

Bases de Dados

T15 - SQL Avançado Parte II

Prof. Daniel Faria

Prof. Flávio Martins

Sumário

- Recapitulação
 - Agregação
 - Selects Encadeados
 - Combinações de Conjuntos
 - Determinação de Elemento Distintivo
- Divisão
- NULLs

Recapitulação

Interrogação de Bases de Dados

SQL ← Álgebra Relacional

```
[WITH with_query [, ...]]
```

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
```

```
  [* | expression [[AS] output_name] [, ...]]
```

```
  [FROM from_item [, ...]]
```

```
  [WHERE condition]
```

```
  [GROUP BY [ALL | DISTINCT] grouping_element [, ...]]
```

```
  [HAVING condition]
```

```
  [{UNION | INTERSECT | EXCEPT} [ALL | DISTINCT] select]
```

```
  [ORDER BY expression [ASC | DESC | USING operator]
```

```
    [NULLS { FIRST | LAST}] [, ...]]
```

```
  [LIMIT {count | ALL}]
```

Project, Rename, Aggregation

Cartesian Product, Joins

Select

Aggregation w/ Grouping

Select (after Aggregation)

Union, Intersection, Difference

Agregação

$G_{\text{COUNT-DISTINCT}(\text{producer})}(\text{produce})$

```
SELECT COUNT(DISTINCT producer) FROM produce;
```

Sem **GROUP BY**: agrega toda a tabela

$seller \xrightarrow{G_{\text{SUM}(\text{price}) \rightarrow \text{total}}} (buys)$

```
SELECT seller, SUM(price) AS total FROM buys GROUP BY seller;
```

Com **GROUP BY**: agrega a tabela para cada valor das colunas no GROUP BY

$\sigma_{\text{total} > 1000} (seller \xrightarrow{G_{\text{SUM}(\text{price}) \rightarrow \text{total}}} (buys))$

```
SELECT seller, SUM(price) AS total FROM buys GROUP BY seller HAVING total > 1000;
```

HAVING: seleção após a agregação

Agregação com Agrupamento

- **GROUP BY** é **obrigatório** quando se retorna colunas agregadas e não agregadas, mas pode ser usado quando se retorna apenas uma das duas
- Deve-se quase sempre retornar a(s) coluna(s) usada(s) para agrupar, ou os resultados não farão sentido
- Pode retornar-se mais colunas, não usadas para agrupar, mas devem ser funcionalmente dependentes das colunas a agrupar
- **HAVING** tem de ser sempre precedido por **GROUP BY**
- Permite fazer agregação sem retornar o agregado

Selects Encadeados

Cláusula	Operador	Alias	Restrições	Uso
WITH	–	Antes	–	Simplificar queries computando SELECTs encadeados de antemão
FROM	–	Depois	–	Fazer queries complexas (e.g. agregação de agregação, seleção antes de join)
WHERE/ HAVING	[NOT] IN	–	Devolve Array	Seleção com base numa lista de valores em vez de um valor único: interseção (IN, =ANY), diferença (NOT IN, !=ALL), encontrar máximo ou mínimo (>=ALL, <=ALL)
	ANY	–		
	ALL	–		
	[NOT] EXISTS	–	Ligado ao SELECT externo	Semelhante aos operadores de arrays, mas mais flexível

Combinar Conjuntos

Combinação	Opção	Cláusula	Restrições
União	UNION	UNION entre dois SELECT	SELECTs c/ colunas compatíveis
	FULL JOIN	FROM	–
Interseção	INTERSECT	INTERSECT entre dois SELECT	SELECTs c/ colunas compatíveis
	INNER JOIN	FROM	–
	IN / =ANY	WHERE c/ SELECT encadeado	SELECT devolve array
	EXISTS	WHERE c/ SELECT encadeado	SELECT ligado ao externo
Diferença	EXCEPT	EXCEPT entre dois SELECT	SELECTs c/ colunas compatíveis
	LEFT JOIN + IS NULL	FROM + WHERE	Seleção na tabela da direita requer SELECT encadeado
	NOT IN / !=ALL	WHERE c/ SELECT encadeado	SELECT devolve array
	NOT EXISTS	WHERE c/ SELECT encadeado	SELECT ligado ao externo

Opções para Interseção

- Que nomes são partilhados por porcos e não porcos?

```
SELECT name FROM pig INTERSECT SELECT name FROM nonpig;
```

→ Eficiente mas requer colunas compatíveis

```
SELECT DISTINCT name FROM pig INNER JOIN nonpig USING (name);
```

→ Quase sempre boa opção; única que permite devolver colunas das duas tabelas

```
SELECT DISTINCT name FROM pig WHERE name IN (SELECT name FROM nonpig);
```

→ Eficiente quando só queremos filtrar uma coluna

```
SELECT DISTINCT name FROM pig p WHERE EXISTS  
    (SELECT name FROM nonpig n WHERE n.name=p.name);
```

→ Menos eficiente mas mais versátil para filtros complexos

Opções para Diferença

- Que porcos não participaram em vendas?

```
SELECT id FROM pig EXCEPT SELECT seller FROM buys;
```

Eficiente mas requer colunas compatíveis

```
SELECT id FROM pig LEFT JOIN buys ON (id=seller) WHERE code IS NULL;
```

Eficiente em casos em que não queremos filtrar a tabela da direita; mais complexa noutros casos

```
SELECT id FROM pig WHERE id NOT IN (SELECT seller FROM buys);
```

Eficiente quando só queremos filtrar uma coluna

```
SELECT id FROM pig WHERE NOT EXISTS  
    (SELECT seller FROM buys WHERE seller=id);
```

Menos eficiente mas mais versátil para filtros complexos

Determinação do Elemento Distintivo

- Qual espécie de animal mais produtiva?

GROUP BY species

SUM(amount)

```
SELECT species
FROM produce JOIN nonpig ON (producer=id)
GROUP BY species HAVING SUM(amount) >= ALL(
    SELECT SUM(amount)
    FROM produce JOIN nonpig ON (producer=id)
    GROUP BY species
);
```

Retornar agrupador

Ligar agrupador ao agregado

Agrupar, agregar e comparar

Equivalente à query externa mas retornando agregado

Determinação do Elemento Distintivo

- Qual o porco que realizou menos vendas (de entre os que realizaram alguma)?

GROUP BY seller

COUNT (*)

```
SELECT seller
```

```
FROM buys
```

```
GROUP BY seller HAVING COUNT(*) <= ALL(
```

```
    SELECT COUNT(*)
```

```
    FROM seller
```

```
    GROUP BY seller
```

```
);
```

Retornar agrupador

Ligar agrupador ao agregado

Agrupar, agregar e comparar

Equivalente à query externa mas retornando agregado

Determinação do Elemento Distintivo

- Que espécie de animal é mais lucrativa?

Determinação do Elemento Distintivo

- Que espécie de animal é mais lucrativa?



`GROUP BY species`



`AVG(price)`

Determinação do Elemento Distintivo

- Que espécie de animal é mais lucrativa?



GROUP BY species



AVG(price)

```
SELECT species
FROM buys JOIN produce USING (code) JOIN nonpig ON (producer=id)
GROUP BY species HAVING AVG(price) >= ALL(
    SELECT AVG(price)
    FROM buys JOIN produce USING (code) JOIN nonpig ON (producer=id)
    GROUP BY species
);
```

Determinação do Elemento Distintivo

- Que porco vendeu produtos de mais vacas?

Determinação do Elemento Distintivo

- Que porco vendeu produtos de mais vacas?



GROUP BY seller



COUNT(*)

```
SELECT seller
FROM buys JOIN produce USING (code) JOIN nonpig ON (producer=id)
WHERE species = 'cow'
GROUP BY seller HAVING COUNT(*) >= ALL(
    SELECT COUNT(*)
    FROM buys JOIN produce USING (code) JOIN nonpig ON (producer=id)
    WHERE species = 'cow'
    GROUP BY seller
);
```

Determinação do Elemento Distintivo

- Que porco vendeu produtos de mais vacas?



`GROUP BY seller`



`COUNT (*)`



Divisão

Divisão

- Operação inversa ao produto cartesiano
- Seleciona tuplos de uma relação que contêm todos os tuplos de uma segunda relação, devolvendo os atributos da primeira relação que não estão na segunda
- Útil para queries de cobertura
 - E.g. que porcos realizaram vendas de produtos de todas as espécies de animais?

a1	b1
a2	b2
a3	b3

 ×

b1	c1
b3	c2

 =

a1	b1	b1	c1
a1	b1	b3	c2
a2	b2	b1	c1
a2	b2	b3	c2
a3	b3	b1	c1
a3	b3	b3	c2

a1	b1	b1	c1
a1	b1	b3	c2
a2	b2	b1	c1
a2	b2	b3	c2
a3	b3	b1	c1
a3	b3	b3	c2

 ÷

b1	c1
b3	c2

 =

a1	b1
a2	b2
a3	b3

Divisão

- Que porcos realizaram vendas de produtos de todas as espécies de animais?
 - Como resolver em SQL se a operação divisão não está implementada (o que é verdade para a maioria dos SGBD SQL)?
 - Podemos inspirar-nos na definição de divisão em Álgebra Relacional:

$$\blacksquare \quad r \div s = \Pi_{r \cap \neg s}(r) - \Pi_{r \cap \neg s}((s \times \Pi_{r \cap s}(r)) - r)$$



Todos os porcos



Todas as combinações
possíveis porco-espécie



Todas as combinações
reais porco-espécie

Divisão

- $\Pi_{r \cap \neg s}(r) - \Pi_{r \cap \neg s}((s \times \Pi_{r \cap \neg s}(r)) - r)$
 - Tradução direta para SQL requer demasiados SELECTs encadeados (por causa das projeções que são necessárias para realizar as diferenças)
 - Mas a lógica da dupla negativa pode ser implementada de forma mais direta com os operadores disponíveis em SQL
- Porcos que realizaram vendas de produtos de todas as espécies de animais: **não existe** nenhuma espécie para a qual eles **não** realizaram vendas

Divisão

- Que **porcos** realizaram vendas de produtos de **todas as espécies** de animais?

```
SELECT id
FROM pig p
WHERE NOT EXISTS (
    SELECT species
    FROM nonpig
    EXCEPT
    SELECT species
    FROM nonpig JOIN produce ON (id=producer) JOIN buys USING (code)
    WHERE seller=p.id
);
```

Quociente (porcos a retornar)

Divisor (espécies)

Dividendo
(espécies das vendas dos porcos)


Divisão


- Que **animais** produziram produtos em todos os dias de 2022?


Divisão

- Que **animais produziram produtos** em **todos os dias de 2022**?

```
SELECT id
  FROM nonpig n
 WHERE NOT EXISTS (
   SELECT t.day::DATE
     FROM generate_series('2022-1-1'::TIMESTAMP,
                          '2022-12-31'::TIMESTAMP, '1 day') AS t(day)
   EXCEPT
   SELECT date
     FROM produce
    WHERE producer=n.id
  ) ;
```

 **Quociente** (animais a retornar)

 **Divisor** (todos os dias != todos os dias em que houve produção)

 **Dividendo**
(dias em que os animais produziram)


Divisão


- Que **mercadores** **compraram ovos** de **todas as galinhas**?


Divisão

- Que **mercadores compraram ovos** de **todas as galinhas**?

```
SELECT ssn
  FROM merchant m
 WHERE NOT EXISTS (
   SELECT id
     FROM nonpig
    WHERE species='chicken'
  EXCEPT
   SELECT producer
     FROM buys b JOIN produce USING (code)
    WHERE b.ssn=m.ssn
 );
```

 **Quociente** (mercadores a retornar)

 **Divisor** (todas as galinhas)

 **Dividendo** (compras do mercador)
[não precisamos de filtrar galinhas
porque vamos calcular a diferença]

Divisão

- Em que **dias** **houve produção** de **todas as espécies**?

Divisão

- Em que **dias** **houve produção** de **todas as espécies**?

```
SELECT DISTINCT date
  FROM produce p1
 WHERE NOT EXISTS (
   SELECT species
     FROM nonpig
  EXCEPT
   SELECT species
     FROM nonpig JOIN produce p2 ON (p2.producer=id)
     WHERE p2.date=p1.date
  );
```

Quociente (datas a retornar) [não vale a pena criar uma série pois só queremos datas em que houve produção!]

Divisor (todas as espécies)

Dividendo (datas de produção das espécies)

NULLs

NULLs

- Na avaliação de expressões relacionais (com operadores de comparação) a ocorrência de NULL leva a um resultado UNKNOWN
 - Incluindo $\text{NULL} = \text{NULL}$
- Em expressões booleanas, NULL é “mais negativo” que TRUE mas “mais positivo” que FALSE:
 - $\text{NULL} \wedge \text{TRUE} = \text{NULL}$
 - $\text{NULL} \vee \text{TRUE} = \text{TRUE}$
 - $\text{NULL} \wedge \text{FALSE} = \text{FALSE}$
 - $\text{NULL} \vee \text{FALSE} = \text{NULL}$

NULLs

- Na computação de DISTINCT (o que inclui UNION, EXCEPT, INTERSECT e GROUP BY), NULL é tratado como qualquer outro valor da perspectiva de unicidade dos tuplos
 - $\text{NULL} = \text{NULL} \Rightarrow \text{TRUE}$
- Em operações numéricas algébricas, qualquer NULL leva a um resultado NULL
 - E.g. $2 + \text{NULL} = \text{NULL}$, $13 * \text{NULL} = \text{NULL}$

NULLs no WHERE & JOIN

- NULLs e UNKNOWNs no WHERE e JOIN são tratados como FALSE
 - Apenas linhas que avaliam TRUE para o conjunto de condições do WHERE são devolvidas, ou no caso de JOIN são emparelhadas
- Em operações de arrays:
 - IN e ANY devolvem TRUE se houver pelo menos um valor no array que retorna verdadeiro (mesmo que haja valores NULL)
 - ALL devolve NULL se houver pelo menos um valor NULL no array
- Em EXISTS, NULLs são praticamente irrelevantes

NULLs no WHERE & JOIN

- Qual é o porco mais velho?

Havendo 1+ dob NULL

```
SELECT id FROM pig WHERE EXTRACT(YEAR FROM AGE(dob)) >= ALL(  
    SELECT EXTRACT(YEAR FROM AGE(dob)) FROM pig);
```



No result.

```
WITH ages AS (SELECT id, EXTRACT(YEAR FROM AGE(dob)) FROM pig)  
SELECT id FROM ages WHERE age = (SELECT MAX(age) FROM ages);
```



id
101

- As duas queries são equivalentes, devolvem resultados diferentes se houver NULLs

NULLs na Agregação

- Em funções de agregação, NULLs são tratados como se não existissem
 - Incluindo COUNT(DISTINCT x), que só contará valores distintos não nulos de x
 - Agregadores numéricos como SUM(x), AVG(x) ignoram por completo os NULLs
 - Exceção COUNT(x) = COUNT(*) \Rightarrow conta sempre todas as linhas, mesmo que haja valores nulos de x
- NULLs em colunas do GROUP BY, por outro lado, levam a que seja criado um grupo com NULL

Lidar com NULLs

- Para filtrar valores nulos ou não nulos, não podemos usar operadores de comparação:
 - “ $x=NULL$ ” devolve UNKNOWN e não TRUE se x for NULL
 - “ $x\neq NULL$ ” devolve UNKNOWN e não FALSE se x for NULL
- Em vez disso, usamos os operadores IS [NOT] NULL
 - IS NULL: $NULL \Rightarrow TRUE$, $\neg NULL \Rightarrow FALSE$
 - IS NOT NULL: $\neg NULL \Rightarrow TRUE$, $NULL \Rightarrow FALSE$

Lidar com NULLs

- Já tínhamos usado IS NULL no LEFT JOIN para computar a diferença:

```
SELECT id FROM pig LEFT JOIN buys ON (id=seller) WHERE code IS NULL;
```

- Sempre que fazemos OUTER JOINS (LEFT, RIGHT ou FULL) temos de ter em conta que a tabela resultante vai ter NULLs
 - Podemos explorar esse aspecto (como no caso acima)
 - Mas regra geral temos de ter cuidado

Lidar com NULLs

- Podemos usar a função COALESCE para substituir NULLs por outro valor

```
SELECT id, COALESCE(name, 'anonymous') FROM pig;
```

- COALESCE devolve o primeiro dos valores listados que for não NULL
 - Podemos listar várias colunas (e.g. description, short description e depois um valor caso ambos sejam NULL)

Lidar com UNKNOWNs

- Há operadores de comparação próprios para a lógica de três valores (podemos aplicá-los sobre expressões relacionais):
 - IS TRUE: $\text{TRUE} \Rightarrow \text{TRUE}$, $\neg \text{TRUE} \Rightarrow \text{FALSE}$
 - IS NOT TRUE: $\neg \text{TRUE} \Rightarrow \text{TRUE}$, $\text{TRUE} \Rightarrow \text{FALSE}$
 - IS FALSE: $\text{FALSE} \Rightarrow \text{TRUE}$, $\neg \text{FALSE} \Rightarrow \text{FALSE}$
 - IS NOT FALSE: $\neg \text{FALSE} \Rightarrow \text{TRUE}$, $\text{FALSE} \Rightarrow \text{FALSE}$
 - IS UNKNOWN: $\text{UNKNOWN} \Rightarrow \text{TRUE}$, $\neg \text{UNKNOWN} \Rightarrow \text{FALSE}$
 - IS NOT UNKNOWN: $\neg \text{UNKNOWN} \Rightarrow \text{TRUE}$, $\text{UNKNOWN} \Rightarrow \text{FALSE}$

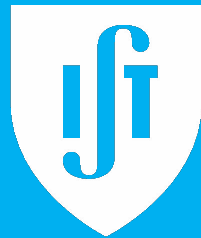
Lidar com UNKNOWNs

- Se havia falhas no registo de datas na base de dados antes de 1990:
 - Que porcos são maiores de idade?

```
SELECT id
FROM pig
WHERE (EXTRACT(YEAR FROM AGE(dob)) >= 18) IS NOT FALSE;
```

- Qual a quantidade total de produtos produzidos antes de 1990?

```
SELECT SUM(amount)
FROM produce
WHERE (date > '1990-01-01') IS NOT TRUE;
```

TÉCNICO LISBOA