NumBI = GerenteBI)
    NATURAL JOIN (PROJECTO AS P(PNome, PNum, PLocal, Num)))
WHERE PLocal = 'Porto';
```

- Obtenha o número do BI dos empregados que não têm dependentes.

```
SELECT NumBI
FROM (EMPREGADO LEFT OUTER JOIN DEPENDENTE
    ON NumBI = EmpBI)
WHERE EmpBI IS NULL;
```


Funções de Agregação

- O SQL permite sumariar informação por utilização de funções de agregação.
  - **COUNT**(<ATRIB>)
  - **SUM**(<ATRIB>)
  - **MAX**(<ATRIB>)
  - **MIN**(<ATRIB>)
  - **AVG**(<ATRIB>)
- Os valores NULL existentes nos atributos a sumariar **não são considerados** pelas funções de agregação.
- Obtenha o valor do salário máximo, do salário mínimo e da soma do salário de todos os empregados.

```
SELECT MAX(Salário), MIN(Salário), SUM(Salário)
FROM EMPREGADO;
```

Funções de Agregação

- Obtenha o número de empregados que trabalham no departamento de Produção e a respectiva média salarial.

```
SELECT COUNT(*) AS NumEmp, AVG(Salário) AS MedSal
FROM (EMPREGADO JOIN DEPARTAMENTO ON NumDep = Num)
WHERE Nome = 'Produção';
```

- Obtenha o número de valores diferentes de salário.

```
SELECT COUNT(DISTINCT Salário)
FROM EMPREGADO;
```

- Obtenha o nome dos empregados com dois ou mais dependentes.

```
SELECT NomeP, NomeF
FROM EMPREGADO AS E
WHERE (SELECT COUNT(*)
        FROM DEPENDENTE
        WHERE E.NumBI = EmpBI) >= 2;
```


GROUP BY

- Permite agrupar tuplos em função de um conjunto de atributos. Em cada grupo, todos os tuplos têm o mesmo valor para os atributos de agrupamento.

  GROUP BY <ATRIB_1>, ..., <ATRIB_N>

- É utilizado em conjunto com as funções de agregação. As funções de agregação são aplicadas separadamente e individualmente sobre cada grupo.
- Se existirem valores NULL nos atributos de agrupamento **é criado um grupo separado** para esse conjunto de tuplos.
- Os atributos do corpo da cláusula SELECT **devem estar em funções de agregação ou devem fazer parte dos atributos de agrupamento**.

- Obtenha o número de empregados por departamento e a respectiva média salarial.

```
SELECT NumDep, COUNT(*) AS NumEmp, AVG(Salário) AS MedSal
FROM EMPREGADO
GROUP BY NumDep;
```

GROUP BY–HAVING

- Permite especificar uma condição de selecção sobre os grupos.
  GROUP BY <ATRIB_1>, ..., <ATRIB_N>
  HAVING <GROUP_COND>
- Equivale à cláusula WHERE mas no contexto do GROUP BY.
- Os atributos do corpo da cláusula HAVING **devem estar em funções de agregação ou devem fazer parte dos atributos de agrupamento**.

- Para os projectos com mais do que dois empregados, obtenha o número do projecto, o nome do projecto e o número de empregados que nele trabalham.

```
SELECT Num, Nome, COUNT(*)
FROM PROJECTO, TRABALHA_EM
WHERE Num = NumProj
GROUP BY Num, Nome
HAVING COUNT(*) > 2;
```


GROUP BY–HAVING

- Para os departamentos com mais do que quatro empregados, obtenha o nome do departamento e o número de empregados cujo salário é superior a 2000 euros.

```
SELECT Nome, COUNT(*) AS NumEmp
FROM EMPREGADO, DEPARTAMENTO
WHERE NumDep = Num AND Salário > 2000
GROUP BY Nome
HAVING NumEmp > 4;
```

- A consulta acima **está errada** porque **não calcula** os departamentos que têm mais do que quatro empregados, mas sim os departamentos que têm mais do que quatro empregados cujo salário é superior a 2000 euros. **Este erro deve-se ao facto da cláusula WHERE ser executada primeiro do que a cláusula HAVING**.

GROUP BY–HAVING

- Para os departamentos com mais do que quatro empregados, obtenha o nome do departamento e o número de empregados cujo salário é superior a 2000 euros.

```
SELECT Nome, COUNT(*) AS NumEmp
FROM EMPREGADO, DEPARTAMENTO
WHERE NumDep = Num AND Salário > 2000
    AND NumDep IN (SELECT NumDep
                   FROM EMPREGADO
                   GROUP BY NumDep
                   HAVING COUNT(*) > 4)
GROUP BY Nome;
```

- Note que a consulta acima **não calcula** os departamentos com mais do que quatro empregados em que nenhum empregado tem um salário superior a 2000 euros.

Resumo das Consultas SQL

- A sintaxe de uma consulta SQL pode resumir-se ao seguinte:

```
SELECT [DISTINCT] <ATRIB_S1>, ..., <ATRIB_SN>
FROM <TABELA_1>, ..., <TABELA_M>
[WHERE <COND>]
[GROUP BY <ATRIB_G1>, ..., <ATRIB_GN>
[HAVING <GROUP_COND>]]
[ORDER BY <ATRIB_O1> [ASC | DESC], ..., <ATRIB_ON> [ASC | DESC]]
```

- Conceptualmente, uma consulta SQL é avaliada da seguinte forma:
  - Identificar as tabelas envolvidas e/ou aplicar as operações de junção definidas em FROM;
  - Seleccionar os tuplos que verificam a condição WHERE;
  - Agrupar tuplos em função dos atributos de agrupamento definidos em GROUP BY;
  - Seleccionar os grupos que verificam a condição HAVING;
  - Ordenar o resultado em função dos atributos definidos em ORDER BY.
  - Apresentar os valores do resultado para os atributos definidos em SELECT;

INSERT

- Permite adicionar tuplos a uma tabela.

  INSERT INTO <TABELA>[(<ATRIB_1>, ..., <ATRIB_N>)]

  [**VALUES** (<VAL_A1>, ..., <VAL_AN>), ..., (<VAL_M1>, ..., <VAL_MN>)

  | <CONSULTA>];
- Quando não se indica os atributos da tabela assume-se que são todos e pela ordem definida quando da sua criação. Os valores são inseridos por essa mesma ordem.
- Quando se explicita o nome dos atributos é possível definir a ordem e indicar apenas parte dos atributos. Os valores são inseridos na ordem definida.
- Os atributos com declarações NOT NULL e sem declarações DEFAULT devem ser sempre indicados.
- Nos atributos não indicados é inserido o valor por defeito (se definido) ou o valor NULL.

INSERT

- Adicionar um novo tuplo a uma tabela.

  INSERT INTO DEPARTAMENTO

  VALUES (‘Investigação’, 10, ‘487563546’, ‘2000-01-30’);
- Adicionar dois novos tuplos a uma tabela indicando apenas parte dos atributos.

  INSERT INTO DEPARTAMENTO(Num, Nome)

  VALUES (11, ‘Publicidade’), (12, ‘Recursos Humanos’);
- Adicionar o resultado de uma consulta a uma tabela.

  CREATE TABLE DEPARTAMENTO_RESUMO (

  Nome AS VARCHAR(50), TotalEmp AS INT, TotalSal AS INT);

  INSERT INTO DEPARTAMENTO_RESUMO

  SELECT Nome, COUNT(*), SUM(Salário)

  FROM (EMPREGADO JOIN DEPARTAMENTO ON NumDep = Num)

  GROUP BY Nome;

DELETE

- Permite remover tuplos de uma tabela.

  DELETE FROM <TABELA>

  [**WHERE** <COND>];
- Quando não se indica a condição WHERE todos os tuplos são removidos mas a tabela mantêm-se na BD embora vazia.
- Esta operação apenas remove directamente tuplos de uma tabela. No entanto, esta operação pode propagar-se a outras tabelas para manter as restrições de integridade referencial.
  - **ON DELETE SET NULL**: coloca o valor NULL na chave externa dos tuplos que referenciam os tuplos removidos.
  - **ON DELETE SET DEFAULT**: coloca o valor por defeito na chave externa dos tuplos que referenciam os tuplos removidos.
  - **ON DELETE CASCADE**: remove todos os tuplos que referenciam os tuplos removidos.

DELETE

- Remover zero ou um tuplo de uma tabela.

  DELETE FROM EMPREGADO

  WHERE NumBI = '487563546';
- Remover zero, um ou vários tuplos de uma tabela.

  DELETE FROM EMPREGADO

  WHERE Salário > 2000;
- Remover zero, um ou vários tuplos de uma tabela.

  DELETE FROM EMPREGADO

  WHERE NumDep IN (SELECT Num

  FROM DEPARTAMENTO

  WHERE Nome = 'Produção');
- Remover todos os tuplos de uma tabela.

  DELETE FROM EMPREGADO;

UPDATE

- Permite alterar os valores dos atributos de um ou mais tuplos de uma tabela.

  UPDATE <TABELA>
  SET <ATRIB_1> = <VAL_1>, ..., <ATRIB_N> = <VAL_N>
  WHERE <COND>;

- Esta operação apenas altera directamente tuplos de uma tabela. No entanto, esta operação pode propagar-se a outras tabelas para manter as restrições de integridade referencial.
  - **ON UPDATE SET NULL**: coloca o valor NULL na chave externa dos tuplos que referenciam os tuplos alterados.
  - **ON UPDATE SET DEFAULT**: coloca o valor por defeito na chave externa dos tuplos que referenciam os tuplos alterados.
  - **ON UPDATE CASCADE**: actualiza com os novos valores a chave externa dos tuplos que referenciam os tuplos alterados.

UPDATE

- Alterar um tuplo de uma tabela.

  UPDATE PROJECTO
  SET Localização = 'Gaia', NumDep = 4
  WHERE Num = 1;

- Alterar vários tuplos de uma tabela.

  UPDATE EMPREGADO
  SET Salário = Salário * 1.05
  WHERE NumDep IN (SELECT Num
  FROM DEPARTAMENTO
  WHERE Nome = 'Produção');

CREATE VIEW

- Permite declarar uma visão diferente da informação existente na base de dados.

  CREATE VIEW <NOME>[(<ATRIB_1>, ..., <ATRIB_N>)]

  AS <CONSULTA>;
- Uma visão é uma espécie de tabela virtual derivada a partir de outras tabelas ou visões.
- Uma visão é apenas uma definição de como sumariar ou agrupar informação. Os tuplos de uma visão não existem fisicamente na BD associados à visão, eles ou residem nas tabelas a partir das quais a visão é derivada ou são calculados quando a visão é utilizada.
- As visões são especialmente úteis para referenciar informação que é manipulada frequentemente ou para fornecer diferentes perspectivas dos dados a diferentes utilizadores.

CREATE VIEW

- Declarar uma visão que contenha para cada empregado o seu nome e o nome do departamento para o qual trabalha.

  CREATE VIEW EMPREGADO_DE

  AS SELECT NomeP, NomeF, Nome

  FROM (EMPREGADO JOIN DEPARTAMENTO ON NumDep = Num);
- Declarar uma visão que contenha para cada departamento o seu nome, número de empregados e o custo envolvido com salários nesse departamento.

  CREATE VIEW DEPARTAMENTO_RESUMO(Nome, TotalEmp, TotalSal)

  AS SELECT Nome, COUNT(*), SUM(Salário)

  FROM (EMPREGADO JOIN DEPARTAMENTO ON NumDep = Num)

  GROUP BY Nome;

CREATE VIEW

- Qual a diferença entre ter uma visão e ter uma tabela?

```
CREATE TABLE DEPARTAMENTO_RESUMO (
    Nome AS VARCHAR(50), TotalEmp AS INT, TotalSal AS INT);
INSERT INTO DEPARTAMENTO_RESUMO
    SELECT Nome, COUNT(*), SUM(Salário)
    FROM (EMPREGADO JOIN DEPARTAMENTO ON NumDep = Num)
    GROUP BY Nome;
```

- Uma visão **não é calculada no momento da sua definição**, mas sim sempre que é utilizada numa consulta.
- Uma visão está **sempre actualizada**. Se alterarmos os tuplos das tabelas a partir das quais a visão foi derivada, a visão reflecte automaticamente essas alterações.

DROP VIEW

- Permite remover uma visão quando esta não é mais necessária.

```
DROP VIEW <VISÃO>;
```

- Remover as visões definidas anteriormente.

```
DROP VIEW EMPREGADO_DE;
DROP VIEW DEPARTAMENTO_RESUMO;
```

Consultas Utilizando Visões

- Existem duas estratégias principais que os SGBDs usam para implementar consultas que utilizam visões:
  - **Transformação da visão**: transformar a consulta numa consulta equivalente que utilize apenas as tabelas a partir das quais a visão é derivada. É ineficiente para visões mais complexas especialmente se várias consultas utilizarem a visão num curto espaço de tempo.
  - **Materialização da visão**: criar e manter fisicamente uma tabela temporária que represente a visão. Parte do pressuposto que várias consultas utilizarão a visão num curto espaço de tempo. Requer uma estratégia eficiente para manter de modo automático a tabela–visão actualizada quando as tabelas a partir das quais a visão é derivada são alteradas (**actualização incremental**).

- Obtenha o nome dos empregados que trabalham no departamento de Produção.

```
SELECT NomeP, NomeF
FROM EMPREGADO_DE
WHERE Nome = 'Produção';
```


Alterações Utilizando Visões

- Para implementar alterações que utilizam visões, o SGBD necessita de transformar a operação de alteração numa operação equivalente que utilize apenas as tabelas a partir das quais a visão é derivada.
- No entanto, se a alteração for sobre os tuplos da própria visão isso pode não ser possível ou ser ambíguo.
  - Uma visão com funções de agregação ou grupos não pode ser alterada.
  - Uma visão derivada a partir da junção de tabelas normalmente não pode ser alterada (leva a situações de ambiguidade).
  - Uma visão derivada a partir de uma tabela só pode ser alterada se a visão incluir a chave primária e todos os atributos NOT NULL e sem valores por defeito da tabela.

- Alterar um tuplo de uma visão.

```
UPDATE EMPREGADO_DE
SET Nome = 'Vendas'
WHERE Nome = 'Produção';
```

Alterações Utilizando Visões

- A operação anterior pode ser transformada em operações equivalentes do ponto de vista da visão, mas com resultados diferentes sobre o ponto de vista das tabelas a partir das quais a visão é derivada.
- Uma transformação possível é:

```
UPDATE DEPARTAMENTO
SET Nome = 'Vendas'
WHERE Nome = 'Produção';
```

- Uma transformação alternativa é:

```
UPDATE EMPREGADO
SET NumDep = (SELECT Num FROM DEPARTAMENTO
              WHERE Nome = 'Vendas')
WHERE NumDep = (SELECT Num FROM DEPARTAMENTO
                WHERE Nome = 'Produção')
```


Elementos Activos em SQL

- São expressões ou comandos da DDL (Data Definition Language) guardados na BD que são executados quando determinados eventos ocorrem.
- O SQL incorpora diferentes tipos de elementos activos:
  - **Integridade de domínio**: restrição que garante que os valores de um atributo devem pertencer ao domínio do atributo.
  - **Integridade da chave**: restrição que garante que não podem existir dois tuplos de uma relação com valores iguais na chave primária.
  - **Integridade de entidade**: restrição que garante que os valores da chave primária não podem ser nulos.
  - **Integridade referencial**: restrição que garante que um tuplo que referencia outra relação tem de referenciar um tuplo existente nessa outra relação.
  - **Asserção**: restrição genérica sobre qualquer conjunto de tuplos que garante que uma determinada condição nunca é violada.
  - **Trigger (gatilho)**: restrição genérica sobre certos eventos que garante a execução de uma determinada acção.

CREATE ASSERTION

- Permite declarar restrições mais genéricas sobre qualquer conjunto de tuplos.

  CREATE ASSERTION <NOME>

  CKECK (<COND>);
- Qualquer operação sobre a BD que viole a asserção (leve COND a tomar o valor falso) é rejeitada.
- A ideia é especificar em COND uma consulta que seleccione os tuplos que violam a condição pretendida e inclui-la dentro de uma cláusula NOT EXISTS. Quando o resultado da consulta é não vazio isso significa que a asserção foi violada.

CREATE ASSERTION

- O salário de um empregado não pode ser superior ao salário do gerente do departamento para o qual o empregado trabalha.

```
CREATE ASSERTION RESTR_SALÁRIO_GERENTE
CHECK (NOT EXISTS (SELECT *
                   FROM EMPREGADO AS E,
                      EMPREGADO AS S, DEPARTAMENTO
                   WHERE E.NumDep = Num
                      AND GerenteBI = S.NumBI
                      AND E.Salário > S.Salário));
```

CREATE TRIGGER

- Permite declarar acções de reacção para determinados eventos (a notação que se segue é do SGBD Oracle que é muito semelhante à do standard SQL-99).

  CREATE TRIGGER <NOME>
  [**AFTER** | **BEFORE**] <EVENTOS> **ON** <TABELA>
  [**FOR EACH ROW**]
  [**WHEN** (<COND>)]
  <ACÇÃO>;

- Um trigger é como que uma regra do tipo **evento–condição–acção** (**ECA**).
  - O evento faz disparar a regra. Normalmente são operações **UPDATE**, **INSERT** ou **DELETE** sobre uma tabela (**ON** <TABELA>) ou sobre os atributos de uma tabela (**OF** <ATRIB_1>, ..., <ATRIB_N> **ON** <TABELA>).
  - A condição é uma restrição que determina se a acção deve ser executada ou não.
  - A acção é uma sequências de comandos SQL ou um procedimento escrito na linguagem de programação do SGBD. A acção pode ser executada depois (**AFTER**) ou antes (**BEFORE**) do evento que a dispara.

CREATE TRIGGER

- Suponha que a tabela DEPARTAMENTO possui um atributo derivado de nome TotalSal que representa o total dos salários dos empregados que trabalham no departamento respectivo.
- Os eventos que podem determinar uma alteração do valor desse atributo são:
  - Inserir um novo empregado.
  - Remover um empregado.
  - Alterar o salário de um empregado.
  - Mudar um empregado de um departamento para outro.

CREATE TRIGGER

- Inserir um novo empregado.

```
CREATE TRIGGER TOTALSAL1
AFTER INSERT ON EMPREGADO
FOR EACH ROW
WHEN (NEW.NumDep IS NOT NULL)
    UPDATE DEPARTAMENTO
    SET TotalSal = TotalSal + NEW.Salário
    WHERE Num = NEW.NumDep;
```


CREATE TRIGGER

- Remover um empregado.

```
CREATE TRIGGER TOTALSAL2
AFTER DELETE ON EMPREGADO
FOR EACH ROW
WHEN (OLD.NumDep IS NOT NULL)
    UPDATE DEPARTAMENTO
    SET TotalSal = TotalSal – OLD.Salário
    WHERE Num = OLD.NumDep;
```

CREATE TRIGGER

- Alterar o salário de um empregado.

```
CREATE TRIGGER TOTALSAL3
AFTER UPDATE OF Salário ON EMPREGADO
FOR EACH ROW
WHEN (NEW.NumDep IS NOT NULL)
    UPDATE DEPARTAMENTO
    SET TotalSal = TotalSal + NEW.Salário – OLD.Salário
    WHERE Num = NEW.NumDep;
```


CREATE TRIGGER

- Mudar um empregado de um departamento para outro.

```
CREATE TRIGGER TOTALSAL4
AFTER UPDATE OF NumDep ON EMPREGADO
FOR EACH ROW
    BEGIN
    UPDATE DEPARTAMENTO
    SET TotalSal = TotalSal + NEW.Salário
    WHERE Num = NEW.NumDep;
    UPDATE DEPARTAMENTO
    SET TotalSal = TotalSal – OLD.Salário
    WHERE Num = OLD.NumDep;
    END;
```

CREATE TRIGGER

- Suponha que pretendemos monitorizar as situações em que o salário de um empregado é superior ao do seu supervisor.

```
CREATE TRIGGER SALÁRIO_SUPERVISOR
BEFORE INSERT OR UPDATE OF Salário, SuperBI ON EMPREGADO
FOR EACH ROW
WHEN (NEW.Salário > (SELECT Salário
                     FROM EMPREGADO
                     WHERE NumBI = NEW.SuperBI))
  PROC_INFORMAR_SUPERVISOR (NEW.SuperBI, NEW.NumBI);
```