Subconsultas

*** subconsulta:** forma alternativa de expressar consultas envolvendo relacionamentos entre tabelas

* **cláusulas para tratamento de subconsultas**: **IN, ANY, ALL, EXISTS**

3.1) Cláusula [NOT] IN

* relação de pertinência → **elemento** ∈ **conjunto**

* **sintaxe: where** *atributo_ou_expressão* **in** (*subconsulta*)

* **exemplos**:

1) buscar o nome dos médicos com consultas marcadas para o dia 13 de Junho de 2006

```
select nome
from Médicos
where codm in (select codm
                from Consultas
                where data = '06/06/13')
```

* **passos no processamento da consulta**:

passo 1: execução da subconsulta

passo 2: execução da consulta externa: comparação das tuplas da consulta externa com a relação resultante da subconsulta

→ otimização: filtra, na subconsulta, apenas atributos e tuplas de interesse para a execução da consulta externa

→ passos válidos para as cláusulas IN, ANY e ALL

2) buscar o nome dos pacientes com idade superior a 21 anos que têm consulta marcada

```
select nome
from Pacientes
where idade > 21
and codp in (select codp
              from Consultas)
```

* **consultar utilizando a cláusula de subconsulta IN**:

a) dados de todas as consultas marcadas para a médica Maria

b) número e andar dos ambulatórios onde nenhum médico dá atendimento (princípio do operador de **diferença** da álgebra relacional: R1 – R2)

c) dados dos pacientes que não têm consulta marcada com o médico Pedro

d) nomes dos funcionários que estão internados como pacientes (princípio do operador de **interseção** da álgebra relacional: R1 ∩ R2)

e) nome e RG dos médicos que têm consultas marcadas no período da manhã e que não dão atendimento em ambulatórios com mais de 40 leitos

* **observação**: o SQL padrão define cláusulas *EXCEPT* e *INTERSECT* para a especificação, respectivamente, de uma diferença e de uma intersecção entre duas consultas SQL (no mesmo estilo da cláusula *UNION*). O MySQL não implementa estas cláusulas.

3.2) Cláusula ANY

* permite outras formas de comparação elemento-conjunto:

= any (*subconsulta*) (= **in**)
> any (*subconsulta*) → verdadeiro se o atributo comparado for maior que algum valor de atributo das tuplas resultantes da subconsulta
< any (*subconsulta*)
< > **any** (*subconsulta*)

* **exemplo:** buscar o nome de todos os funcionários, exceto o mais idoso

```
select nome
from Funcionários
where idade < any (select idade
                   from Funcionários)
```

* **consultar utilizando as cláusulas de subconsulta vistas até agora (IN e ANY)**:

a) números e andares de todos os ambulatórios, exceto o de menor capacidade
b) nome e a idade do médico mais jovem (**sem usar função MIN!**)
c) qual(is) é (são) o nome e o RG do(s) paciente(s) que será(ão) o(s) último(s) a ser(em) atendido(s) no dia 13 de Junho de 2006?
d) número do ambulatório com maior capacidade dentre os ambulatórios onde nenhum médico dá atendimento (**desafio!**)

3.3) Cláusula ALL

* condição a ser satisfeita para todos os elementos de um conjunto:

= all (*subconsulta*) → igual a todos
> all (*subconsulta*) → maior que todos
< all (*subconsulta*) → menor que todos
< > **all** (*subconsulta*) → diferente de todos (= **not in**)

* **exemplo**: buscar o nome dos funcionários que recebem salários superiores aos salários pagos no departamento enfermaria

```
select nome
from Funcionários
where salário > all (select salário
                     from Funcionários
                     where departamento = 'enfermaria')
```

*** consultar utilizando as cláusulas de subconsulta vistas até agora (IN, ANY e ALL)**:

a) nome e RG dos pacientes com consultas marcadas para horários anteriores a todos os horários de consultas para o dia 12 de Junho de 2006

b) nome e RG dos médicos que não atendem em ambulatórios com capacidade superior à capacidade dos ambulatórios do segundo andar

c) nome dos médicos com consultas marcadas para horários mais tarde que as consultas do médico Pedro

*** observação**: a omissão das 3 cláusulas de subconsulta já vistas em uma consulta SQL indica uma comparação do tipo **elemento-elemento** (neste caso, a subconsulta deve retornar uma tabela com uma única tupla)

- **exemplo**: buscar o nome e o RG dos médicos que possuem a mesma especialidade do médico João (RG = 1000010000)

```
select nome, RG
from Médicos
where RG < > 1000010000
and especialidade = (select especialidade
                     from Médicos
                     where RG = 1000010000)
```

- **consultar utilizando comparações elemento-elemento sempre que possível**:

a) número dos ambulatórios com capacidade superior à capacidade do ambulatório 1

b) nome e RG dos funcionários que não trabalham no departamento da funcionária Lúcia (RG = 2200020000) e não estão internados como pacientes

c) nome e RG dos médicos que atendem em ambulatórios que estão em andares superiores ao andar onde atende o médico João (codm = 1)

3.4) Cláusula [NOT] EXISTS

* **exists** (*subconsulta*) → verdadeiro se a tabela resultante da subconsulta não for vazia na avaliação de uma tupla associada à tabela da consulta externa

* processa a subconsulta na avaliação de cada tupla da consulta externa

* **exemplos**:

1) buscar o nome dos médicos com consultas para 13 de Junho de 2006

```
select nome
from Médicos m
where exists (select  *
                    from consultas
                    where data = '06/06/13'
                    and codm = m.codm)
```

* atributos do resultado da subconsulta são irrelevantes. Importa apenas saber se existem tuplas na resposta da subconsulta

* a cláusula **exists** implementa o **quantificador existencial** do cálculo relacional. Veja a consulta acima escrita no cálculo relacional:

{**t.nome** | t ∈ Médicos ∧ ∃ c ∈ Consultas (c.data = '06/06/13' ∧
t.codm = c.codm) }

*** consultar utilizando apenas a cláusula exists**:

a) nomes dos pacientes que também são funcionários

b) número do ambulatório com a menor capacidade

c) nomes dos médicos que têm consulta marcada com a paciente Ana

2) buscar os nomes dos médicos que têm consulta marcada com **todos** os pacientes (princípio do operador de **divisão** da álgebra relacional)

```
select  m.nome
from Médicos m
where not exists (select *
                    from Pacientes p
                    where not exists (select *
                                        from Consultas c
                                        where p.codp = c.codp
                                         and m.codm = c.codm))
```

***** interpreta-se da seguinte forma: *"buscar os nomes dos médicos tais que não existam pacientes que não tenham consultas marcadas com eles"*

Simulação:

Médicos (codm)	Pacientes (codp)	Consultas (codm, codp)
1	1	1 1
2	2	1 4
3	3	**2** 1
4	4	**2** 2
		2 3
		2 4

* Para o médico 1: existem pacientes (2 e 3) que não têm consultas marcadas com este médico **(1ª subconsulta retorna não vazia)**
* Para o médico 2: não existe paciente que não tenha consulta marcada com este médico **(1ª subconsulta retorna vazia) – Ele é buscado!**

* como a linguagem SQL implementa apenas o quantificador existencial, ela se baseia na propriedade de equivalência entre os quantificadores universal (∀) e existencial (∃) para suportar consultas do tipo *"para todos"*:

$$\forall\ t \in r\ (P(t)) \equiv \neg\ \exists\ t \in r\ (\neg\ P(t))$$

ou seja, se desejo verificar que, para todo paciente exista um médico **M** que tenha consulta marcada com ele, isto significa dizer que não existe paciente que não tenha consulta marcada com o médico **M**

*** realizar as seguintes atualizações no BD:**

a) o médico Carlos agora dá atendimento no ambulatório 1

b) a paciente Lúcia (codp = 3) marcou consulta com o médico João (codm = 1) para o dia 14 de Junho de 2003, às 16:30

* **realizar as seguintes consultas:**

a) nome e RG dos pacientes que têm consultas marcadas com todos os médicos

b) nome e RG dos médicos ortopedistas que têm consultas marcadas com todos os pacientes de Fpolis

c) número do ambulatório no qual todos os médicos ortopedistas dão atendimento, se existir

3.5) Subconsultas na Cláusula FROM

* **from** (*subconsulta*) as *nome_tabela* → permite a geração de uma tabela derivada, cujo nome é *nome_tabela*, que é utilizada no processamento da consulta externa

* permite uma otimização eficiente da consulta, pois restringe linhas e colunas das tabelas envolvidas antes de realizar um produto cartesiano ou junção

*** exemplos**:

1) buscar os dados dos médicos com consultas para 13 de Junho de 2006 e os horários das suas consultas

```
select Medicos.*, C.hora
from Medicos join
        (select codm, hora
         from Consultas
         where data = '06/06/13') as C
on Médicos.codm = C.codm
```

2) buscar o número e o andar dos ambulatórios em que médicos de Florianópolis dão atendimento

```
select Amb.*
from        (select nroa, andar
            from ambulatorios) as Amb
      join
            (select nroa
            from Medicos
            where cidade = 'Fpolis')  as M_ort
      on Amb.nroa = M_ort.nroa
```

* **consultar, utilizando subconsultas na cláusula FROM**:

a) dados de todas as consultas marcadas para a médica Maria

b) nome e cidade dos pacientes que têm consultas marcadas com médicos ortopedistas

c) nome e RG dos pacientes de Florianópolis que não têm consulta marcada com o médico João