Exemplos de triggers (cont)

2. Alteração do salário de um ou mais empregados
3. Mudança de empregados de um departamento para outro.

```
R2: CREATE TRIGGER TotalSal2
    AFTER UPDATE OF Salario ON Empregado
    FOR EACH ROW
    WHEN (NEW.Ndep IS NOT NULL)
      UPDATE Departamento
      SET TotalSal=TotalSal+NEW.Salario-OLD.Salario
      WHERE Dnum=NEW.Ndep;
R3: CREATE TRIGGER TotalSal3
    AFTER UPDATE OF Ndep ON Empregado
    FOR EACH ROW
    BEGIN
      UPDATE Departamento
      SET TotalSal=TotalSal+NEW.Salario
      WHERE Dnum=NEW.Ndep;
      UPDATE Departamento
      SET TotalSal=TotalSal-OLD.Salario
      WHERE Dnum=OLD.Ndep;
    END;
```


Exemplos de triggers (cont)

4. Remover um ou mais empregados.

```
R4: CREATE TRIGGER TotalSal4
    AFTER DELETE ON Empregado
    FOR EACH ROW
    WHEN (OLD.Ndep IS NOT NULL)
      UPDATE Departamento
      SET TotalSal=TotalSal-OLD.Salario
      WHERE Dnum=OLD.Ndep;
```

Triggers (cont.)

- A execução da acção pode ser condicionada:
  – `AFTER` – a condição `WHEN` é testada depois do evento-de-disparo.
  – `BEFORE` – a condição `WHEN` é testada antes do evento-de-disparo.
  – `INSTEAD OF` – a acção é executada se a condição `WHEN` se verifica e o evento-de-disparo não é executado.
- Eventos-de-disparo: `UPDATE, INSERT, DELETE`.
- `FOR EACH ROW` – especifica que a regra vai ser disparada uma vez por cada linha (tuplo) afectada pelo evento-de-disparo.

```
CREATE TRIGGER nome_trigger
  (AFTER|BEFORE) trigger-eventos ON nome-tabela
  [FOR EACH ROW]
  [WHEN condição]
  trigger-acção;        <-- PL/SQL
trigger-evento = INSERT|DELETE|UPDATE [OF atributos]
```


Exemplos de triggers

Suponha que temos as relações:

```
Empregado(Nome, _EBI, Salario, Ndep, SuperBI)
Departamento(Dnome, _Dnum, TotalSal, DirBI)
```

`TotalSal` é um atributo derivado e manter o seu valor actualizado pode ser feito através de um trigger. Os eventos que podem determinar uma alteração do valor desse atributo são:

1. Inserir (um ou mais) novos empregados

```
R1: CREATE TRIGGER TotalSal1
    AFTER INSERT ON Empregado
    FOR EACH ROW
    WHEN (NEW.Ndep IS NOT NULL)
      UPDATE Departamento
      SET TotalSal=TotalSal+NEW.Salario
      WHERE Dnum=NEW.Ndep;
```

Asserções

Permitem verificar condições genéricas: qualquer expressão que pode aparecer a seguir a `WHERE`.

```
CREATE ASSERTION nome CHECK (condição)
```

Qualquer alteração à BDs que leve a condição a tomar o valor `FALSO` será rejeitada. A condição tem (obviamente) de dar um valor booleano. Exemplos:

1- A duração total de todos os filmes de um dado estúdio não deve exceder 10000 minutos.

```
CREATE ASSERTION TotDuração CHECK
  (10000 >=ALL (SELECT SUM(Duração) FROM Filme GROUP BY Estúdio);
```

2- O salário de um empregado não pode ser maior do que o salário do director do departamento onde o empregado trabalha.

```
CREATE ASSERTION TectoSalario CHECK
  (NOT EXISTS (SELECT *
               FROM Empregado E, Empregado S, Departamento D
               WHERE E.Salario>S.Salario AND E.Ndep=D.Dnum AND
                     D.DirBI=S.EBI));
```


Triggers (gatilhos) em SQL3

- Baseiam-se em regras do tipo *evento-condição-acção* (ECA)
  – O *evento* que faz disparar o trigger. São normalmente operações `UPDATE`, mas também pode ser temporal.
  – A *condição* que determina se a acção deve ser executada.
  – A *acção* a tomar.
- A acção pode ser executada antes, depois ou em vez do evento que a dispara.

Exemplo: Trigger para comparar o salário de um empregado com o do seu supervisor.

```
CREATE TRIGGER InformaSupervisor
  BEFORE INSERT OR UPDATE OF Salario, SuperBI ON Empregado
  FOR EACH ROW
  WHEN (NEW.Salario> (SELECT Salario FROM Empregado
                      WHERE EBI=NEW.SuperBI))
    Informa_Supervisor(New.SuperBI, New.EBI);
```

Manutenção de integridade referencial

- são proibidas inserções (`INSERT`) e actualizações (`UPDATE`) na tabela referenciante (que contém a chave-externa) que violem a integridade referencial.
- as eliminações (`DELETE`) e actualizações (`UPDATE`) na tabela referenciada (externa) que violem a integridade referencial são tratadas de acordo com a acção indicada:
  - `CASCADE` – propaga as eliminações à tabela referenciante.
  - `SET NULL` – coloca valores nulos nos atributos referenciantes.
  - `SET DEFAULT` – coloca valores nulos nos atributos referenciantes.
  - por omissão: proíbe essas actualizações ou eliminações.
- Exemplo:

```
CREATE TABLE Estudio(
  nome CHAR(30) PRIMARY KEY, morada  VARCHAR(80),
  presidente CHAR(9) REFERENCES Presidente(PBI)
       ON DELETE SET NULL,
       ON UPDATE CASCADE);
```


Restrições sobre atributos

Os valores permitidos para os atributos podem ser restringidos

- através de restrições expressas na sua definição
- através de restrições expressas num domínio usado na sua definição.

$\rightarrow$ `CHECK condição` – a condição envolve o atributo cujos valores se quer restringir, mas pode assemelhar-se à condição de uma clausula `WHERE`.

```
sexo CHAR(1) CHECK (sexo IN ('F','M'))
```

```
presidente CHAR(9) CHECK (presidente IN (SELECT PBI FROM Presidente))
```

$\rightarrow$ restrições de domínio:

```
CREATE DOMAIN sexoDom CHAR(1)  CHECK (VALUE IN ('F','M'));
```

`VALUE` refere-se a um valor do domínio.

Restrições na chave

• ao definir-se uma chave na definição da CREATE TABLE garante-se que operações de inserção ou actualização satisfazem os requisitos de unicidade da chave.

```
CREATE TABLE Actor(
  nome  CHAR(30) PRIMARY KEY,
  ...
)
```

```
CREATE TABLE Filme(
  título CHAR(30),
  ano  CHAR(10), ...
  PRIMARY KEY (título,ano));
```

• pode-se proibir a existência de valores nulos para um dado atributo. Basta adicionar `NOT NULL` na definição do atributo na tabela.

```
CREATE TABLE Actor(
  nome  CHAR(30) PRIMARY KEY,
  dNasc DATE NOT NULL,
  ...
)
```


Restrições de integridade referencial

• Chaves-externas: os valores refereciados têm de existir.

• Se a chave externa for apenas um atributo, pode-se adicionar à definição do atributo: `REFERENCES <tabela>(<atributo>)`

• Ou usar a definição `FOREIGN KEY`.

• Exemplo:

```
CREATE TABLE Participa(
  actor  CHAR(30) REFERENCES Actor(nome),
  filme  CHAR(50),
  ano    CHAR(10),
  salario DECIMAL(9,2),
  FOREIGN KEY (filme,ano) REFERENCES Filme(título,ano)
);
```

Visões: exemplos

Q32: Criar uma visão que contenha para cada departamento o nome, o número de empregados e o custo envolvido com salários nesse departamento.

```
CREATE VIEW DepStats (Dnome, Nemps, TotSalario)
AS SELECT Dnome, COUNT(*), SUM(Salario)
FROM  Departamento, Empregado
WHERE Dnum=Ndep             <-- junção entre Departamento e Empregado
GROUP BY Dnome;
```

Notar que o facto de uma visão incluir uma junção de tabelas dificulta muito a actualização da visão pois seria díficil ver como fazer reflectir a operação de actualização nas tabelas envolvidas.

Elementos activos em SQL

- são expressões ou comandos, escritos uma vez e guardados na BDs, que se espera sejam executados quando determinados eventos ocorrerem.
- o SQL2 permite indicar restrições associadas a relações que são testadas sempre que há modificaçãoes nessas relações:
  - restrições na chave
  - restrições de integridade referencial
  - restrições de domínio
  - asserções genéricas
- O SQL3 permite a definição de gatilhos (triggers)

GROUP BY e HAVING: exemplos

Q31: Para todas as componentes cuja quantidade total fornecida for maior do que 300 (excluindo do total todos os fornecimentos cuja quantidade é igual ou inferior a 200), obtenha o número da componente e a quantidade máxima fornecida dessa componente; ordene o resultado por ordem decrescente das quantidades máximas.

```
SELECT Cnr, MAX(Qtd)
FROM   FC
WHERE  Qtd>200
GROUP BY Cnr
HAVING SUM(Qtd)>300
ORDER BY Qtd DESC
```


Visões em SQL: CREATE VIEW e DROP VIEW

- uma visão (view) é uma tabela derivada de outras tabelas.
- uma visão é uma tabela virtual que pode não existir fisicamente em disco.
- útil para especificar uma tabela que agrupe informação que precisamos de referenciar com frequência, pese embora a informação possa residir fisicamente em outras tabelas.
- não tem limitação no acesso; mas a actualização levanta problemas...
- uma visão está sempre actualizada. Se modificarmos os tuplos nas tabelas de que a visão depende, a visão vê essas alterações automaticamente (responsabilidade do DBMS).

Para definir uma visão: `CREATE VIEW nome [ atributos ] AS comando_selecção`

Para remover uma visão: `DROP VIEW nome`

Agrupamentos: GROUP BY e HAVING

GROUP BY: cria grupos de tuplos t.q. em cada grupo todos os tuplos têm o mesmo valor para o atributo de agrupamento. Útil para aplicação de funções de agregação.

Q28: Para cada componente fornecida, obter o número de componente e o total do fornecimento para essa componente.

```
SELECT Cnr, SUM(Qtd) AS Total     Res: (Cnr,Total)=(C1,600),(C2,1000),
FROM   FC                                (C3,400),(C4,500),
GROUP BY Cnr                             (C5,500),(C6,100)
```

HAVING: é normalmente usada para restringir ou eliminar grupos; equivale à clausula WHERE mas associado aoGROUP BY.

Q29: Obter os números das componentes de todas as componentes fornecidas por mais do que um fornecedor.

```
SELECT Cnr
FROM   FC
GROUP BY Cnr
HAVING COUNT(*)>1  <-- ficam os grupos que satisfazem esta condição
```


GROUP BY e HAVING: exemplos

Q30: Para cada departamento com mais de 5 empregados, obter o número do departamento e o número de empregados com salário superior a 150 contos.

```
SELECT Dnum, COUNT(*)
FROM   Deparatamento, Empregado
WHERE  Dnum=Ndep AND Salario > 150 AND
       Ndep IN (SELECT Ndep
                FROM Empregado
                GROUP BY Ndep
                HAVING COUNT(*)>5)
GROUP BY Dnum;
```

Note-se que também podemos associar funções de agregação aos atributos da clausula HAVING.

Funções de agregação

Em muitos casos pretende-se que no resultado de uma questão apareçam contagens, médias, valores máximos ou mínimos de determinados atributos.

As funções de agregação aparecem normalmente na clausula `SELECT` e associadas a um atributo:

```
SELECT  [COUNT | SUM | MAX | MIN | AVG]  atributos
```

`Q24`: Obter o número total de empregados do departamento 5.
`Q25`: Obter o total de salários de todos empregados, o salário máximo e mínimo, e a média dos salários.

```
Q24: SELECT COUNT(*)
     FROM   Empregado
     WHERE  Ndep=5
```

```
Q25: SELECT SUM(Salario), MAX(Salario),
            MIN(Salario), AVG(Salario)
     FROM Empregado
```


Funções de agregação: exemplos

`Q26`: Obtenha o número de fornecedores com categoria inferior à categoria máxima actual na tabela de fornecedores.

```
Q24: SELECT COUNT(Fnum)
     FROM   F
     WHERE  Cat < (SELECT MAX(Cat) FROM F);
```

`Q26`: Obtenha os números de fornecedores cuja categoria é maior ou igual à média para a respectiva cidade.
`Q27`: Obter os nomes de todos empregados que têm 2 ou mais dependentes.

```
Q26: SELECT F1.Fnum
     FROM   F AS F1
     WHERE  F1.Cat >=
          (SELECT AVG(Cat)
           FROM F
           WHERE Cidade=F1.Cidade);
```

```
Q27: SELECT Pnome, Unome
     FROM   Empregado AS E
     WHERE
          (SELECT COUNT(*)
           FROM DEPENDENTE AS D
           WHERE D.EBI=E.EBI);
```

Exemplos de questões encadeadas: IV

Q21: Obter o primeiro e último nome dos funcionários cujo salário é maior do que o salário de qualquer empregado do departamento 5.

```
SELECT Pnome,Unome
FROM   Empregado
WHERE  Salario >ALL (SELECT Salary
                     FROM Empregado
                     WHERE Ndep=5);
```

Q22: Obter o nome dos empregados que trabalham em *todos* os prjectos controlados pelo departamento 5.

```
SELECT Pnome, Unome
FROM   Empregado AS E
WHERE  ( (SELECT Pnum
          FROM TrabalhaEm AS T
          WHERE E.EBI=T.EBI)
          CONTAINS
         (SELECT Pnum
          FROM Projecto AS T
          WHERE Dnum=5));
```


Renomeação de atributos e tabelas de junção

É possível renomear os atributos que aparecem no resultado da questão, fazer:

```
SELECT atributo AS novo_nome ...
```

Tabelas de junção: o SQL2 já permite que a junção de tabelas seja feita de forma explicíta. Exemplo, a questão Q23: Obter o nome e endereço de todos os empregados que trabalham para o departamento Vendas.

```
SELECT Pnome, Unome, Endereço
FROM   Empregado, Departamento
WHERE  Dnome='Vendas' AND Dnum=Ndep; <-- junção implicita

ou

SELECT Pnome, Unome, Endereço
FROM   (Empregado JOIN Departamento ON Ndep=Dnum)  <-- junção explicita
WHERE Dnome='Vendas';
```

Outras operações para a junção: NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN.

Exemplos de questões encadeadas II

Q16: Obter os números de fornecedores com categoria inferior à categoria do fornecedor F1.

```
SELECT Fnum                                                    Res: F2
FROM   F
WHERE  Cat <ALL (SELECT Cat FROM F WHERE Fnum='F1');
```

Ambiguidade na questão Q16?

Não, porque por regra *uma referência a um atributo não qualificado, refere-se à relação declarada na questão mais interior*.

Q17: Obter os nomes dos fornecedores que fornecem pelo menos uma componente.

```
SELECT Fnome                                          Res: F1,F2,F3,F4
FROM   F
WHERE  EXISTS (SELECT * FROM FC WHERE Fnr=Fnum);
```


Exemplos de questões encadeadas: III)

Q18: Obter os nomes dos fornecedores que não fornecem a componente C2.

```
SELECT Fnome                                          Res: F1,F2,F3,F4
FROM   F
WHERE  NOT EXISTS (SELECT *
                   FROM FC
                   WHERE Fnr=Fnum AND Cnr='C2');
```

Q19: Obter os nomes dos fornecedores que fornecem todas as componentes.
ou "obter os nomes dos fornecedores para os quais não existe uma componente que eles não forneçam".

```
SELECT Fnome                                          Res: F1
FROM   F
WHERE  NOT EXISTS (SELECT *
                   FROM C
                   WHERE NOT EXISTS (SELECT *
                                     FROM FC
                                     WHERE Fnr=Fnum AND Cnr=Cnum));
```

Condições envolvendo relações

- `EXISTS R` – verdadeiro sse `R` (sub-consulta) não for vazio.
- `s IN R` – verdadeiro sse `s` for igual a um dos valores de `R`.

  `s NOT IN R` – verdadeiro sse `s` não for igual a qualquer valor de `R`.

  `R` é uma sub-consulta ou uma lista de átomos.
- `s >ALL R` – verdadeiro sse `s` for maior do que todos os valores em `R`.

  Podemos ter outros operadores de comparação, e.g. `s <>ALL R` que tem o mesmo significado que `s NOT IN R`.
- `s >ANY R` – verdadeiro sse `s` for maior do que pelo menos um valor em `R`.

  Podemos ter outros operadores de comparação, e.g. `s =ANY R` tem o mesmo significado que `s IN R`.
- Os operadores `EXISTS`, `ALL` e `ANY` podem ser negados colocando-se `NOT` antes.

  `NOT EXISTS R` – verdadeiro sse R for vazio.

Exemplos de questões encadeadas: I

`Q13:` Obter os números de fornecedores que fornecem *pelo menos uma* componente de cor vermelha.

```
SELECT Fnr                                                    Res: F1,F4
FROM   FC
WHERE  Cnr IN (SELECT Cnum FROM C WHERE Cor='verm');
```

`Q14`: Obter os números de componentes cujo peso é 12, 16 ou 17 grs.

```
SELECT Cnum                                                   Res: C1,C2,C3
FROM   C
WHERE  Peso IN (12,16,17);
```

`Q15`: Obter o BI todos os empregados que trabalham nos mesmas combinações (projecto,horas) que o empregado 'Joao Regras' cujo EBI é '1234567'.

```
SELECT EBI
FROM   TrabalhaEm
WHERE  (Pnum,Horas) IN (SELECT Pnum, Horas
                        FROM TrabalhaEm
                        WHERE EBI='1234567');
```

Ordenação do resultado

• **ordenação – ORDER BY**:
Q11: Listar os nomes dos alunos inscritos em BD por ordem alfabética.

```
SELECT nome
FROM Alunos, Inscrições
WHERE codDisc='BD' AND codigo=codAluno
ORDER BY nome
```

A ordenação é por defeito por ordem crescente. Podemos indicar exactamente o que se pretende:

```
ORDER BY [ASC|DESC] atributo
```


Questões encadeadas

Por vezes é necessário comparar um atributo com os valores de uma relação obtida por uma questão encadeada.
A questão Q12: "Listar o Director do Departamento Vendas" pode ser formulada em SQL como:

```
SELECT E.Pnome, E.Unome
FROM Empregado AS E, Departamento AS D
WHERE D.nome='Vendas' AND E.EBI=D.dirEBI;

ou

SELECT nome
FROM Empregado
WHERE EBI =  (SELECT DirEBI
              FROM Departamento
              WHERE nome='Vendas');
```

Note-se que o resultado da sub-consulta é um valor apenas.

Questões envolvendo operações sobre conjuntos.

O SQL incorpora directamente algumas das operações sobre conjuntos da álgebra relacional, como sejam:

```
UNION -- reunião
EXCEPT -- diferença
INTERSECT -- intersecção
```

as relações resultantes são conjuntos de tuplos, em que os tuplos em duplicado são eliminados do resultado.

→ caso não se queira que os tuplos em duplicado sejam eliminados, deve seguir-se ao operador o qualificativo `ALL`: `R INTERSECT ALL S`.
→ estas operações só se aplicam a relações "compatíveis para a reunião".

`Q8`: Obter os números das componentes que ou pesam mais de 18 grs ou são fornecidas pelo fornecedor F2 (ou ambos).

```
(SELECT C.Cnum FROM C WHERE C.Peso > 18)
 UNION
(SELECT FC.Cnum FROM FC WHERE FC.Fnum='F2')
```


Operadores de comparação e aritméticos

- **Strings:** `atributo = 'string'` ou `atributo LIKE '_a%'`.
- **Ops aritméticos:** `+, *, -, /`

`Q9`: Mostrar os salários dos empregados que trabalham no projecto 5 com um aumento de 10%.

```
SELECT Pnome, Unome, 1.1*Salario
FROM Empregado AS E, Projecto AS P, TrabalhaEm AS T
WHERE Pnum=5 AND P.Pnum=T.Pnum AND E.EBI=T.EBI
```

- **operador BETWEEN**:

`Q10`: Obter todos os empregados do departamento 7 cujo salario está entre 100c e 150c.

```
SELECT *
FROM Empregado
WHERE Ndep=7 AND (Salario BETWEEN 100 AND 150)
```

UPDATE

Permite modificar os valores dos atributos de um ou mais tuplos de uma relação.

```
UPDATE relação SET novas_valores WHERE condição;
```

A condição de `WHERE` selecciona os tuplos a modificar. Em `SET` indicam-se os atributos que serão modificados e os respectivos novos valores. os seus novos valores relação são removidos mas a tabela mantem-se, embora vazia.

Exemplos:

```
M4: UPDATE Disciplinas
    SET nome='Redes e Sistemas Distribuídos'
    WHERE  nome IN (SELECT nome
                    FROM Disciplinas
                    WHERE nome LIKE 'Compl%Operacao' AND anoLect > '96/97';

M5: UPDATE Empregado
    SET salario= salario*1.05
    WHERE Ndep= 5;
```


BD sobre Fornecedores e Componentes

F	Fnum	Fnome	Cat	Cidade
	F1	SOS	20	Lisboa
	F2	TLP	10	Porto
	F3	AMP	30	Porto
	F4	CCB	20	Lisboa
	F5	RTB	30	Braga

C	Cnum	Cnome	Cor	Peso	Cidade
	C1	prego	verm	12	Lisboa
	C2	porca	verde	17	Porto
	C3	parafuso	azul	17	Coimbra
	C4	parafuso	verm	14	Lisboa
	C5	anilha	azul	12	Porto
	C6	anilha	verde	19	Lisboa

FC	Fnr	Cnr	Qtd
	F1	C1	300
	F1	C2	200
	F1	C3	400
	F1	C4	200
	F1	C5	100
	F1	C6	100
	F2	C1	300
	F2	C2	400
	F3	C2	200
	F4	C2	200
	F4	C4	300
	F4	C5	400

INSERT

Permite adicionar tuplos a uma relação.

INSERT INTO R(A_1,...,A_n) VALUES (v_1,...,v_n);

O tuplo a inserir tem o valor v_i no atributo A_i, para $i = 1, 2, \ldots, n$.

M1: Adicionar um novo aluno à tabela de Alunos

```
INSERT INTO Alunos
VALUES ('19990307007','Alice','1982-05-13','Rua Direita, 4 -- 4000 Porto');
```

quando não se indica os atributos da tabela, supõe-se que são todos e pela ordem definida quando da sua criação.

→ se adicionarmos um tuplo com valores para apenas alguns dos atributos, é necessário indicar quais, ou respeitar a ordem dos atributos na tabela. Os restantes ficam com valores por defeito ou Null.

→ é possível adicionar tuplos em que os valores são obtidos como resultado de uma questão:

INSERT INTO R(A_1,...,A_n) SELECT A_1,...,A_n FROM ...;

DELETE

Permite eliminar tuplos de uma relação.

DELETE FROM relação WHERE condição_remoção;

Se a condição WHERE for omitida, todos os tuplos da relação são removidos mas a tabela mantem-se, embora vazia.

Exemplos:

```
M2: DELETE FROM Inscrições
    WHERE  anoLect < '97/98';

M3: DELETE FROM Inscrições;
```

→ a condição a seguir ao WHERE pode ser uma questão complexa (veremos exemplos mais adiante).

→ esta operação pode propagar-se a outras tabelas para preservar a integridade referêncial.

DISTINCT

O SQL não elimina automaticamente tuplos em duplicado, porque:

- eliminar duplicados é dispendioso ⇒ ordenar+remover
- se se pretender aplicar funções de agregação (somas, contagens, médias,...) não se pretende, na maioria dos casos, eliminar duplicados.
- o utilizador pode querer ver os tuplos em duplicado no resultado

Para eliminar duplicados, usa-se:

```
SELECT DISTINCT lista_atributos
FROM  relação
```

Q7: Nomes de alunos que estão inscritos em 99/00 em disciplinas do DCC.

```
SELECT DISTINCT A.nome                      <-- evita que um aluno apareça
FROM  Alunos AS A, Disciplinas AS D, Inscrições AS I
WHERE I.anoLect='99/00' AND D.depto='DCC'    -- tantas vezes quantas as disc.
      AND I.codDisc=D.codigo                 -- do DCC em que esta inscrito
      AND I.codAluno=A.codigo;
```


Comandos SQL para actualização da BDs

Iremos ver formas mais complexas e potentes de interrogar uma BDs, mas primeiro vamos ver os comandos que permitem modificar o estado da BDs. São três:

- **INSERT** – adicionar tuplos a uma relação
- **DELETE** – remover tuplos de uma relação
- **UPDATE** – actualizar valores de atributos de alguns tuplos

Ambiguidades os nomes de atributos e Sinónimos

Se as relações envolvidas na questão tiverem nomes de atributos em comum, pode verificar-se uma situação ambígua. A solução é qualificar os atributos com o nome da relação a que pertencem.

Q4: Obter os nomes dos alunos e das disciplinas a que estão inscritos.

```
SELECT A.nome, D.nome                          <-- qualif. dos atributos
FROM  Alunos AS A, Disciplinas AS D, Inscrições <-- sinónimos ou aliases
WHERE A.codigo=codAlunos AND D.codigo=codDisc
```

Q5: Listar para cada empregado o 1o. e último nome, e o 1o e último do seu supervisor imediato.

```
SELECT E.Pnome, E.Unome, S.Pnome, S.Unome
FROM  Empregado AS E, Empregado AS S
WHERE E.superBI=S.EBI;
```


SELECT * e Comparação de strings

O simbolo * pode substituir a lista de atributos a seleccionar, significando que se pretende seleccionar todos os atributos de uma relação sem os mencionar explicitamente.

Q6: Obter os dados dos alunos de CC. O código de CC é da forma YYYY0307XXX, em que YYYY é o ano e XXX é o número de ordem.

```
SELECT *
FROM  Alunos
WHERE código LIKE '____0307%';
```

Comparação de strings: `nome_atributo [NOT] LIKE '_a%'`

→ '_' pode ser um qualquer caracter.
→ '%' pode ser um número arbitrário de caracteres.

SQL: Questões.

SELECT	lista de atributos	← *projecção*
FROM	tuplos de tabelas	← *relações*
WHERE	condição sobre	
	os tuplos seleccionados	← *selecção + outras operações*

Exemplos:

Q1: Obter a lista das disciplinas do 3o Ano.

```
SELECT nome
FROM   Disciplinas
WHERE  anoCurso=3;
```

na Álgebra Relacional: $\pi_{\texttt{nome}}(\sigma_{\texttt{anoCurso=3}}(\texttt{Disciplinas}))$

Exemplos questões simples de SQL.

Q2: Obter o nome de todos os alunos inscritos na disciplina de EDA bem 99/00.

```
SELECT nome
FFROM Alunos, Inscrições
WHERE codDisc=‘‘EDA’’ AND      <--- condição de selecção
      anoLect=‘‘99/00’’ AND    <--- condição de selecção
      codigo=codAluno;         <--- cond. de junção entre 2 relações
```

na álgebra relacional: $\pi_{\texttt{nome}}(\texttt{Aluno} \bowtie_{\texttt{codigo=codAluno}} (\sigma_{\texttt{codDisc=EDAANDanoLect=99/00}}(\texttt{Inscricoes})))$

Omissão do WHERE – significa que não impomos condições, pelo que todos os tuplos da relação indicada em FROM são seleccionados.

Q3: Seleccionar todos os códigos e nomes de disciplinas.

```
SELECT codigo, nome
FROM Disciplinas
```

SQL: DROP TABLE / ALTER TABLE

- `DROP TABLE nome_tabela [CASCADE|RESTRICT]` – remove uma tabela.
  - Com `CASCADE` são também removidas todas as restrições e visões que referenciam a tabela.
  - Com `RESTRICT`, a tabela só é removida se não existirem restrições sobre a tabela.
- `ALTER TABLE nome_tabela COM atributo;` modifica uma tabela. Se `COM = DROP` (cascade ou restrict), remove um atributo ou restrição. Se `COM = ALTER`, modifica o atributo ou restrição.

Um exemplo: `ALTER TABLE Inscrições ADD exame INT;`

SQL: Definição de novos domínios

Em SQL podemos indicar um tipo de dados "default" para os atributos, mas podemos também declarar um novo domínio de dados e usá-lo na definição dos atributos.

```
CREATE DOMAIN novo_tipo AS tipo_base;
```

Exemplo: `CREATE DOMAIN CodTipo AS CHAR(7);`

Os comandos de criação de tabelas podiam agora ser:

```
CREATE TABLE Inscrições (
  codAluno CodTipo  NOT NULL,
  codDisc  CodTipo  NOT NULL,
  anoLect  CHAR(5),
  PRIMARY KEY (codAluno,codDisc),
  FOREIGN KEY (codAluno) REFERENCES Alunos(codigo),
  FOREIGN KEY (codDisc) REFERENCES Disciplinas(codigo)
);
```

SQL: CREATE TABLE

permite criar uma nova relação, indicando-se o seu nome, atributos (nome e tipo de dados associados) e restrições (chave e restrições de integridade).

```
CREATE TABLE nome_tabela (
  atributo_1 tipoDados_1 [restrição_atributo_1],  ... ,
  atributo_N tipoDados_N [restrição_atributo_N],
  [ restrição_tabela_1 {, restrição_tabela_i } ]
);
```

em que `tipoDados_i`:

numérico: INT, SHORTINT, REAL, DOUBLE PRECISION, DECIMAL(n,d) (n digitos decimais com d digitos depois do ponto decimal).

strings: CHAR(n) comprimento fixo, VARCHAR(n) comprimento variável até n caracteres, BIT(n) string de n bits, VARBIT(n) string até n bits.

Data: DATE – YYYY-MM-DD (10 caracteres)

Hora: TIME – HH:MM:SS (8 caracteres)

Exemplo CREATE TABLE

Considere-se o seguinte esquema:

`Alunos(codigo, nome, dtNasc, endereço)` – `codigo` é chave.
`Disciplinas(codigo, nome, anoCurso, depto)` – `codigo` é chave.
`Inscrições(codAluno, codDisc, anoLect)` – `(codAluno,codDisc)` é chave.

Comandos para criar as tabelas:

```
CREATE TABLE Alunos (
  codigo   CHAR(7)     NOT NULL,
  nome     VARCHAR(30) NOT NULL,
  dtNasc   DATE,
  Endereço VARCHAR(30),
  PRIMARY KEY (codigo)
);
CREATE TABLE Disciplinas (
  codigo   CHAR(7)     NOT NULL,
  nome     VARCHAR(20) NOT NULL,
  anoCurso CHAR(1),
  depto    INT,
  PRIMARY KEY (codigo)
);
CREATE TABLE Inscrições (
  codAluno CHAR(7)  NOT NULL,
  codDisc  CHAR(7)  NOT NULL,
  anoLect  CHAR(5),
  PRIMARY KEY (codAluno,codDisc),
  FOREIGN KEY (codAluno)
    REFERENCES Alunos(codigo),
  FOREIGN KEY (codDisc)
    REFERENCES Disciplinas(codigo) );
```

SQL – Structured Query Language

Linguagem para definir, questionar e actualizar a BD. É usada por todos os sistemas comerciais de BDs relacionais.

$\rightarrow$ SQL = DDL (data definition lang.) + DML (data manipulation lang.), permite:

- criar as tabelas
- definir visões sobre os dados
- criar e remover índices
- ligar-se a outras linguagens, como o C
- definir permissões de acesso a tabelas

- O SQL resultou do SEQUEL da IBM.
- SQL1 (1986) – 1a versão ANSI.
- SQL2 (1992) – 2a versão ANSI.
- SQL3 – novo standard em curso.
- SQL3 inclui conceitos OO.