SQL Programming Language

By: Mohamed Aziz Tousli

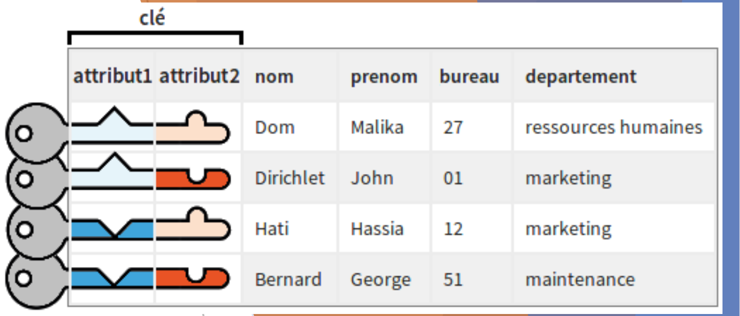
Relational Model

- **Relational model**: Used to represent structured data
  - **Relational data bases**: Structured data
  - **Non-relational data bases / Big data**: Unstructured data
- **RDBMS** = Relational Database Management System → **SQL** Programming Language
- **Relationship** = Table with data | Line = Object with same nature in a non-specific order

Relational model	Relational data bases
Relationship	Table
Tuple, n-Tuple, Record, Vector	Row
Attribute	Column
Diagram: Set of attributes	Set of columns
Field, Domain	Column Type

- PS: A relationship can’t have two identical tuples
- PS: Empty value = **NULL**

Keys

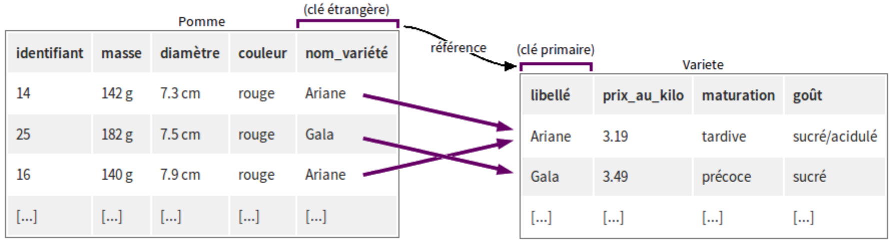


clé

attribut1	attribut2	nom	prenom	bureau	departement
		Dom	Malika	27	ressources humaines
		Dirichlet	John	01	marketing
		Hati	Hassia	12	marketing
		Bernard	George	51	maintenance

- A **Primary Key [PK]** is a minimum attribute group that determines a unique tuple
- A primary key / A candidate key must be:
  - Basic (Integer, short string…)
  - Logical (Can be assigned to identify an object)
  - ➔ Solution: **Artificial key**: Attribute that we add to the relationship that doesn't characterize the object

  PS: Can't be NULL



Pomme

(clé étrangère)

identifiant	masse	diamètre	couleur	nom_variété
14	142 g	7.3 cm	rouge	Ariane
25	182 g	7.5 cm	rouge	Gala
16	140 g	7.9 cm	rouge	Ariane
[...]	[...]	[...]	[...]	[...]

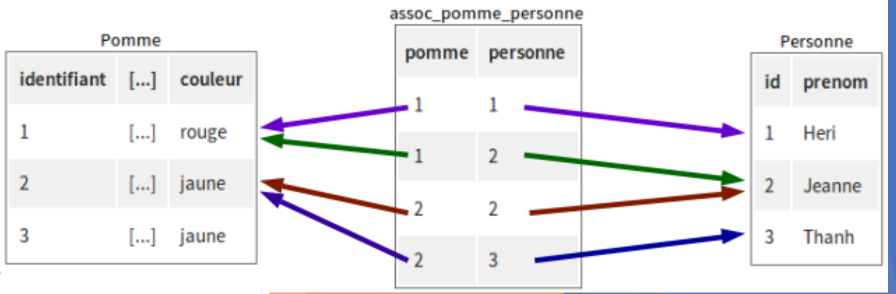
référence

(clé primaire)

Variete

libellé	prix_au_kilo	maturation	goût
Ariane	3.19	tardive	sucré/acidulé
Gala	3.49	précoce	sucré
[...]	[...]	[...]	[...]

- A **Foreign Key [FK]** in table A is **primary key** in table B. It is used to link relationships between them
- Avoid redundancy: It is not good to put all the data in one big table
  - How to divide a table? If an attribute A depends only on G (and G is not a candidate key), then it is possible to create a new relation which will contain the attributes A and G (G will be a candidate key for the new created relationship)
- Normalization: Remove redundancy!
- Cardinality of the link between table A and table B:
  - From 1 to many, from many to 1, from 1 to 1, from many to many
    - → **Association table**: The primary key is composed of at least two foreign keys



Pomme

identifiant	[...]	couleur
1	[...]	rouge
2	[...]	jaune
3	[...]	jaune

assoc_pomme_personne

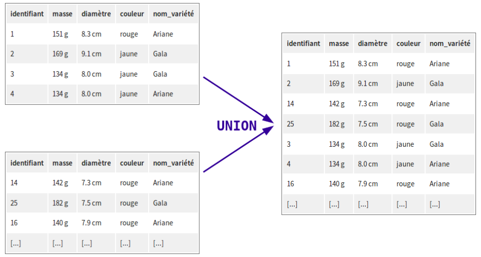
pomme	personne
1	1
1	2
2	2
2	3

Personne

id	prenom
1	Heri
2	Jeanne
3	Thanh

Relational Algebra

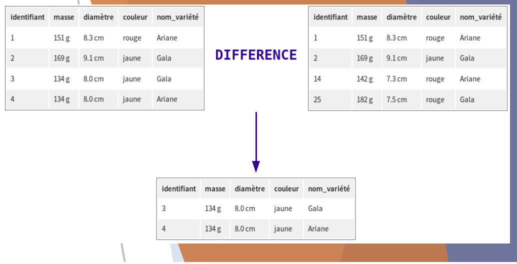
- **Relational model:** Used to manipulate structured data
- **Projection** is selecting the *columns* of a relationship
- **Restriction** is selecting the *rows* of a relationship, under a condition
- Set operators are operations that relate to 2 relationships of the same schem
- **Union (+)** of two relations R1 and R2 of the same schema produces a third re which contains the set of tuples of R1 and R2.
- **Difference (-)** between a relation R3 and R2 gives a relation R1 which contains all the tuples belong to R2
- **Intersection** between two relations R1 and R2 gives a third relation containing the tuples that are present in both R1 and R2
  - PS: A Intersection B = A Difference (A Difference B)
- **Cartesian product** between two relations R1 and R2 is composed of all the possible combinations between the tuples of R1 and the tuples of R2
- **Division** of a relation R1 by a relation R2 (knowing that R1 and R2 have at least one common attribute) gives a third relation R3 comprising all the attributes of R1 which do not belong to R2, and which contains all the tuples which, when joined to those of R2, always give a tuple of R1



identifiant	masse	diamètre	couleur	nom_variété
1	151 g	8.3 cm	rouge	Ariane
2	169 g	9.1 cm	jaune	Gala
3	134 g	8.0 cm	jaune	Gala
4	134 g	8.0 cm	jaune	Ariane

identifiant	masse	diamètre	couleur	nom_variété
14	142 g	7.3 cm	rouge	Ariane
25	182 g	7.5 cm	rouge	Gala
16	140 g	7.9 cm	rouge	Ariane
[...]	[...]	[...]	[...]	[...]

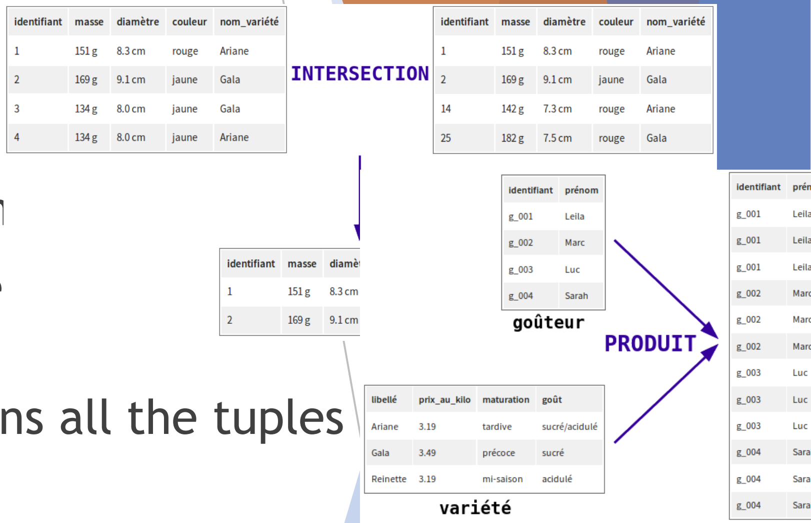
identifiant	masse	diamètre	couleur	nom_variété
1	151 g	8.3 cm	rouge	Ariane
2	169 g	9.1 cm	jaune	Gala
14	142 g	7.3 cm	rouge	Ariane
25	182 g	7.5 cm	rouge	Gala
3	134 g	8.0 cm	jaune	Gala
4	134 g	8.0 cm	jaune	Ariane
16	140 g	7.9 cm	rouge	Ariane
[...]	[...]	[...]	[...]	[...]



identifiant	masse	diamètre	couleur	nom_variété
1	151 g	8.3 cm	rouge	Ariane
2	169 g	9.1 cm	jaune	Gala
3	134 g	8.0 cm	jaune	Gala
4	134 g	8.0 cm	jaune	Ariane

identifiant	masse	diamètre	couleur	nom_variété
1	151 g	8.3 cm	rouge	Ariane
2	169 g	9.1 cm	jaune	Gala
14	142 g	7.3 cm	rouge	Ariane
25	182 g	7.5 cm	rouge	Gala

identifiant	masse	diamètre	couleur	nom_variété
3	134 g	8.0 cm	jaune	Gala
4	134 g	8.0 cm	jaune	Ariane



identifiant	masse	diamètre	couleur	nom_variété
1	151 g	8.3 cm	rouge	Ariane
2	169 g	9.1 cm	jaune	Gala
3	134 g	8.0 cm	jaune	Gala
4	134 g	8.0 cm	jaune	Ariane

identifiant	masse	diamètre	couleur	nom_variété
1	151 g	8.3 cm	rouge	Ariane
2	169 g	9.1 cm	jaune	Gala
14	142 g	7.3 cm	rouge	Ariane
25	182 g	7.5 cm	rouge	Gala

identifiant	masse	diamè
1	151 g	8.3 cm
2	169 g	9.1 cm

identifiant	prénom
g_001	Leila
g_002	Marc
g_003	Luc
g_004	Sarah

goûteur

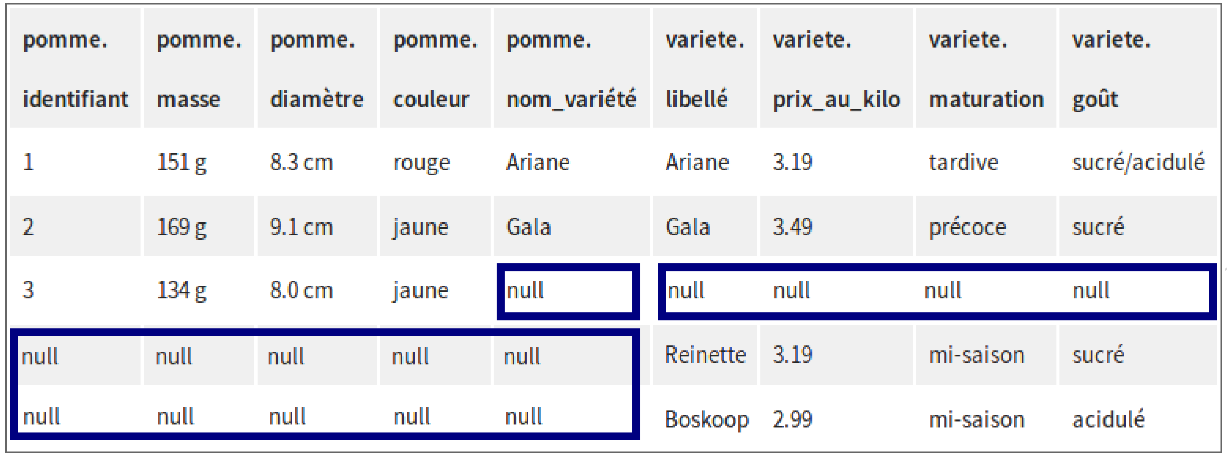
libellé	prix_au_kilo	maturation	goût
Ariane	3.19	tardive	sucré/acidulé
Gala	3.49	précoce	sucré
Reinette	3.19	mi-saison	acidulé

variété

identifiant	prén
g_001	Leila
g_001	Leila
g_001	Leila
g_002	Marc
g_002	Marc
g_002	Marc
g_003	Luc
g_003	Luc
g_003	Luc
g_004	Sara
g_004	Sara
g_004	Sara

Juncture

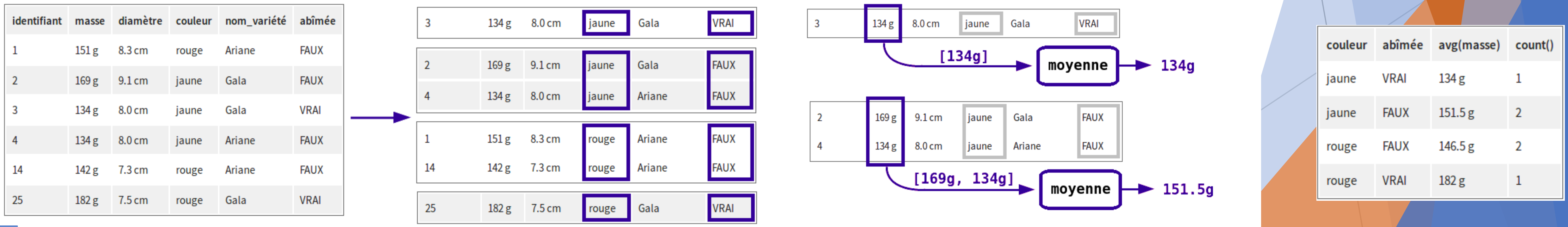
- **Juncture:** The juncture creates a large table that will contain the information of two tables
  - **Intern juncture:** Juncture under the condition foreignKey.table1 = primaryKey.table2
  - **Left outer juncture:** Juncture that will keep all the lines of the left table adding **null** values for the non correspondence of the right table
  - **Right outer juncture:** Juncture that will keep all the lines of the right table adding **null** values for the non correspondence of the left table
  - **Total juncture:** Left outer juncture + Right outer juncture
  - **Natural juncture:** Intern juncture without the need to specify the condition since it’s obvious
- **Juncture = Cartesian Product + Restriction**



pomme.	**pomme.**	**pomme.**	**pomme.**	**pomme.**	**variete.**	**variete.**	**variete.**	**variete.**
identifiant	**masse**	**diamètre**	**couleur**	**nom_variété**	**libellé**	**prix_au_kilo**	**maturation**	**goût**
1	151 g	8.3 cm	rouge	Ariane	Ariane	3.19	tardive	sucré/acidulé
2	169 g	9.1 cm	jaune	Gala	Gala	3.49	précoce	sucré
3	134 g	8.0 cm	jaune	null	null	null	null	null
null	null	null	null	null	Reinette	3.19	mi-saison	sucré
null	null	null	null	null	Boskoop	2.99	mi-saison	acidulé

Aggregation

- **Aggregation**: Calculate a result that relates to several rows of a table. We need:
  - A group of partitioning attributes: **aggregates**
    - Goal: Create groups of rows, so that two rows in the same group have the same values for the partitioning attributes
  - A function of aggregation
    - Input: Group of lines → Calculation → Output: A unique value
  - → Output table of aggregation has the number of aggregates as number of lines



identifiant	masse	diamètre	couleur	nom_variété	abîmée
1	151 g	8.3 cm	rouge	Ariane	FAUX
2	169 g	9.1 cm	jaune	Gala	FAUX
3	134 g	8.0 cm	jaune	Gala	VRAI
4	134 g	8.0 cm	jaune	Ariane	FAUX
14	142 g	7.3 cm	rouge	Ariane	FAUX
25	182 g	7.5 cm	rouge	Gala	VRAI

3	134 g	8.0 cm	jaune	Gala	VRAI
2	169 g	9.1 cm	jaune	Gala	FAUX
4	134 g	8.0 cm	jaune	Ariane	FAUX
1	151 g	8.3 cm	rouge	Ariane	FAUX
14	142 g	7.3 cm	rouge	Ariane	FAUX
25	182 g	7.5 cm	rouge	Gala	VRAI

3	134 g	8.0 cm	jaune	Gala	VRAI

2	169 g	9.1 cm	jaune	Gala	FAUX
4	134 g	8.0 cm	jaune	Ariane	FAUX

couleur	abîmée	avg(masse)	count()
jaune	VRAI	134 g	1
jaune	FAUX	151.5 g	2
rouge	FAUX	146.5 g	2
rouge	VRAI	182 g	1

SQL (1)

- CREATE TABLE tableName (attributeName ATTRIBUTE_TYPE [NOT NULL], …, PRIMARY KEY (attributeName), FOREIGN KEY(attributeName) REFERENCES tableName2(attributeName2));
  - ATTRIBUTE_TYPE = INTEGER, FLOAT, NUMERIC, VARCHAR, TEXT, TIMESTAMP, DATE, BOOLEAN…
- INSERT INTO tableName (attributeName, …) VALUES (attributeValue, …);
  - Must be in the same order!
  - .csv file is like a data base table

SELECT [DISTINCT] [Function] attributeName AS newAttributeName, … [*]

FROM tableName1, … [request] //anotherRequest → Nested request

JOIN tableName2, … ON junctureCondition

WHERE condition [IN/EXISTS/ALL/ANY request] //To use before aggregation

GROUP BY attributeName

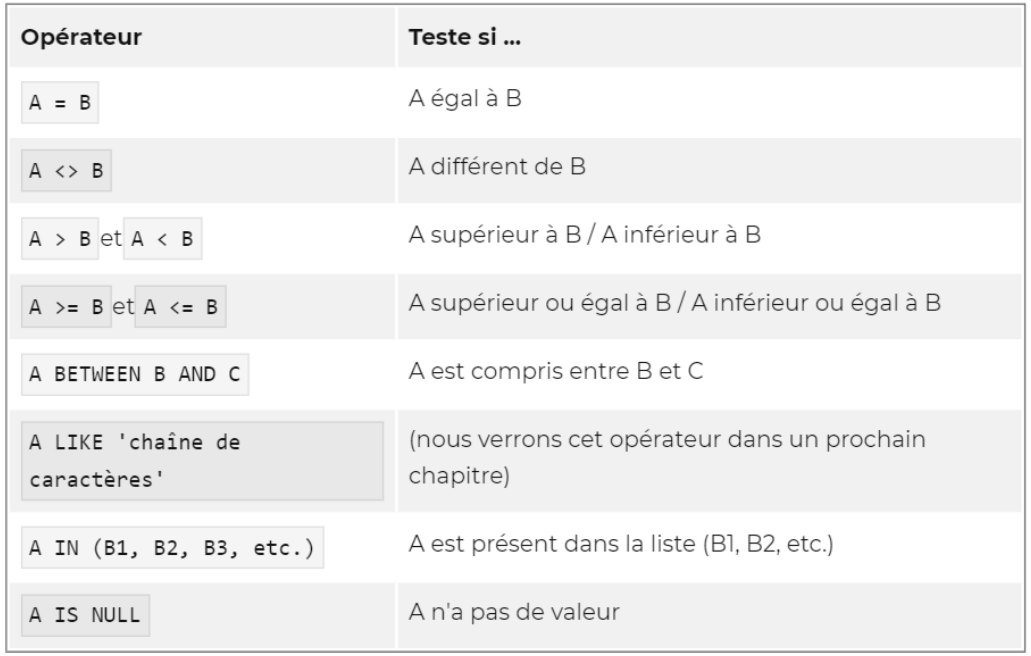
ORDER BY attributeName, … [DESC]

HAVING condition //To use after aggregation, can contain aggregation function

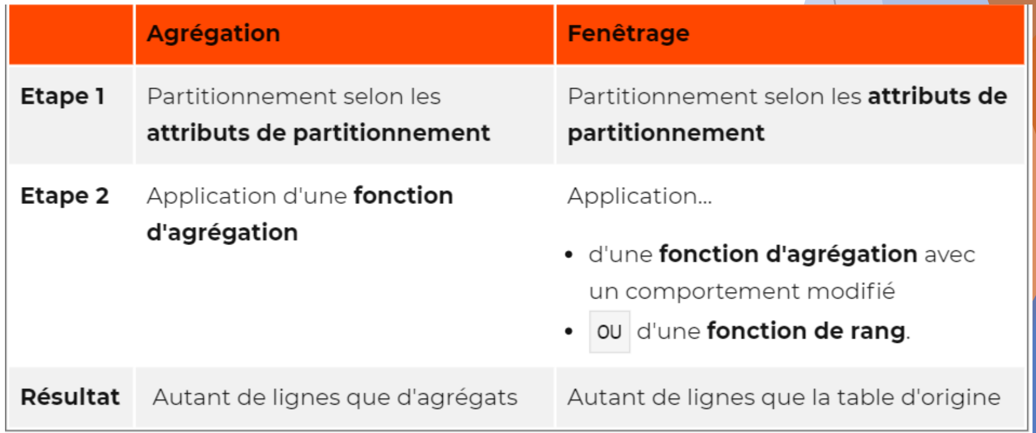
- Functions can be: Scalar (on each line) or Aggregation (on all lines)
- str LIKE '%str_' //Compare str → _ to replace unknown character | % to replace alot of uknown characters
- EXISTS checks if the request contains at least one row (Faster than IN)
- Function() **OVER([PARTITION BY] … [ORDER BY] …)** //Make aggregation function return the same lines as tableName

SQL (2)

- Request1 UNION Request2
- Request1 EXCEPT Request2 ⇔ Request1 WHERE attribute NOT IN Request2
- Request1 INTERSECT Request2 ⇔ Request1 WHERE attribute IN Request2
- SELECT * FROM t1, t2 WHERE (t1.fk = t2.pk); //Intern juncture method 1
- SELECT * FROM t1 JOIN t2 ON (t1.fk = t2.pk); //Intern juncture method 2
  - Other types of juncture: RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN, NATURAL JOIN



Opérateur	Teste si ...
`A = B`	A égal à B
`A <> B`	A différent de B
`A > B` et `A < B`	A supérieur à B / A inférieur à B
`A >= B` et `A <= B`	A supérieur ou égal à B / A inférieur ou égal à B
`A BETWEEN B AND C`	A est compris entre B et C
`A LIKE 'chaîne de caractères'`	(nous verrons cet opérateur dans un prochain chapitre)
`A IN (B1, B2, B3, etc.)`	A est présent dans la liste (B1, B2, etc.)
`A IS NULL`	A n'a pas de valeur



	Agrégation	Fenêtrage
Etape 1	Partitionnement selon les **attributs de partitionnement**	Partitionnement selon les **attributs de partitionnement**
Etape 2	Application d'une **fonction d'agrégation**	Application... • d'une **fonction d'agrégation** avec un comportement modifié • `OU` d'une **fonction de rang**.
Résultat	Autant de lignes que d'agrégats	Autant de lignes que la table d'origine