TÉCNICO LISBOA

Sistemas de Informação e Bases de Dados

Class 09: SQL (cont.)

Prof. Paulo Carreira

Class Outline

- Direct Aggregation
- Partitioned Aggregation
- Nested Queries
- **NULL** values

SQL Block Recap

Query Block Structure

```
SELECT <column(s)>
FROM <table(s)>
WHERE <row-condition(s)>
GROUP BY <column(s)
HAVING <agg-condition(s)>
ORDER BY <column(s)>
```

Direct Aggregation

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
```

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 average balance of accounts, 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

Partitioned per city

```
SELECT COUNT(*)
FROM customer
```

```
GROUP BY customer_city
```

Finding Partitions

```
SELECT *  FROM customer
```

```
customer_name | customer_street | customer_city
--------------+-----------------+---------------
Adams         | Main Street     | Lisbon
Brown         | Main
Cook          | Main
Davis         | Church Street   | Oporto
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
1

```
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 <u>name of</u> customers per city

↑↑↑ This <u>is not</u> an aggregate query (why?)🤔

```
SELECT customer_name, COUNT(*)
FROM customer
GROUP BY customer_city;
```

↑↑↑ Projects an attribute that does not exist

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

Partitioning by an column with unique values

What happens when the partitioning is made on a filed that only has 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!

⚠ We are grouping by an attribute (or attibutes) that are a candidate key

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 750€

```
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: Aggregate Filtering

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** $G_1,\ldots,G_m$, $F_1(A_1),\ldots,F_k(A_n)$
(1) **FROM** *table*
(2) **WHERE** *condition*
(3) **GROUP BY** $P_1,\ldots,P_m$
(4) **HAVING** *aggregate_condition*

where:

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

Nested Queries
(or Sub-Queries)

Nested Queries

- Sub-SELECTs can appear in the clauses

  FROM

  WHERE

  HAVING (shown later)

- A **SELECT** block can include other **SELECT** blocks (i.e., sub-SELECTs)

- Typical usage:
  - Test the occurrence of a value in a set
  - Compare sets
  - Count the number of elements in a set

Nesting in **FROM** clause

What is the name of the clients with more that 750 euro on their accounts?

```
SELECT *
FROM depositor NATURAL JOIN account
WHERE balance > 750
```

```
SELECT C.customer_name
FROM (SELECT *
      FROM depositor D JOIN account A
      ON D.account_number = A.account_number
      WHERE balance > 750) AS C
```

Nesting in **JOIN**s (1/2)

Which Clients have an account on a branch in Lisbon or Oporto?

```
SELECT customer_name
FROM depositor d
    NATURAL JOIN account
    NATURAL JOIN branch
WHERE branch_city = 'Lisbon' OR branch_city =
  'Oporto';
```

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

Nesting in **JOIN**s (2/2)

Which Clients have an account on a branch in Lisbon or Oporto?

```
SELECT customer_name
FROM depositor AS d
    NATURAL JOIN account
    NATURAL JOIN (
         SELECT *
         FROM branch
         WHERE branch_city = 'Lisbon'
         OR branch_city = 'Oporto') AS b;
```

IN Operator

value **IN** (<set>)

The **IN** operator can be used to test if a value is in a set of values

```
SELECT customer
FROM customer
WHERE customer_city IN
  ('Lisbon','Oporto', 'Faro')
```

Nesting with the **IN** operator

Who are the clients that have a Loan but that do not have an Account?

```
SELECT customer_name
FROM customer
WHERE customer_name IN (
    SELECT customer_name FROM borrower)
  AND customer_name NOT IN (
    SELECT customer_name FROM depositor)
```

Example with **IN**

What are the cities of all clients that have a Loan but do not have any Account with more than 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 d join account a
       on d.account_number = a.account_number
    WHERE balance > 1000)
```

Multiple nested sub-queries

*What are the **names** of the Clients that **have an Account on** a branch **in** Lisbon or Oporto?*

```
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'));
```

Operators **ALL** and **ANY**

The operators **ALL** and **ANY** extend relational operators to work with sets

<*value*> *relop* **ALL** (**<*set*>**)

Tests if ***value*** can be related to ALL elements of the set using the relational operator *relop*

<value> *relop* **ANY** (**<*set*>**)

Tests if ***value*** can be related to SOME element of the set using relational operator *relop*

- **IN** is the equivalent to **= ANY**
- **NOT IN** is equivalent to **<> ALL**

Comparing sets with the **ALL** operator

*Which Accounts have more money **than all the Accounts of the** branch Central?*

```
SELECT A.account_number
FROM account A
WHERE A.balance >= ALL (
    SELECT B.balance
    FROM account B
    WHERE B.branch_name = 'Central')
```

What will the result be if there are <u>no Accounts</u> in the branch Central?

⚠ The comparison with **ALL** returns **True**

Comparing sets with the **ANY** operator

*Which Accounts have more money **than some Account of the** branch Central?*

```
SELECT A.account_number
FROM account A
WHERE A.balance > ANY (
    SELECT B.balance
    FROM account B
    WHERE B.branch_name = 'Central')
```

What will the result be if there are <u>no Accounts</u> in the branch Central?

⚠ Comparing using **ANY** will return False

Example: Finding the maximum value

Which Accounts have the greatest balance? (the balance greater than any other balance)

```
SELECT A.account_number
FROM account A
WHERE A.balance >= ALL (
    SELECT B.balance
    FROM account B)
```

What if we write `WHERE A.balance > ALL (…` ?

⚠ The result would be empty

NULL Values

NULL

- Table cells can have NULL values (often rendered as blank)
- Special value added to the domain of every column The meaning of NULL is ambiguous
  - Unknown
  - Not applicable
  - Unfilled

Every column in SQL is nullable by default (to revert this, the constraint NOT NULL must is be used)

Sources of NULL values

Finding records with NULL

- All employees

```
SELECT *
FROM employee
```

```
eid  | name     | birthdate
-----+----------+------------
   1 | Alice    | 1995-10-10
   2 | Bob      | 1996-03-02
   3 | Caroline | 1996-04-04
   5 | Eduard   | 1994-10-03
   4 | Daniel   | 1998-04-04
   6 | Florence |
```

- All employees with whose birthdate was not recorded

```
SELECT *
FROM employee
WHERE birthdate = null
```

```
eid  | employee | birthdate
-----+----------+------------
```

⚠What is failing here?

⚠ At least one employee does not have any value for birthdate

NULL Arithmetic

- Any arithmetic expression involving **NULL** is results in **NULL** (1+null, 10*null, etc…)
- Any logic expression involving **NULL** will evaluate to **NULL** (null **and** true, null or false null, etc…)
- Any relational expression with **NULL** returns **unknown** (null = null, null <> null, etc…)

- The predicates **is null** can be used to check for null values

```
SELECT *
FROM employee
WHERE birthdate is null
```

⚠ The result of **WHERE** condition is treated as **false** if it evaluates to **unknown**

NULL

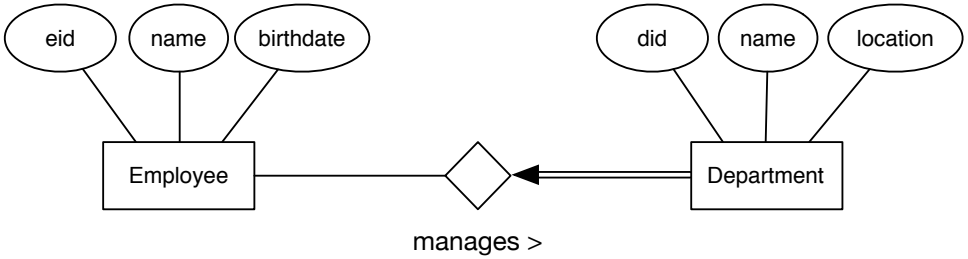
Which employees did not enter a birthdate?

```
SELECT *
FROM employee
WHERE birth_date = NULL
```

```
SELECT *
FROM employee
WHERE birth_date <> NULL
```

```
SELECT *
FROM employee
WHERE birth_date IS NULL
```

Left Outer Join




```
CREATE TABLE employee(
      eid INTEGER,
      name VARCAHR(80) NOT NULL,
      birthdate DATE,
      PRIMARY KEY(eid)
);
```

```
CREATE TABLE department(
      did INTEGER,
      name  VARCHAR(20) NOT NULL,
      location VARCHAR(40) NOT NULL,
      mid integer,
      PRIMARY KEY (did),
      FOREIGN KEY (mid) REFERNCES
      e
);
```

Employee

eid	name	birthdate
1	Alice	10/10/1995
2	Bob	03/02/1996
3	Caroline	04/04/1997
4	Daniel	03/04/1998
5	Eduard	10/03/1994

Department

did	name	location	mid
1	Finance	Buraca	1
2	Marketing	Damaia	2
3	Sales	Chelas	3

Left Outer Join

Get all information about employees (including departments that they manage, in case they do manage)

```
SELECT eid, e.name as employee, birthdate,
    d.name as manages
FROM employee e
    LEFT OUTER JOIN department d
      ON e.eid = d.mid;
```

⚠What happens to the employees that do not manage anything?

```
eid  | employee | birthdate  |  manages
-----+----------+------------+-----------
   1 | Alice    | 1995-10-10 | Finance
   2 | Bob      | 1996-03-02 | Marketing
   3 | Caroline | 1996-04-04 | Sales
   5 | Eduard   | 1994-10-03 |
   4 | Daniel   | 1998-04-04 |
   6 | Florence |            |
```

Inconsistent treatment of NULL values by SQL

Null and Aggregates

This aggregation statement ignores **NULL** amounts

```
SELECT SUM(amount) FROM loan
```

- Result will not be null if there is at least one non-null amount
- Null is treated by SUM as a neutral element instead of an absorbent element

⚠ SUM reverses the convention for null arithmetics!

NULL and Aggregates

```
CREATE TABLE t(x VARCHAR(10));
INSERT INTO t VALUES(NULL);
INSERT INTO t VALUEs('Hello');
```

```
SELECT * FROM t;
```

```
   x
-------

 Hello
```

⚠ Using COUNT(attr) without DISTINCT is typically an error!

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

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

- Ignores nulls

```
SELECT COUNT(*) FROM t;
```

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

- Counts all lines

```
SELECT COUNT(1) FROM t;
```

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

- Counts all lines

All aggregate operations except **count(*)** ignore tuples with **null** values on the aggregated attributes.

Problems with NULL

Employeex

Name	Age	Dept
Alice	24	Finance
Bob	23	Marketing
Caroline	23	Sales
Florence	NULL	NULL
Daniel	21	Sales
Eduard	25	NULL

What is the number of distinct Departments?

```
SELECT COUNT(DISTINCT dept)
FROM employee
```

3

⚠ NULL departments are ignored

Problems with NULL

What is the maximum age?

```
SELECT MAX(age)
FROM employeex
```

```
25
```

⚠ NULL departments are ignored

What is maximum of an empty table?

```
DELETE FROM employeex;

SELECT MAX(age)
FROM employeex;
```

```
<null>
```

Problems with NULL

What is the average Age?

Employeex

Name	Age	Dept
Alice	24	Finance
Bob	23	Marketing
Caroline	23	Sales
Florence	NULL	NULL
Daniel	21	Sales
Eduard	25	NULL

```
SELECT AVG(age)
FROM employee
```

23.2

⚠ AVG ignores NULLs

Problems with NULL

What is the average age per department?

```
SELECT dept, AVG(age)
FROM employeex
GROUP BY dept
```

Dept	Age
NULL	24
Finance	24
Marketing	23
Sales	22

GROUP BY creates a **NULL** group

⚠How many departments are there after all? 3 or 4?🙀

⚠ **GROUP BY** assumes that **`null=null`** is TRUE!🙀🙀

Problems with NULL

Who is the oldest employee?

```
SELECT name, age
FROM employeex
WHERE age >= ALL (
    SELECT age
    FROM employeex
);
```

```
SELECT name, age
FROM employeex
WHERE age >= (
    SELECT MAX(age)
    FROM employeex
);
```

name	age

⚠Empty

name	age
Eduard	25

⚠ The queries are equivalent but return different results

Replacing NULLs

*Using the **COALESCE** function*

```
SELECT name,
  COALESCE(age, 0) AS agex ,
  COALESCE(dept, 'None') AS depx
FROM employeex;
```

```
   name    | agex |    depx
----------+------+-----------
 Alice     |   24 | Finance
 Bob       |   23 | Marketing
 Caroline  |   23 | Sales
 Florence  |    0 | None
 Daniel    |   21 | Sales
 Eduard    |   25 | None
```

TÉCNICO LISBOA

Sistemas de Informação e Bases de Dados

Class 10: SQL (cont.)

Prof. Paulo Carreira

Bibliografia

Class outline

- 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
- Cross Product
- Division in SQL

Determining the distinctive element(s)

Determining the absolute maximum

*What is the **greatest** balance of an account?*

```
SELECT MAX(balance)
FROM account
```

```
max
-----------
900.0000
```

Determining the information associated to the distinctive element

*Find the account with the **greatest** balance?*

```
SELECT account_number, MAX(balance)
FROM account
GROUP BY account_number
```

We need to determine the greatest balance and then find the associated account

Multiple possible solutions exist!

Determining the information associated to the distinctive element

*Find the account with the **greatest** balance?*

```
SELECT account_number, balance
FROM account
WHERE balance >= ALL (
  SELECT balance
  FROM account
)
```

```
 account_number | balance
----------------+----------
 A-201          | 900.0000
```

```
SELECT account_number, balance
FROM account
WHERE balance = (
  SELECT MAX(balance)
  FROM account
)
```

```
 account_number | balance
----------------+----------
 A-201          | 900.0000
```

The element that is greater that every other

*Find the account with the **greatest** balance?*

Solution with JOIN and MAX

```
SELECT account_number, A.balance
FROM account A JOIN (
  SELECT MAX(balance) AS balance
  FROM account
) B
ON A.balance = B.balance;
```

```
 account_number | balance
----------------+----------
 A-201          | 900.0000
```

The element that is greater that every other

What is the branch with the greatest sum of account balances?

```
SELECT brach_name, SUM(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)
```

⚠ This query determines the greatest amount per branch (and *not* the branch with the greatest amount!)

Determining the group with the greatest element (Part 1)

*What is the Branch with the **greatest sum** of Account balances?*

```
SELECT branch_name, SUM(balance)
FROM account
GROUP BY branch_name
HAVING SUM(balance) >= ALL (
    SELECT SUM(balance)
    FROM account
    GROUP BY branch_name);
```

Determining the group with the greatest element (Part 2)

What is the Branch with more accounts?

```
SELECT branch_name, COUNT(*)
FROM account
GROUP BY branch_name
HAVING COUNT(*) >= ALL (
    SELECT COUNT(*)
    FROM account
    GROUP BY branch_name);
```

Correlated Queries

Correlation using **EXISTS**

What are the name of the Customers that have an Account on Branch located in Lisbon or Oporto?

```
SELECT D.customer_name
FROM depositor D JOIN account A
    on D.account_number = A.account_number
WHERE EXISTS (
   SELECT *
   FROM branch B
   WHERE branch_city IN ('Lisbon', 'Oporto')
    AND B.branch_name = A.branch_name);
```

- **EXISTS** allows to test that a set is **not** empty
- The sub-query uses data of the main query

Correlation using **UNIQUE**

What is name of the Clients that have only one Account?

```
SELECT C.customer_name
FROM customer as C
WHERE UNIQUE (
    SELECT D.account_number
    FROM Depositor D
    WHERE D.customer_name = C.customer_name)
```

- The **UNIQUE** operator returns `True` <u>if there are no duplicates</u> in the set (or the set defined by the sub-query)
- What is the result of **UNIQUE** if the set is empty?
  ⚠ Returns True

⚠ This standard operator does not work on all DBMS (e.g. no supported in Microsoft SQLServer)

Cross Join (or Cartesian Product)

The CROSS JOIN (or Cartesian product)

depositor × account

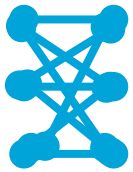
```
SELECT *
FROM depositor d, account a
```

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
Brown	A-215	A-215	Metro	600.0000
Brown	A-215	A-102	Uptown	700.0000

CROSS JOIN

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	Z



Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca

```
SELECT e.name, d.name
FROM Employee e, Department d
```

The FROM clause lists the tables involved in the query

e.ei	e.name	e.did	d.di	d.name	e.loc
1	Alice	X	X	Marketing	Damaia
2	Barbara	Y	X	Marketing	Damaia
3	Carlos	T	X	Marketing	Damaia
1	Alice	X	Y	Sales	Amadora
2	Barbara	Y	Y	Sales	Amadora
3	Carlos	T	Y	Sales	Amadora
3	Carlos	T	Z	Production	Buraca

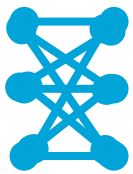
Employee X Department

Outputs **all combinations of rows,** irrespective of the contents

CROSS JOIN with a Filter

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	Z



Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca

```
SELECT e.name, d.name
FROM Employee e, Department d
WHERE e.did = d.did
```

Query has a filter

e.eid	e.name	did	d.name	e.loc
1	Alice	X	Marketing	Damaia
2	Barbara	Y	Sales	Amadora
3	Carlos	Z	Production	Buraca

Rows that **do not match** the filter **are dropped**

```
SELECT *
FROM Employee e JOIN Department d
        ON e.did = d.did
```

Cartesian product with a filter

- What happens when we filter a cross join?

```
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
 Cook          | A-102          | A-102          | Uptown      | 700.0000
 Cook          | A-101          | A-101          | Downtown    | 500.0000
 Flores        | A-305          | A-305          | Round Hill  | 800.0000
 Johnson       | A-201          | A-201          | Uptown      | 900.0000
 Iacocca       | A-217          | A-217          | University  | 650.0000
 Ev                                                            | 550.0000
 O                                                             | 750.0000
 Br                                                            | 850.0000
(1(
```

```
SELECT *
FROM account a JOIN depositor d
  ON a.account_number = d.account_number
```

Example

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
```

Cross join of 3 tables with two filters

Division

Division in SQL

Two ways of specifying division operation in SQL

- Using the **DIVIDE** operator (not always supported by RDBMSs)
- Using double negation (the typical way to specify division queries)

Division

E
A
B

×

D
□
△
○

=

E	D
A	□
A	△
A	○
B	□
B	△
B	○

÷

E
A
B

=

D
□
△
○

E
A
B

×

D
□
△
○

=

E	D
A	□
A	△
A	○
B	□
B	△
~~B~~	~~○~~

÷

E
A
B

=

D
□
△

Division with **EXCEPT**

Who are the Depositors that have Accounts on all branches?

```
SELECT DISTINCT customer_name
FROM depositor d
WHERE NOT EXISTS(
   SELECT branch_name
   FROM branch
  EXCEPT
  SELECT branch_name
  FROM (account a
    JOIN depositor d
      ON a.account_number = d.account_number) b
  WHERE b.customer_name = x
)
```

All branches

All branches of a given customer X

All branches where *X* does NOT have an account

Division with **EXCEPT**

Who are the Depositors that have Accounts on all branches?

```
SELECT DISTINCT customer_name
FROM depositor d
WHERE NOT EXISTS(
    SELECT branch_name
    FROM branch
    EXCEPT
    SELECT b.branch_name
    FROM (account a join depositor d
        ON a.account_number = d.account_number) b
    WHERE b.customer_name =  d.customer_name
)
```

Division with **EXCEPT**

Who are the Depositors that have Accounts on all branches of Lisbon?

```
SELECT DISTINCT customer_name
FROM depositor d
WHERE NOT EXISTS(
    SELECT branch_name
    FROM branch
    WHERE branch_city = 'Lisbon'
    EXCEPT
    SELECT b.branch_name
    FROM (account a JOIN depositor d
        ON a.account_number = d.account_number) b
    WHERE b.customer_name =  d.customer_name
);
```

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

Division with **EXCEPT**

Who are the Depositors that have Accounts on all branches of Freixo de Espada a Cinta?

```
SELECT DISTINCT customer_name
FROM depositor d
WHERE NOT EXISTS (
    SELECT branch_name
    FROM branch
    WHERE branch_city = 'Freixo de Espada a Cinta'
    EXCEPT
    SELECT b.branch_name
    FROM (account a JOIN depositor d
        ON a.account_number = d.account_number) b
    WHERE b.customer_name =  d.customer_name
);
```

⚠ Empty

```
customer_name
--------------
Oliver
Brown
Iacocca
Evans
Cook
Johnson
Flores
```

Division by an empty set results return the original set

Division with **COUNT**

Who are the Depositors that have Accounts on all branches of Lisbon?

```sql
SELECT customer_name
FROM depositor NATURAL JOIN account a
GROUP BY customer_name
HAVING COUNT(DISTINCT branch_name) = (
    SELECT COUNT(*)
    FROM branch
    WHERE branch_city='Lisbon');
```

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

⚠ By coincidence, 'Brown' has opened accounts in as many branches there are in Lisbon

Division with **COUNT**

Who are the Depositors that have Accounts on all branches of Freixo de Espada a Cinta?

```
SELECT customer_name
FROM depositor NATURAL JOIN account a
GROUP BY customer_name
HAVING COUNT(DISTINCT branch_name) = (
    SELECT count(*)
    FROM branch
    WHERE branch_city='Freixo de Espada a Cinta');
```

⚠ Always zero

```
customer_name
--------------
```

Empty result

⚠ There many ways to solve division, but most of them are WRONG! 🙀