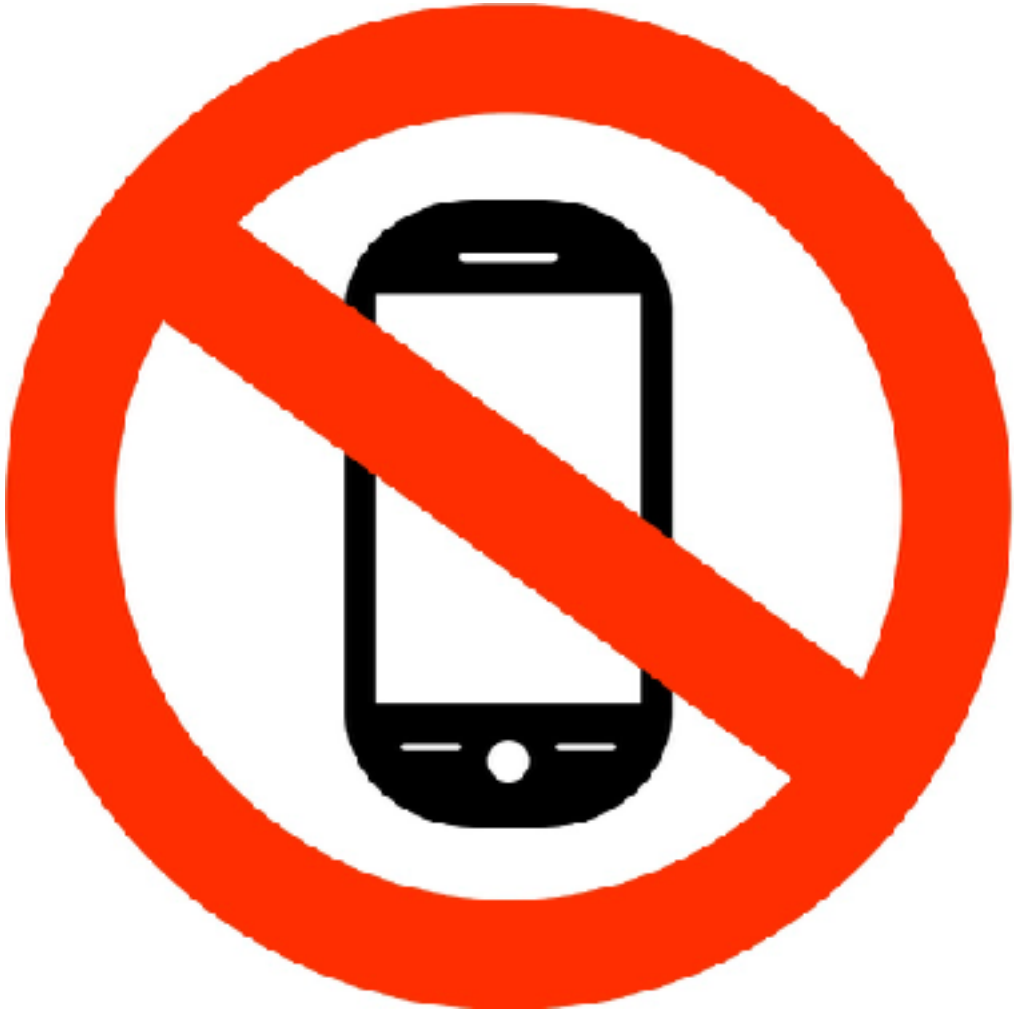
TÉCNICO LISBOA

Bases de Dados

Aula 09: SQL (cont.)

Prof. Paulo Carreira



Sumário

- Direct Aggregation
- Partitioned Aggregation
- Null values

Direct Aggregation

Direct Aggregation

```
select F1(C1),…,Fk(Cn)
from table
where condition
```

- Applies the functions to the values where the ***condition*** is true
- **F_1, …, F_k** are Aggregation Functions
- **C_1, …, C_n** are columns of ***table***

Aggregate Functions

- `COUNT ([DISTINCT] A)`
  - The count of (distinct) values on column A
- `SUM ([DISTINCT] A)`
  - The sum of (distinct) values on column A
- `AVG ([DISTINCT] A)`
  - The average of (distinct) values on column A
- `MAX (A)`
  - The maximum value on column A
- `MIN (A)`
  - The minimum value on column A

Aggregate Functions

- Find the number of customers in the bank

```
select count(*)
from customer;
```

```
 count
-------
    15
```

- Find the number of depositors in the bank

```
select count(distinct customer_name)
from depositor;
```

```
 count
-------
     7
```

- Find the average account balance at the 'Central' branch

```
select avg(balance)
from account
where branch_name = 'Central'
```

```
   avg
----------
 650.00
```

Aggregate Functions

- Find the average account balance, and the sum of balances at the branches 'Central' or 'Uptown'

```
select avg(balance), sum(balance)
from account
where branch_name = 'Central'
   or branch_name = 'Uptown'
```

```
  avg    |  sum
--------+---------
 725.00 | 2900.00
```

Partitioned Aggregation

Partitioned Aggregation

Find the number of customers per city

Counting customers

```
select count(*)
from customer
```

Partitioned per city

```
group by customer_city
```

Finding Partitions

```
select *  from customer
```

```
customer_name | customer_street | customer_city
---------------+-----------------+---------------
Adams          | Main
Brown          | Main
Cook           | Main Street     | Lisbon
Davis          | Chur
Evans          | Fore
Flores         | Stat
Gonzalez       | Sunn
Iacocca        | Spri
Johnson        | New
King           | Gard
Lopez          | Gran
Martin         | Roya
Nguyen         | Scho
Oliver         | Firs
Parker         | Hope
```

```
select *  from customer order by customer_city
```

customer_name	customer_street	customer_city
King	Garden Street	Aveiro
Flores	Station Street	Braga
Martin	Royal Street	Braga
Johnson	New Street	Cascais
Nguyen	School Street	Castelo Branco
Iacocca	Spring Steet	Coimbra
Evans	Forest Street	Coimbra
Gonzalez	Sunny Street	Faro
Adams	Main Street	Lisbon
Cook	Main Street	Lisbon
Davis	Church Street	Oporto
Brown	Main Street	Oporto
Oliver	First Stret	Oporto
Parker	Hope Street	Oporto
Lopez	Grand Street	Vila Real

Partitioned Aggregation: Step-by-step

```
select *
from customer
order by customer_city
```

customer_name	customer_street	customer_city
King	Garden Street	Aveiro
Flores	Station Street	Braga
Martin	Royal Street	Braga
Johnson	New Street	Cascais
Nguyen	School Street	Castelo Branco
Iacocca	Spring Steet	Coimbra
Evans	Forest Street	Coimbra
Gonzalez	Sunny Street	Faro
Adams	Main Street	Lisbon
Cook	Main Street	Lisbon
Davis	Church Street	Oporto
Brown	Main Street	Oporto
Oliver	First Stret	Oporto
Parker	Hope Street	Oporto
Lopez	Grand Street	Vila Real

```
select count(*)
from customer
```

count
15

```
select count(*)
from customer
group by customer_city
```

count
1
2
1
1
2
1
2
4
5

```
select customer_city, count(*)
from customer
group by customer_city
```

customer_city	count
Aveiro	1
Braga	2
Cascais	1
Castelo Branco	1
Coimbra	2
Faro	1
Lisbon	2
Oporto	4
Vila Real	1

The **group by** clause

- A select statement with a **group by** clause has the form:

(4) `select` $P_1,\ldots,P_m$, $F_1(A_1),\ldots,F_k(A_n)$

(1) `from` *table*

(2) `where` *condition*

(3) `group by` $P_1,\ldots,P_m$

Only columns specified in the **group by** clause below!

- Where:
  - **F_1, …, F_k** are Aggregation Functions
  - **A_1, …, A_n** are the aggregated columns of ***table***
  - **P_1, …, P_m** are partitioning columns (of ***table***)

Note that P1, …, Pm should not (in principle) be unique; in other words, the table is expected to have duplicates combinations of values for the columns P1, …, Pm.

Partitioned Aggregation: Example

- Find the name of customers per city

↑↑↑ This is not an aggregate query (why?)

```
select customer_name, count(*)
from customer
group by customer_city;
```

Selecting an **attribute** that is **not partitioned** is an error!

$$\pi_{customer\_name, count} \, {}_{customer\_city}G_{count()}(customer))$$

↑↑↑ Projects an attribute that does not exist

Partitioning by an unique column

- What happens when the partitioning is made through a filed with all distinct values?

```
select customer_name, count(*)
from customer
group by customer_name;
```

```
 customer_name | count
---------------+-------
 Oliver        |     1
 Iacocca       |     1
 Parker        |     1
 Davis         |     1
 Lopez         |     1
 Martin        |     1
 Adams         |     1
 Brown         |     1
 Gonzalez      |     1
 Evans         |     1
 King          |     1
 Nguyen        |     1
 Cook          |     1
 Flores        |     1
 Johnson       |     1
```

Each group (partition) will have only one row!

Aggregate Filtering

- Find the names of all branches where the average account balance is above 750€

```
select branch_name, avg(balance)
from account
where avg(balance) > 750;
group by branch_name
```

```
select *
from (
      select branch_name, avg(balance)
      from account
      group by branch_name
)
where balance > 750;
```

Aggregate Filtering

- Find the names of all branches where the average account balance is above than 700€

```
select branch_name, avg(balance)
from account
group by branch_name
having avg(balance) > 750;
```

```
 branch_name |   avg
-------------+-----------
 Uptown      | 800.00
 Round Hill  | 800.00
```

Note: predicates in the **having** clause are applied <u>after the formation of groups</u> whereas predicates in the **where** clause are applied <u>before forming groups!</u>

Example

What are the branches with at least two accounts?

```
select branch_name, count(*)
from account
group by branch_name
having count(*) >= 2;
```

```
 branch_name | count
-------------+-------
 Central     |     2
 Uptown      |     2
 Downtown    |     2
```

```
select branch_name
from account
group by branch_name
having count(*) >= 2;
```

```
 branch_name
--------------------
 Central
 Uptown
 Downtown
```

The **having** clause

- A select statement with a group by clause has the form:

```
(5) select G1,…,Gm, F1(A1),…,Fk(An)
(1) from table
(2) where condition
(3) group by P1,…,Pm
(4) having aggregate_condition
```

- Where:
  - the ***aggregate_condition*** is a condition with aggregate functions

Null Values

NULL

- Valor especial adicionado ao Domínio de todas as colunas
- Qualquer coluna é *nullable* por omissão (a não ser que se especifique a restrição NOT NULL)
- O que significa o valor NULL?
  - Valor desconhecido
  - Valor não preenchido
  - Valor não aplicável

Null values

- It is possible for tuples to have the **null** value
- The result of any arithmetic expression involving **null** is: **null**
- Any comparison with **null** returns **unknown**
- The predicates **is null** and **is unknown** can be used to check for null values and unknown results

```
select loan_number
from loan
where amount is null
```

```
select loan_number
from loan
where amount > 1000 is unknown
```

Note: The result of **where** condition is treated as **false** if it evaluates to **unknown**

Null and Aggregates

- The statement

```
select sum(amount) from loan
```

  ignores null amounts!

- Result is not null if there is <u>at least one</u> non-null amount
- All aggregate operations except **count**(*) ignore tuples with **null** values on the aggregated attributes.

Null and Aggregates

```
create table t(x varchar(10));
insert into t values(null);
insert into t values('Hello');
```

```
select * from t;
```

```
   x
-------

 Hello
```

```
select count(x) from t;
```

```
 count
-------
     1
```

```
select count(*) from t;
```

```
 count
-------
     2
```

```
select count(1) from t;
```

```
 count
-------
     2
```

NULL

```
create table Employee(
    name varchar(50) not null,
    age numeric(3),
    dept varchar(20),
);
```

Problems with NULL

```
select *
from Employee
```

Name	Age	Dept
Cajó	20	Finance
Quim	30	Finance
xico	NULL	Chemistry
ze	40	NULL
zico	50	NULL

NULL

- Como determinar os empregados que não indicaram a sua idade

```
select *
from Employee
where age = null
```

```
select *
from Employee
where age <> null
```

```
select *
from Employee
where age is null
```

Problems with NULL

Name	Age	Dept
Cajó	20	Finance
Quim	30	Finance
Xico	NULL	Chemistry
Zé	40	NULL
Zico	50	NULL

- What is the number of distinct Departments

```
select COUNT(distinct dept)
from Employee
```

2

Problems with NULL

Name	Age	Dept
Cajó	20	Finance
Quim	30	Finance
Xico	NULL	Chemistry
Zé	40	NULL
Zico	50	NULL

- What is the average Age

```
select AVG(age)
from Employee
```

35

Problems with NULL

- What is the average age per Department?

```
select dept, AVG(age)
from Employee
group by dept
```

Dept	Age
NULL	45
Chemistry	NULL
Finance	25

Problems with NULL

- What is the Department with the highest average age?

```
select dept, avg(age)
from Employee
group by dept
having avg(age) >= all (
 select avg(age)
    from Employee
    group by dept);
```

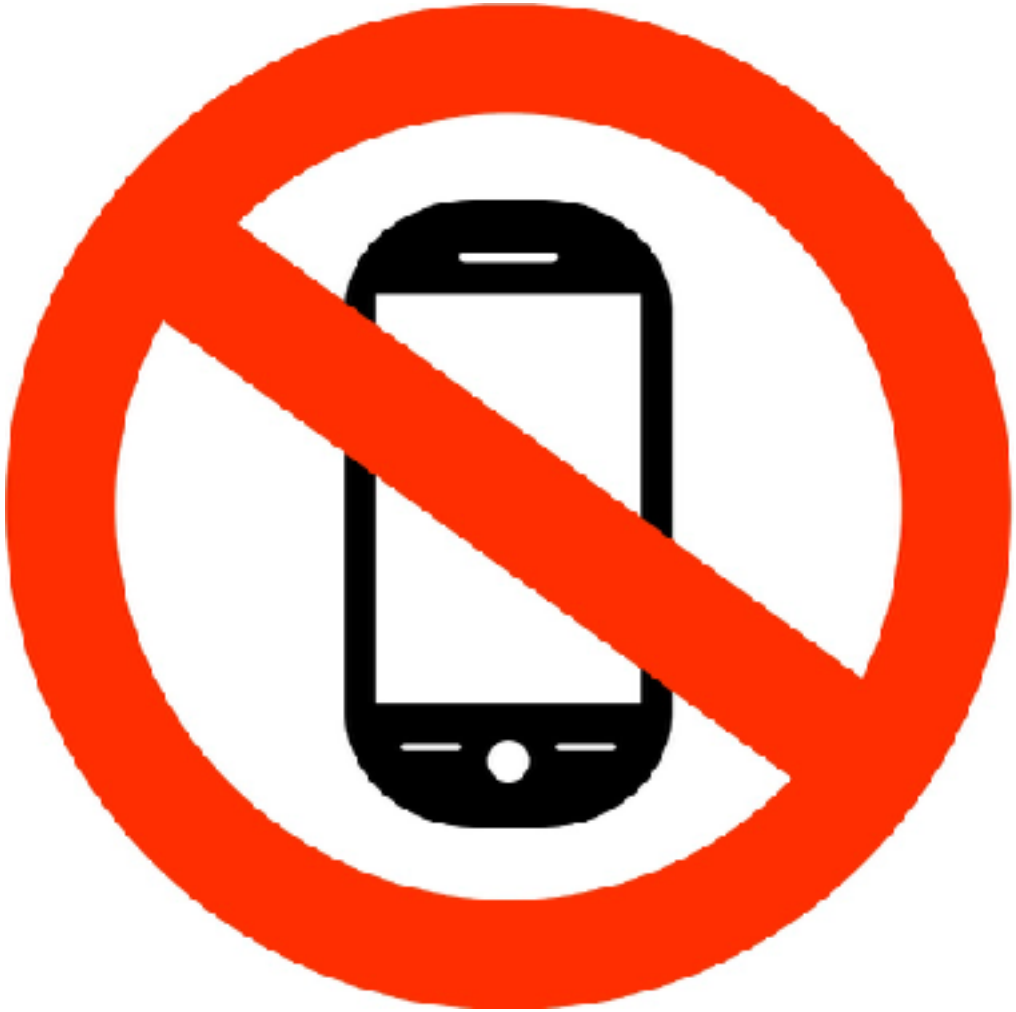
Age	Dept

TÉCNICO LISBOA

Bases de Dados

Aula 10: SQL (cont.)

Prof. Paulo Carreira



Sumário

- Nested Queries
- The operators IN and NOT IN
- Relational set comparisons with ALL and SOME
- Uses of WHERE >= ALL and HAVING >= ALL
- Correlated queries
- The operators EXISTS and UNIQUE

Cartesian Product

The Cartesian product

- The from clause lists the r
  the query
- Example: find the Cartesia

borrower × I

We are selecting all combinations. There is no 'where' clause (no filtering).

```
select *
from depositor d, account a
```

customer_name	account_number	account_number	branch_name	balance
Johnson	A-101	A-101	Downtown	500.0000
Johnson	A-101	A-215	Metro	600.0000
Johnson	A-101	A-102	Uptown	700.0000
Johnson	A-101	A-305	Round Hill	800.0000
Johnson	A-101	A-201	Uptown	900.0000
Johnson	A-101	A-222	Central	550.0000
Johnson	A-101	A-217	University	650.0000
Johnson	A-101	A-333	Central	750.0000
Johnson	A-101	A-444	Downtown	850.0000
Brown	A-215	A-101	Downtown	500.0000

Cartesian product with filter

- What happens when we filter a Cartesian product?

table variables

```
select *
from depositor d, account a
where d.account_number = a.account_number;
```

```
customer_name | account_number | account_number | branch_name | balance
--------------+----------------+----------------+-------------+----------
 Johnson      | A-101          | A-101          | Downtown    | 500.0000
 Brown        | A-215          | A-215          | Metro       | 600.0000
 Coc                                                            000
 Coc                                                            000
 Flo                                                            000
 Joh                                                            000
 Iacocca      | A-217          | A-217          | University  | 650.0000
 Evans        | A-222          | A-222          | Central     | 550.0000
 Oliver       | A-333          | A-333          | Central     | 750.0000
 Brown        | A-444          | A-444          | Downtown    | 850.0000
(10 rows)
```

```
select *
from depositor natural join account a;
```

Example II

- Find the names of the cities of the customers with accounts having more than 750 € in balance

```
select customer_city
from account a, depositor d, customer c
where a.account_number = d.account_number
and c.customer_name = d.customer_name
and balance > 750;
```

```
customer_city
---------------
 Oporto
 Braga
 Cascais
```

Nested Queries

(Sub-consultas ou Sub-Interrogações)

Sub-Interrogações

- Um comando `SELECT` pode conter outros comandos `SELECT`
- Podem aparecer nas cláusulas:

  `FROM`
  `WHERE`
  `HAVING` (a ver mais adiante)
- Aplicações típicas
  - ocorrência num conjunto
  - comparação de conjuntos
  - número de elementos num conjunto

Exemplo com FROM

Quais os nomes dos clientes com contas com mais de 750 euros?

```
select C.customer_name
from (select *
      from depositor
        natural join account
      where balance > 750) as C
```

Exemplo com FROM (1/2)

Quais os nomes dos clientes têm uma Conta ou num balcão de Lisboa ou do Porto?

```
select customer_name
from depositor
     natural inner join account
     natural inner join branch
where branch_city = 'Lisbon' or
  branch_city = 'Oporto';
```

```
customer_name
---------------
Johnson
Cook
Brown
```

Exemplo com FROM (2/2)

Quais os nomes dos clientes têm uma Conta ou num balcão de Lisboa ou do Porto?

```
select customer_name
from depositor
     natural inner join account
     natural inner join (
        select *
        from branch
        where branch_city = 'Lisbon'
        or branch_city = 'Oporto') B;
```

Exemplo com IN

Quais as cidades de todos os clientes que têm empréstimo mas não têm conta

- O operador `IN` permite testar se um valor pertence a um conjunto de elementos

```
select customer_city
from customer
where customer_name in (
    select customer_name from borrower)
  and customer_name not in (
    select customer_name from depositor)
```

Exemplo com IN

Quais as cidades de todos os clientes que têm empréstimo mas não têm nenhuma conta com mais do que 1000 €

```
select customer_city
from customer
where customer_name in (
    select customer_name from borrower)
and customer_name not in (
    select customer_name
    from depositor natural inner join account
    where balance > 1000)
```

Múltiplas sub-interrogações

Quais os nomes dos clientes têm uma Conta ou num balcão de Lisboa ou do Porto?

```
select customer_name
from Depositor as D
where D.account_number in (
  select account_number
  from Account as A
  where A.branch_name in (
    select branch_name
    from branch
    where branch_city = 'Lisbon'
    or branch_city = ‘Oporto'));
```

Comparação de conjuntos com ALL

Quais as contas com mais dinheiro do **que todas as contas** do balcão Central?

```
select A.account_number
from account A
where A.balance >= all (
    select B.balance
    from account B
    where B.branch_name = 'Central')
```

- E se não existir nenhuma conta no balcão Central?
  - A comparação com `ALL` retorna verdadeiro

Comparação de conjuntos com SOME

Quais as contas com mais dinheiro do que **pelo menos uma** conta do balcão Central?

```
select A.account_number
from account A
where A.balance > some (
    select B.balance
    from account B
    where B.branch_name = 'Central')
```

- E se não existir nenhuma conta no balcão Central?
  - A comparação com `SOME` retorna falso

ALL & SOME

- Fórmula geral:
  - *op* **ALL**
  - *op* **SOME** (ou *op* **ANY**)
- **IN** equivalente a **= ANY**
- **NOT IN** equivalente a **<> ALL**

Exemplo de escolha do máximo

Quais as contas com maior saldo?

```
select A.account_number
from account A
where A.balance >= all (
    select B.balance
    from account B)
```

- E se escrevêssemos `where A.balance > all (…` ?
  - O resultado seria vazio

Determinação do elemento distintivo

Determinar o maior absoluto

Qual o maior montante?

```
select max(amount)
from loan
```

```
amount
-----------
9000.0000
```

Determinar informação associada ao maior elemento

Qual o empréstimo com o maior montante?

- Solução 1

```
select loan_number, amount
from loan
where amount >= all (
  select amount
  from loan
)
```

```
 loan_number |  amount
-------------+-----------
 L-21        | 9000.0000
```

- Solução 2

```
select loan_number, amount
from loan
where amount = (
  select max(amount)
  from loan
)
```

```
 loan_number |  amount
-------------+-----------
 L-21        | 9000.0000
```

O elemento maior que todos os outros

Qual o empréstimo com maior montante?

- Solução 3

```
select loan_number, amount
from loan natural inner join (
  select max(amount) as amount
  from loan
) A;
```

```
 loan_number |  amount
-------------+-----------
 L-21        | 9000.0000
```

O elemento maior que todos os outros

O balcão com maior montante total das suas contas

```
select brach_name, max(balance)
from account
group by branch_name;
```

```
 branch_name |     sum
-------------+-----------
 Central     | 1300.0000
 Uptown      | 1600.0000
 Round Hill  |  800.0000
 Downtown    | 1350.0000
 University  |  650.0000
 Metro       |  600.0000
(6 rows)
```

Determinar o grupo com o elemento maior (parte 1)

- O balcão com maior montante total das suas contas

```
select branch_name, sum(balance)
from account
group by branch_name
having sum(balance) >= all (
    select sum(balance)
    from account
    group by branch_name);
```

Determinar o grupo com o elemento maior (parte 2)

- O balcão com mais contas

```
select branch_name, count(*)
from account
group by branch_name
having count(*) >= all (
    select count(*)
    from account
    group by branch_name);
```

TÉCNICO LISBOA

Bases de Dados

Aula 11: SQL (cont.)

Prof. Paulo Carreira

Correlated queries
(Interrogações co-relacionadas)

Exemplo com EXISTS

Quais os nomes dos clientes têm uma Conta num balcão de Lisboa ou do Porto?

```
select D.customer_name
from depositor as D
    natural inner join account as A
where exists (
  select *
  from branch B
  where branch_city in ('Lisbon', 'Oporto')
    and B.branch_name = D.branch_name);
```

- **EXISTS** permite testar se um conjunto não é vazio
- A sub-interrogação usa dados da tabela da interrogação principal

Exemplo com UNIQUE

Quais os nomes dos clientes apenas uma Conta?

```
select C.customer_name
from customer as C
where unique (
    select D.account_number
    from Depositor D
    where D.customer_name = C.customer_name)
```

- O operador **unique** quando aplicado a um conjunto (sub-interrogação), devolve true se não houver duplicados na resposta da sub-interrogação
  - Também retorna **true** se a resposta for vazia

Divisão

Divisão em SQL

- Operador DIVIDE
- Dupla negação

Divisão com EXCEPT

Quais são os nomes dos depositantes que abriram conta em todos balcões?

```
select customer_name
from depositor d
where not exists(
 select branch_name
 from branch
 except
 select branch_name
 from (account
   natural join depositor) b
 where b.customer_name = x
)
```

Todos os balcões

Todos os balcões do cliente *x*

Todos os balcões onde o cliente X NÃO tem conta

Divisão com EXCEPT

Quais são os nomes dos depositantes que abriram conta em todos balcões?

```
select customer_name
from depositor d
where not exists(
 select branch_name
 from branch
 except
  select branch_name
  from (account
    natural join depositor) b
  where b.customer_name =
       d.customer_name
)
```

Divisão com NOT EXISTS

Q9: Quais os nomes dos marinheiros que reservaram todos os barcos?

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (
      SELECT B.bid
      FROM Boats B
      WHERE NOT EXISTS (
            SELECT R. bid
            FROM Reserves R
            WHERE R.bid = B.bid
              AND R.sid = S.sid ))
```

- Para cada marinheiro S, verificamos se não existe nenhum barco que não tenha sido reservado por esse marinheiro.

Recursive queries

The `with` Clause

- The WITH clause provides a way of defining a temporary table whose definition is available only to the query in which the WITH clause occurs
- Example: find all accounts with the maximum balance

```
with max_balance(value) AS
  select max(balance)
  from account
select account_number
from account, max_balance
where account.balance =
  max_balance.value
```