PREVIEW OF SECTION F.10

F.10 What is a Transform operation and how do I write it?

This chapter also answers the questions:

- **How do I create cross tables?**

[57] There could be some type conversion combinations that make **Union** operators non-associative.

- **What is a crosstab Query?**

A **Transform operation** is an SQL operation performed with the **Transform** operator (click *F.10.1*) plus its corresponding operands. Therefore, a **Transform** operation is the complete SQL code associated to the **Transform** operator.

A **Transform** operation produces cross tables in a very flexible way. In its **simplest** form, a **Transform** operation **generates new field names** from the **values** of the **field** written in the "**PIVOT**" clause, **and also**, places the **values** of the **expression** written in the "**TRANSFORM**" clause in the **correct** places **under** the **newly generated field names**.

Transform is **extremely** useful to **display** your Query **results** as a **cross table**. Imagine you have the following record-list (in this case it is a Table[58], but most frequently it is a Query result or an inner SQL operation):

T_Capital_Rainfall		
Capital	**Cal_Year**	**Rainfall**
Beijing	2018	25
Beijing	2019	41
Washington	2018	22
Washington	2019	17
Beijing	2020	27
Beijing	2021	38
Washington	2020	26
Washington	2021	9

Now you want to display this information as a **cross table**, with **fields** "**Capital**" (renamed to "**Cap_City**"), "**2018**", "**2019**", "**2020**" and "**2021**", and the **value** of "**Rainfall**" in the **corresponding cell under the fields** "**2018**", "**2019**", "**2020**" and "**2021**". In summary, what you want is:

F_Transform_1				
Cap_City	**2018**	**2019**	**2020**	**2021**
Beijing	25	41	27	38
Washington	22	17	26	9

Then, this is **exactly** what you would get with the following **Transform** operation[59]:

```
TRANSFORM First(Rainfall) AS GenVals
SELECT Capital AS Cap_City
FROM T_Capital_Rainfall
GROUP BY Capital
PIVOT Cal_Year ;
```

The "**n**" **leftmost output fields** of a **Transform** are called the "**SELECT**" **fields**, because they are determined by the "**SELECT**" clause. The **output fields** placed to the **right** of the "**SELECT**" **fields** are called the "**PIVOT**" **fields**, because they are

[58] This is the Table "**T_Capital_Rainfall**" in file "**Company_Database.accdb**".
[59] This is the Query "**F_Transform_1**" from the "**Company_Database.accdb**" file.

generated by converting to ***String*** the **returned values** of the "**`PIVOT`**" **expression** (click *F.10.11*), subject to the modifications of the optional "**`IN`**" **list** (click *F.10.12*). In this example, we have one "**`SELECT`**" **field** (field name "**`Cap_City`**") and four "**`PIVOT`**" **fields** (with field **names** "**2018**", "**2019**", "**2020**" and "**2021**").

Transform operations **cannot** be nested. The **Transform** operation must be **the first** operation in a Query, and a Query **cannot contain** any other **Transform** operation. Consequently, a **Transform** operation **cannot** be used as an **input record-list** in any other Query or SQL operation. A Query that has a **Transform** operation **cannot** be used in any other Query. The **Transform** operation can therefore only be the **very last** operation over a record-list.

If you want to know more about a **Transform** operation, you may click:

- "*F.10.1 What is the Transform operator?*"
- "*F.10.2 What is the input record-list ("`FROM`" clause) of a Transform?*"
- "*F.10.3 What are the output fields of a Transform?*"
- "*F.10.4 What is the output record-list of a Transform?*"
- "*F.10.5 Can I see an example of a Transform operation?*"
- "*F.10.6 What is the "`TRANSFORM`" clause of a Transform?*"
- "*F.10.7 What is the "`SELECT`" clause of a Transform?*"
- "*F.10.8 What is the "`ORDER BY`" clause of a Transform?*"
- "*F.10.9 What is the "`WHERE`" clause of a Transform?*"
- "*F.10.10 What is the "`GROUP BY`" clause of a Transform?*"
- "*F.10.11 What is the "`PIVOT`" clause of a Transform?*"
- "*F.10.12 What is the "`IN`" clause of a Transform?*"
- "*F.10.13 How do the clauses from Transform and Select compare?*"
- "*F.10.14 How do I write a correct (syntax) Transform?*"

F.10.1 What is the Transform operator?

Transform is a **consulting** (click *F.6*) operator that works over one single **input record-list**, plus other operands that are not record-lists.

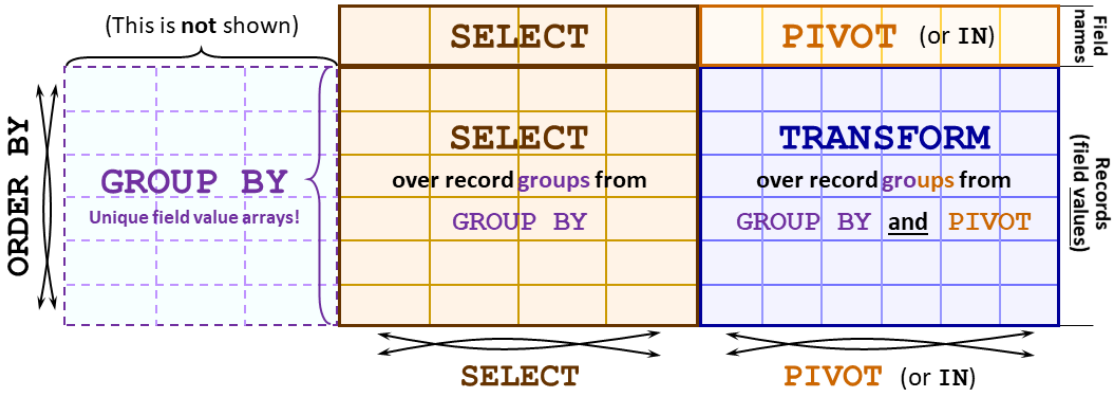
The **Transform** operator includes the clauses "**`SELECT`**", "**`DISTINCT`**" (optional and irrelevant), "**`DISTINCTROW`**" (optional and not advisable), "**`FROM`**", "**`WHERE`**" (optional), "**`GROUP BY`**", "**`ORDER BY`**" (optional) and "**`IN`**" (optional).

A simplified view of writing (syntax) a **Transform operation** is:

```
  TRANSFORM PIVOT-field-values-expression()
  SELECT [DISTINCT] [DISTINCTROW] Output-expression() 1 to n
  FROM Input-record-list
[ WHERE Where-Boolean-expression(Input-field-names) ]
  GROUP BY Group_by-expression(Input-field-names) 1 to k
[ ORDER BY Order_by-expression() 1 to w ]
  PIVOT PIVOT-field-names-expression(Input-field-names) ;
[ IN ( List-of-PIVOT-values ) ]
```

The clauses enclosed between square brackets "**[]**" are optional.

The following picture shows the **output** of the **Transform** operation:



The **Transform** operator creates a **cross table** by converting the **distinct values** of the "**PIVOT**" **expression** (click *F.10.11*) into **additional generated field names** (i.e., "**columns**") and presenting the **results** of the "**TRANSFORM**" **expression** (click *F.10.6*) in the corresponding **cells** (**row** x **column**).

The **Transform** operator produces:

- The **rows** by doing record **aggregation** over the (mandatory) "**GROUP BY**" **expressions**.
- The **fixed** leftmost **columns** (the "**SELECT**" **fields names** and **values**) with the conventional "**SELECT**" **expressions**.
- The **cross table column names** (the "**PIVOT**" **field names**) by doing record **aggregation** over the (mandatory) "**PIVOT**" **expression**,
- The **cross table column values** by doing record **aggregation** of the "**TRANSFORM**" **expression** over the "**GROUP BY**" **expressions** **jointly with** the "**PIVOT**" **expression**.

Remind that record aggregation produces **only one value** from **each group** of records. I now explain the above bullet points in more detail.

The **number** of **rows** is determined by the **1** to **k** "**GROUP BY**" **expressions** (click *F.10.10*). There will be **as many rows** as **groups** of (retained) **input records** (see below). **Each group** contains the (retained) **input records** that produce the **same results** in **all** the **1** to **k** "**GROUP BY**" **expressions**. Notice that the resulting **arrays** of "**k**" **results** from the "**GROUP BY**" **expressions** are **all distinct**. Notice also that these **result arrays** are **not shown** in the **output** of the **Transform operation** and are just internal for its processing. The "**GROUP BY**" clause of a **Transform** works exactly the

same as the "**GROUP BY**" clause (click *F.7.9*) of a **Select**.

The "**n**" **leftmost output fields** of a **Transform** are called the "**SELECT**" **fields**, because they are determined by the "**SELECT**" clause. The **output fields** placed to the **right** of the "**SELECT**" **fields** are called the "**PIVOT**" **fields**, because they are **generated** by converting to ***String*** the **returned values** of the "**PIVOT**" **expression** (click *F.10.11*), subject to the modifications of the optional "**IN**" **list** (click *F.10.12*).

The number of **select columns** "**n**", their **field names**, the **field order** and the **field values** are determined by the "**SELECT**" clause (click *F.10.7*) with the "**GROUP BY**" clause (click *F.10.10*). The **value** of **field** "**j**" from the **row** "**i**" is produced by applying the corresponding "**SELECT**" **expression** "**j**", to **all** the records in the **group** "**i**" of (retained) **input records** corresponding to **this row**. These **groups** of (retained) **input records** are **the same** as the ones indicated in the previous paragraph.

The **number** of "**PIVOT**" **columns** and their **field names** are determined by the **distinct values** returned by the "**PIVOT**" **expression** computed over **all** the (retained) **input records**. The **field order** is (click *F.10.3.3*), left to right, **ascending alphanumeric** by the **value** returned by the "**PIVOT**" **expression**. However, if the optional "**IN**" clause is used, then the **number** of "**PIVOT**" **columns**, the **field names** and the **field order** are **all** determined by the "**IN**" **list** (click *F.10.12*).

The **field value** of "**PIVOT**" **field** "**j**" from the **row** "**i**" is produced by applying the "**TRANSFORM**" **expression**, to **all** the records in the **group** "**(i, j)**" of (retained) **input records** corresponding to **row** "**i**" and **column** "**j**". Each **group** "**(i, j)**" contains the (retained) **input records** that produce the **same results** in **all** the **1** to **k** "**GROUP BY**" **expressions** as the **values** in **row** "**i**" **and the same result** in the "**PIVOT**" **expression** as the **field name** of column "**j**". If there is a **field name collision** with a "**SELECT**" **field**, or the "**IN**" clause is used, this becomes slightly more complex (click *F.10.4.1*).

The order of records is **unknown**, unless the optional "**ORDER BY**" clause (click *F.10.9*) is used. The ordering works the same as the "**ORDER BY**" clause (click *F.7.12*) from the **Select** operator, with just one restriction: the "**ORDER BY**" **expressions must** be a **list** of **any number** of **exactly the same** "**GROUP BY**" **expressions** or the "**PIVOT**" **expression**.

The "**Input-record-list**" is stated in the "**FROM**" clause. The "**FROM**" clause in the **Transform** operator works exactly the same as the "**FROM**" clause (click *F.7.4*) in the **Select** operators.

If the optional "**WHERE**" clause is used, the **input records** that produce ***True*** in the "**Where-Boolean-expression()**" are **retained**, while the other ones are **discarded**. The "**WHERE**" clause (click *F.10.9*) of a **Transform** operator works exactly the same as the "**WHERE**" clause (click *F.7.7*) of a **Select** operator. If against my advice you use the optional "**DISTINCTROW**" clause, you may click *F.7.8*.

Very important to highlight that if the **input record-list** is a Table name, no records will be **deleted** from the Table, and the Table remains **unmodified**. This is so because all the SQL **consulting** operators work over an **image** of Table's records, and **not** over the Table records themselves. Remind that the **Transform** operator is a **consulting** SQL operator (i.e., one that can only consult Tables) and it is not a **data-changing** SQL operator. If you want to actually **modify** your Table's records, you may click

"*F.6.2 What are the SQL data-changing operators?*".

If you want to know more about **Transform**, you may click:

- "*F.10.2 What is the input record-list ("FROM" clause) of a Transform?*"
- "*F.10.3 What are the output fields of a Transform?*"
- "*F.10.4 What is the output record-list of a Transform?*"
- "*F.10.5 Can I see an example of a Transform operation?*"
- "*F.10.13 How do the clauses from Transform and Select compare?*"
- "*F.10.14 How do I write a correct (syntax) Transform?*"

If you want to know the SQL **color codes** used in this **Lightning Guide**, you may click "*F.11.2 What are the SQL color codes used in this Guide?*".

F.10.2 What is the input record-list ("FROM" clause) of a Transform?

In a **Transform** operation (click *F.10.1*), the mandatory "**FROM**" clause indicates its **input record-list**, as follows:

```
FROM {    [[()] {Table-name or Query-name} []]                                ]
      or [     {Table-name or Query-name}     [AS Input-record-list-name] ]
      or [ (   {Select-opr or Union-opr }   ) [AS Input-record-list-name] ]
      or [[()]  Inner-or-Outer-Join-opr    []] or Cross-Join-opr            ] }
```

The **input record-list** is indicated in its "**FROM**" clause, **exactly the same** as the one from the **Select** operation.

If you want to write (syntax) a correct "**FROM**" clause, you may click *F.10.14*.

F.10.3 What are the output fields of a Transform?

You may click:

- "*F.10.3.1 What is the number of output fields of a Transform?*"
- "*F.10.3.2 What are the output field names of a Transform?*"
- "*F.10.3.3 What is the output field order of a Transform?*"
- "*F.10.3.4 What are the output data/field types of a Transform?*"
- "*F.10.4 What is the output record-list of a Transform?*"

F.10.3.1 What is the number of output fields of a Transform?

In a **Transform** operation (click *F.10.1*), the **number** of **output fields** is indicated in the "**SELECT**", "**PIVOT**" and "**IN**" clauses as follows:

```
TRANSFORM PIVOT-field-values-exp()
SELECT [DISTINCT] [TOP int [PERCENT]] [DISTINCTROW or ALL]
       {    * or
            Output-exp_1(exp-elements) [AS Output-field-name_1]
        [, ...
         , Output-exp_n(exp-elements) [AS Output-field-name_n] ] }
 ...
PIVOT PIVOT-field-names-exp(Input-field-names)
[IN ( List-of-PIVOT-values ) ]
```

The "**n**" **leftmost output fields** of a **Transform** are called the "**SELECT**" **fields**, because they are determined by the "**SELECT**" clause. The **output fields** placed to the **right** of the "**SELECT**" **fields** are called the "**PIVOT**" **fields**, because they are **generated** by converting to ***String*** the **returned values** of the "**PIVOT**" **expression** (click *F.10.11*), subject to the modifications of the optional "**IN**" **list** (click *F.10.12*).

What is the number of output "SELECT" fields of a Transform?

The **number** "**n**" of "**SELECT**" **fields** is the **number** of "**SELECT**" **expressions**. This is **exactly the same** as in a **Select** operation (click *F.7.5*).

What is the number of output "PIVOT" fields of a Transform?

If the optional "**IN**" clause is **not** used, the **number** of "**PIVOT**" **fields** is the **number** of **distinct values** returned by the "**PIVOT**" **expression** computed over **all** the (retained) **input records**.

Else, if the optional "**IN**" clause **is used**, then the **number** of "**PIVOT**" **fields** is the **number** of **values** in the "**IN**" **list**.

F.10.3.2 What are the output field names of a Transform?

In a **Transform** operation (click *F.10.1*), the **output field names** are indicated in the "**SELECT**", "**PIVOT**" and "**IN**" clauses as follows:

```
SELECT [DISTINCT] [TOP int [PERCENT]] [DISTINCTROW or ALL]
       {   * or
           Output-exp_1(exp-elements) [AS Output-field-name_1]
        [, ...
         , Output-exp_n(exp-elements) [AS Output-field-name_n] ] }
 ...
PIVOT PIVOT-field-names-exp(Input-field-names)
[IN ( List-of-PIVOT-values ) ]
```

The "**n**" **leftmost output fields** of a **Transform** are called the "**SELECT**" **fields**, because they are determined by the "**SELECT**" clause. The **output fields** placed to the **right** of the "**SELECT**" **fields** are called the "**PIVOT**" **fields**, because they are **generated** by converting to ***String*** the **returned values** of the "**PIVOT**" **expression** (click *F.10.11*), subject to the modifications of the optional "**IN**" **list** (click *F.10.12*).

What are the names of output "SELECT" fields of a Transform?

The **names** of the "**n**" "**SELECT**" **fields** are the identifiers "**Output-field-name_i**" from the "**SELECT**" clause. This is **exactly the same** as in a **Select** operation (click *F.7.5*).

What are the names of output "PIVOT" fields of a Transform?

Regarding the **names** of the "**PIVOT**" **fields**, we have the following three cases:

- The "**IN**" clause is **not** used, **and** the **distinct value** (converted to ***String***) returned by the "**PIVOT**" **expression** is **different** from **all** the "**SELECT**" **field names**. Then, the **name** of this "**PIVOT**" **field** is the **distinct value** (converted to ***String***) returned by the "**PIVOT**" **expression**.
- The "**IN**" clause **is used**, **and** the **value** (converted to ***String***) from the "**IN**" **list** is

different from **all** the "**SELECT**" **field names**.
Then, the **name** of this "**PIVOT**" **field** is the **value** (converted to ***String***) from "**IN**" **list**.

- The **value** (converted to ***String***), which is either a **distinct value** returned by the "**PIVOT**" **expression** or a **value** from the "**IN**" **list**, is the **same as** one of the "**SELECT**" **field names**.
  Then, the **name** of this "**PIVOT**" **field** is "**FieldN**", where "**N**" is an integer value assigned by MS-Access.

The data type of the "**PIVOT**" **expression** is **usually *String***, but it can also be any other data type. In case the data type of the **expression** **is not *String***, then MS-Access will **convert its result** to a ***String*** (to become a **field name**). The values **True/Yes/On** or **ticked** and **False/No/Off** or **unticked** are converted to the **field names** "**-1**" and "**0**", respectively. An **integer-like** data type value is converted to the **field name** of the equivalent ***String*** (e.g., "**-12**", "**14**"). A ***Date*** value is converted to the **field name** of the equivalent ***String*** (e.g., "**04/05/2003**"). A fractional data type is converted to the **field name** of the equivalent ***String***, but, replacing the period "**.**" with underscore "**_**". The reason for this is that names **cannot** contain a period "**.**" (click *D.2.5*). Therefore, the number "**-0.45**" is converted to the **field name** "**-0_45**".

In case that one or more **input records** make the expression in the "**PIVOT**" clause produce **Null**, these **Nulls** will produce a **valid** "**PIVOT**" **field name** which is "**<>**". I strongly recommend that you **avoid** using the string "**<>**" in the "**List-of-PIVOT-values**" of the "**IN**" clause, to avoid confusion with a "**PIVOT**" field arising from a **Null** (which is most likely not intentional).

A **Null**, either returned by the "**PIVOT**" **expression** or written in the "**IN**" **list**, will produce "**<>**" as the "**PIVOT**" **field name**. If both a **Null** and an explicit "**<>**" happen, the field corresponding to **Null** will be named "**<>**", while the field corresponding to "**<>**" will be named "**FieldN**", where "**N**" is an integer value assigned by MS-Access.

Field names **cannot** include the period "**.**" character (click *D.2.5*). For this reason, every period "**.**" character in a **value** from the "**PIVOT**" **expression** or in a **value** from the "**IN**" **list**, will be converted to the underscore "**_**" character in the resulting "**PIVOT**" **field name**. For example, the returned **values** numeric "**3.4**" and string "**Oh.No**" from the "**PIVOT**" **expression** produce the "**PIVOT**" **field names** "**3_4**" and "**Oh_No**" respectively; likewise, the **values** "**54.6**" and "**Hi.there**" found in the "**IN**" **list** produce the "**PIVOT**" **field name** "**54_6**" and "**Hi.there**", respectively.

If you use the "**IN**" clause, the **data type** returned by the "**PIVOT**" **expression** **must** be the same as the one of **all** the elements in the "**IN**" **list**, either **directly** or through **type conversion**. Otherwise, the Query will **crash**. Curiously, a **Null**, either **returned** by the "**PIVOT**" **expression** or explicitly **written** in "**IN**" **list**, is always considered as having a compatible data type. I now present a few examples to clarify this paragraph:

- The following works, because "**Cal_Year**" is a **number** and **all** the elements in the "**IN**" **list** are **numbers**, or **can be** converted to a **number**, or are **Null**.

```
PIVOT Cal_Year
IN (2016, 2020, "2018", "34", Null)
```

- The following **crashes**, because "**`Cal_Year`**" is a **number** and the **string** "**`Hello`**" **cannot** be converted to a **number**.

```
PIVOT Cal_Year
IN (2016, 2020, "2018", "Hello", Null)
```

- The following works, because "**`Date_Time`**" is a ***Date/Time*** and **all** the elements in the "**`IN`**" **list** are either ***Date/Time***, or **can be** converted to a ***Date/Time***, or are **Null**.

```
PIVOT Date_Time
IN (#2018-1-1#, "3/January/2020", Null)
```

- The following **crashes**, because "**`Date_Time`**" is a ***Date/Time*** and the **string** "**`Hello`**" **cannot** be converted to a **number**.

```
PIVOT Date_Time
IN (#2018-1-1#, "Hello", Null)
```

F.10.3.3 What is the output field order of a Transform?

In a **Transform** operation (click *F.10.1*), the **output field order** is indicated in the "**`SELECT`**", "**`PIVOT`**" and "**`IN`**" clauses as follows:

```
SELECT [DISTINCT] [TOP int [PERCENT]] [DISTINCTROW or ALL]
       {   * or
           Output-exp_1(exp-elements) [AS Output-field-name_1]
        [, ...
         , Output-exp_n(exp-elements) [AS Output-field-name_n] ] }
 ...
PIVOT PIVOT-field-names-exp(Input-field-names)
[IN ( List-of-PIVOT-values ) ]
```

The "**n**" **leftmost output fields** of a **Transform** are called the "**`SELECT`**" **fields**, because they are determined by the "**`SELECT`**" clause. The **output fields** placed to the **right** of the "**`SELECT`**" **fields** are called the "**`PIVOT`**" **fields**, because they are **generated** by converting to ***String*** the **returned values** of the "**`PIVOT`**" **expression** (click *F.10.11*), subject to the modifications of the optional "**`IN`**" **list** (click *F.10.12*).

What is the order of output "`SELECT`" fields of a Transform?

The **order** of the "**n**" "**`SELECT`**" **fields** is the same as the one of the "**`SELECT`**" **expressions**. This is **exactly the same** as in a **Select** operation (click *F.7.5*).

What is the order of output "`PIVOT`" fields of a Transform?

If the optional "**`IN`**" clause is **not** used, the **order** of "**`PIVOT`**" **fields** is, left to right, **ascending** by the **value** produced by the "**`PIVOT`**" **expression**. Notice that if the "**`PIVOT`**" **expression** produces a **numeric value**, the **field** ordering will be ascending **numeric**, while if it produces a ***String*** **value** the field ordering will be ascending **alphabetical**, which is **different**. You can easily change between both by enclosing the "**`PIVOT`**" **expression** in a type conversion function (click *G.2.5*). If a **Null** is produced, it will **always** be the **first** (leftmost) "**`PIVOT`**" **field**, with **field name** "**`<>`**".

Else, if the optional "**`IN`**" clause **is used**, then the "**`PIVOT`**" **field order** is the one you wrote in the "**`IN`**" **list** (click *F.10.12*).

Notice that the field order just described is **maintained** even when a "**`PIVOT`**" **field name** (either converted from the "**`PIVOT`**" **expression** or from the "**`IN`**" **list**) is **the**

same as a "**SELECT**" **field name**. In this case, the "**PIVOT**" **field name** is changed to "**FieldN**", but its **position** within the "**PIVOT**" **fields** is **not** changed.

The "**PIVOT**" **field name** corresponding to a **Null** value (i.e., the field name "**<>**") can **also** be reordered by including it in the "**IN**" **list**. I strongly recommend that you **never use** the string "**<>**" neither as a "**SELECT**" **field name**, nor as a **name** in the "**IN**" list, nor as a name produced by the "**PIVOT**" **expression**. If you use "**<>**" in either of these three cases, you create the risk of **mistaking** that field with a "**PIVOT**" **field** arising from **Null** (which is most likely not intentional).

F.10.3.4 What are the output data/field types of a Transform?

In a **Transform** operation (click *F.10.1*), the **output** data/field **types** are indicated in the "**TRANSFORM**", "**SELECT**", "**PIVOT**" and "**IN**" clauses as follows:

```
TRANSFORM PIVOT-field-values-exp()
SELECT [DISTINCT] [TOP int [PERCENT]] [DISTINCTROW or ALL]
       {   * or
           Output-exp_1(exp-elements) [AS Output-field-name_1]
        [, ...
         , Output-exp_n(exp-elements) [AS Output-field-name_n] ] }
 ...
PIVOT PIVOT-field-names-exp(Input-field-names)
[IN ( List-of-PIVOT-values ) ]
```

The "**n**" **leftmost output fields** of a **Transform** are called the "**SELECT**" **fields**, because they are determined by the "**SELECT**" clause. The **output fields** placed to the **right** of the "**SELECT**" **fields** are called the "**PIVOT**" **fields**, because they are **generated** by converting to ***String*** the **returned values** of the "**PIVOT**" **expression** (click *F.10.11*), subject to the modifications of the optional "**IN**" **list** (click *F.10.12*).

What is the data/field type of output "SELECT" fields of a Transform?

The **data/field type** of the "**n**" "**SELECT**" **fields** are the ones of the corresponding "**SELECT**" **expressions**. This is **exactly the same** as in a **Select** operation (click *F.7.5*).

What is the data/field type of output "PIVOT" fields of a Transform?

Regarding the data/field **type** of the "**PIVOT**" **fields**, we have the following three cases:

- The "**IN**" clause is **not** used, **and** the **distinct value** (converted to ***String***) returned by the "**PIVOT**" **expression** is **different** from **all** the "**SELECT**" **field names**.
  Then, the **data type** of this "**PIVOT**" **field** is the one of the "**TRANSFORM**" **expression** "**PIVOT-field-values-exp()**".
- The "**IN**" clause **is used**, **and** the **value** (converted to ***String***) from the "**IN**" **list** is **different** from **all** the "**SELECT**" **field names**.
  Then, the **data type** of this "**PIVOT**" **field** is **unknown** (and all this field's values are **Null**).
- The **value** (converted to ***String***), which is either a **distinct value** returned by the "**PIVOT**" **expression** or a **value** from the "**IN**" **list**, is the **same as** one of the "**SELECT**" **field names**.
  Then, the **data type** of this "**PIVOT**" **field** is the one of the **same-name** "**SELECT**" **field**.

If you want to know what are the **output** field **values** and/or what is the **output record-list** of a **Transform**, you may click "*F.10.4 What is the output record-list of a Transform?*".

F.10.4 What is the output record-list of a Transform?

You may click:

- "*F.10.4.1 What are the output field values of a Transform?*"
- "*F.10.4.2 What are the output records of a Transform?*"
- "*F.10.4.3 How many output records does a Transform produce?*"

F.10.4.1 What are the output field values of a Transform?

In a **Transform**, the **output field values** are determined by the "**TRANSFORM**", "**SELECT**", "**PIVOT**" and "**IN**" clauses as follows:

```
TRANSFORM
 PIVOT-field-values-exp      (  Output-field-names
                             , PIVOT-field-names-exp(Input-field-names)
                             , Group_by-exp_1(Input-field-names)
                             , ...
                             , Group_by-exp_k(Input-field-names)
                             , SQL_agg_func(exp_t11(INOUT-field-names))
                             , ...
                             , SQL_agg_func(exp_t1d(INOUT-field-names))
                            ) [AS Output-values-Identifier]

 SELECT [DISTINCT] [DISTINCTROW or ALL]
                Output-exp_1(  Output-field-names
                             , PIVOT-field-names-exp(Input-field-names)
                             , Group_by-exp_1(Input-field-names)
                             , ...
                             , Group_by-exp_k(Input-field-names)
                             , SQL_agg_func(exp_o11(INOUT-field-names))
                             , ...
                             , SQL_agg_func(exp_o1y(INOUT-field-names))
                            ) [ AS Output-field-name_1 ]
            [ , ...
              , Output-exp_n(  Output-field-names
                             , PIVOT-field-names-exp(Input-field-names)
                             , Group_by-exp_1(Input-field-names)
                             , ...
                             , Group_by-exp_k(Input-field-names)
                             , SQL_agg_func(exp_on1(INOUT-field-names))
                             , ...
                             , SQL_agg_func(exp_onz(INOUT-field-names))
                            ) [ AS Output-field-name_n ] ]
   ...
  PIVOT PIVOT-field-names-exp(Input-field-names)
 [IN ( List-of-PIVOT-values ) ]
```

The "**n**" **leftmost output fields** of a **Transform** are called the "**SELECT**" **fields**, because they are determined by the "**SELECT**" clause. The **output fields** placed to the **right** of the "**SELECT**" **fields** are called the "**PIVOT**" **fields**, because they are **generated** by converting to ***String*** the **returned values** of the "**PIVOT**" **expression** (click *F.10.11*), subject to the modifications of the optional "**IN**" **list** (click *F.10.12*).

What are the values of output "SELECT" fields of a Transform?

The **values** of the "**n**" "**SELECT**" **fields** are the result of the "**SELECT**" **expressions**,

computed over the **groups** of (retained) **input records** produced by the "**GROUP BY**" **expressions**. This is the same as in a **Select-group_by_aggreg** operation (click *F.7.6.2*), with only one difference. The one difference is that in a **Transform**, the "**SELECT**" **expressions 1** to **n** can **also** use as an **element** the "**PIVOT**" **expression**. You may see this in the SQL code above. This is somehow **surprising**, because the "**PIVOT**" **expression** produces **multiple** values (**different** in the general case) when computed over the (retained) **input records** in the **group** corresponding to a given **output** record (remind that we are using a "**GROUP BY**" clause). The way this works is that when evaluating a "**SELECT**" **expression** that includes the "**PIVOT**" **expression**, it returns the result **as if** the "**PIVOT**" **expression** was **enclosed** in the "`Min()`" SQL aggregate function. This is, whenever you write the the "**PIVOT**" **expression** "`PIVOT-field-names-expression()`" within a "**SELECT**" **expression**, it works **as if** you had written "`Min(PIVOT-field-names-expression())`".

Notice that if a "**PIVOT**" **field** is discarded because the "**IN**" clause is used, and the corresponding **result** from the "**PIVOT**" **expression** is not in the "**IN**" **list**, the **result** of using the "**PIVOT**" **expression** within the "**SELECT**" **expression** will remain unaffected. This is, you will still get the "`Min()`" **value** over **all** the values produced, even if the "`Min()`" **value** is **not** in the "**IN**" **list**.

Finally, as a rather strange case, if you use the "**PIVOT**" **expression** as an **element** of a "**SELECT**" **expression**, and you also use the "**IN**" clause, the corresponding "**SELECT**" **expression** will produce the exception value "**#Error**". Curiously, if the "**SELECT**" **expression** is **exactly the same** as the "**PIVOT**" **expression**, this works fine. This may be an MS-Access **bug**.

What are the values of output "PIVOT" fields of a Transform?

Regarding the **values** of the "**PIVOT**" **fields**, we have the following three cases:

- The "**IN**" clause is **not** used, **and** the **distinct value** (converted to ***String***) returned by the "**PIVOT**" **expression** is **different** from **all** the "**SELECT**" **field names**.
  Then, the **value** of this "**PIVOT**" **field** is the result of the "**TRANSFORM**" **expression** computed over the **groups** of (retained) **input records** produced by the "**GROUP BY**" **expressions jointly with** the "**PIVOT**" **expression**.
  In more detail, the **value** of **field** "**j**" and **row** "**i**" is produced by applying the "**TRANSFORM**" **expression**, to **all** the records in the **group** "**(i, j)**" of (retained) **input records** corresponding to **row** "**i**" and **column** "**j**". Each **group** "**(i, j)**" contains the (retained) **input records** that produce the **same results** in **all** the **1** to **k** "**GROUP BY**" **expressions** as the **values** in **row** "**i**" **and the same result** in the "**PIVOT**" **expression** as the **field name** of column "**j**". Since these **groups** of **input records** depend on the result of the "**PIVOT**" **expression**, they will be different (in the general case) for **each** "**PIVOT**" **field**, and therefore, the "**TRANSFORM**" **expression** will produce **different values** (i.e., different **columns** of **values**) below **each** of the "**PIVOT**" **fields**.

- The "**IN**" clause **is used**, **and** the **value** (converted to ***String***) from the "**IN**" **list** is **different** from **all** the "**SELECT**" **field names**.
  Then, the **value** of this "**PIVOT**" **field** is **Null** in **all** the **output** records.

- The **value** (converted to ***String***), which is either a **distinct value** returned by the

"**PIVOT**" **expression** or a **value** from the "**IN**" **list**, is the **same as** one of the "**SELECT**" **field names**.
Then, the **value** of this "**PIVOT**" **field** is **the same as** the one in the **same-name** "**SELECT**" **field** in **each and every output** record. This is, the "**column**" of **values** under this **field** is the same as the "**column**" of **values** under the **same-name** "**SELECT**" **field**.

In the first bullet above, notice that it is **not surprising** to be able to use the "**PIVOT**" **expression** as one of the **elements** of the "**TRANSFORM**" **expression**, because in this case the "**PIVOT**" **expression** produces **the same value** in **every** record of **each group**. However, in the explanation above in "***What are the values of output "SELECT" fields of a Transform?***" it was surprising that you could use the "**PIVOT**" **expression** as an **element** of the "**SELECT**" **expressions**, because in that case the "**PIVOT**" **expression may** produce **different** values in the **different** records of **each group**.

Why does the same expression produce different values within the "SELECT" and "TRANSFORM" expressions?

Because in a **Transform**, the "**SELECT**" **expressions** and the "**TRANSFORM**" **expression** are computed over **different groups** of **input records**. The "**SELECT**" **expressions** are computed over **groups** of **input records** that produce the same results in **all** the "**GROUP BY**" **expressions**. However, the "**TRANSFORM**" **expression** is computed over **groups** of **input records** that produce the same result in **all** the "**GROUP BY**" **expressions and also** in the "**PIVOT**" **expression**.

Let me show this with an example over the Table "**T_Capital_Rainfall_District**".

T_Capital_Rainfall_District			
Capital	**District**	**Cal_Year**	**Rainfall**
Beijing	Dongcheng	2018	14
Beijing	Xicheng	2018	11
Beijing	Dongcheng	2019	18
Beijing	Xicheng	2019	23
Washington	Downtown	2018	10
Washington	Bloomingdale	2018	12
Washington	Downtown	2019	8
Washington	Bloomingdale	2019	9

If you now run the Query[60]:

```
TRANSFORM Sum(Rainfall) AS GenVals
SELECT Capital AS Cap_City, Sum(Rainfall) AS Total
FROM T_Capital_Rainfall_District
GROUP BY Capital
PIVOT Cal_Year ;
```

[60] This is the Query "**F_Transform_sum**" from the "**Company_Database.acccb**" file.

you then get the result:

F_Transform_Sum			
Cap_City	Total	2018	2019
Beijing	32	14	18
Beijing	34	11	23
Washington	18	10	8
Washington	21	12	9

Notice how the "**TRANSFORM**" **expression** "**Sum(Rainfall)**" and the "**SELECT**" **expression** "**Sum(Rainfall)**", that are **exactly the same**, produce **different** results under the field names "**Total**", "**2018**" and "**2019**". The reason is that the "**SELECT**" **expression** "**Sum(Rainfall)**" is computed over the record **groups** produced by the "**GROUP BY**" **expressions**, while **the same** "**TRANSFORM**" **expression** "**Sum(Rainfall)**" is computed over the record **groups** produced by the "**GROUP BY**" **expressions jointly with** the "**PIVOT**" **expression**. In other words, the "**SELECT**" **expression** "**Sum(Rainfall)**" is the **sum** for each "**Capital**" (i.e., adding **all districts** and **all years**) while the "**TRANSFORM**" **expression** "**Sum(Rainfall)**" is the **sum** for each "**Capital**" **and** each "**Cal_Year**" (i.e., adding **all districts**). Notice how the "**TRANSFORM**" **expression** produces a **different column** of **values** for each "**PIVOT**" **field** (i.e., for "**2018**" and "**2019**").

Since in this case we are using the "**Sum()**" aggregate function, the result under "**Total**" is the addition of the results under "**2018**" and "**2019**". However, notice that this **does not** happen with some other SQL aggregate functions (see the last bullet point of *F.10.5*).

F.10.4.2 What are the output records of a Transform?

In a **Transform**, the **output records** are determined by the optional "**WHERE**" clause, the "**GROUP BY**" clause and the optional "**ORDER BY**" clause as follows:

```
[WHERE Where-Boolean-exp(Input-field-names) ]
 GROUP BY     Group_by-exp_1(Input-field-names)
           [ , ...
             , Group_by-exp_k(Input-field-names) ]

[ORDER BY     Group_by-exp_x(Input-field-names) [DESC]
           [ , ...
             , Group_by-exp_y(Input-field-names) [DESC] ] ]
```

Transform produces **as many records** as **groups** of (retained) **input records** from the "**GROUP BY**" **expressions**. **Each group** contains the (retained) **input records** that produce the **same results** in **all** the "**GROUP BY**" **expressions**. Notice that the resulting **field value arrays** from "**Group_by-expression()**" **1** to **k** are **all distinct**. Notice also that these resulting **field value arrays** are **not shown** in the result of the **Transform operation**.

For each such **output record** its field **values** will be the ones indicated in the previous subsection *F.10.4.1*.

The order of records is **unknown**, unless the optional "**ORDER BY**" clause

(click *F.10.12*) is used. The ordering works the same as the "**`ORDER BY`**" clause (click *F.7.12*) from the **Select** operator, with just one restriction: the expressions in the **Transform** "**`ORDER BY`**" clause **must** be a **list** of **any number** of **exactly the same** "**`GROUP BY`**" **expressions**.

F.10.4.3 How many output records does a Transform produce?

Knowing how many records a **Transform** produces is **very** useful when **debugging** your Queries.

A **Transform** produces **as many records** as **groups** of (retained) **input records** are produced by the "**`GROUP BY`**" **expressions**. **Each group** contains the (retained) **input records** that produce the **same results** in **all** the "**`GROUP BY`**" **expressions**.

Notice that the **number** of **output** records is determined only by the (retained) **input records** (i.e., the "**`WHERE`**" ***Boolean*** **expression**) and the "**`GROUP BY`**" **expressions**.

F.10.5 Can I see an example of a Transform operation?

A simple but quite complete example of a **Transform operation**[61] is:

```
TRANSFORM StDev(Temp_Max) AS GenVals
SELECT  Capital AS Cap_City, Cal_Year AS C_Year
      , StDev(Temp_Max) AS StDev_T_Max_Year
FROM T_Capital_temps
GROUP BY Capital, Cal_Year
PIVOT Quart
```

The Table "**`T_Capital_Temps`**" used in this example has the following structure and values:

T_Capital_Temps					
City	District	Cal_Year	Quart	Temp_max	Temp_min
Beijing	Dongcheng	2018	Q1	12.3	0
Beijing	Dongcheng	2018	Q2	4	1
Beijing	Dongcheng	2018	Q3	7.8	6.7
Beijing	Dongcheng	2018	Q4	17	15
Beijing	Xicheng	2018	Q1	2.26	-3.25
Beijing	Xicheng	2018	Q2	5.6	-4.5
Beijing	Xicheng	2018	Q3	30	25
Beijing	Xicheng	2018	Q4	18	13
Brasilia	Asa_Norte	2019	Q1	7.4	-0.96
Brasilia	Asa_Norte	2019	Q2	7.57	-1.05
Brasilia	Asa_Norte	2019	Q3	17.5	7.4
Brasilia	Asa_Norte	2019	Q4	10.04	2.35
Brasilia	Guara_I	2019	Q1	10.62	3.17
Brasilia	Guara_I	2019	Q2	11.43	4.21
Brasilia	Guara_I	2019	Q3	12.4	5.52
Brasilia	Guara_I	2019	Q4	11.12	3.81

[61] This is the Query "**F_Transform_examp**" from the "**Company_Database.accdb**" file.

T_Capital_Temps					
City	District	Cal_Year	Quart	Temp_max	Temp_min
Washington	Anacostia	2018	Q1	6.67	-0.57
Washington	Anacostia	2018	Q2	7.55	-1.24
Washington	Anacostia	2018	Q3	7.65	-1.07
Washington	Anacostia	2018	Q4	7.68	-0.95
Washington	Downtown	2018	Q1	12.13	-9.55
Washington	Downtown	2018	Q2	5.67	-3.25
Washington	Downtown	2018	Q3	2.26	-4.3
Washington	Downtown	2018	Q4	17.57	-2.23
Washington	Bloomingdale	2018	Q1	3.15	2.13
Washington	Bloomingdale	2018	Q2	7.16	-1.92
Washington	Bloomingdale	2018	Q3	7.53	-1.58
Washington	Bloomingdale	2018	Q4	8.85	-0.9

Then, the **Transform** operation above would produce the following **output** record-list:

F_Transform_examp						
Cap_City	C_Year	StDev_T_Max_Year	Q1	Q2	Q3	Q4
Beijing	2018	9.28	7.10	1.13	15.70	0.71
Brasilia	2019	3.17	2.28	2.73	3.61	0.76
Washington	2018	3.98	4.52	0.99	3.08	5.40

Let me explain in more detail why are you getting this **output** record-list.

What is the input record-list of this example?

The **input record-list** is the Table "`T_Capital_Temps`" in the "**`FROM`**" clause.

Since in this example there is no "**`WHERE`**" clause, the **retained input records** are **all** the records in the Table "`T_Capital_Temps`".

What is the output record-list of this example?

The **output** record-list of is determined by the "**`GROUP BY`**" clause. The "**`GROUP BY`**" **expressions** are "`Capital`" and "`Cal_Year`". Therefore, the **output** records correspond to the **distinct values** produced by the (retained) **input records** in the "**`GROUP BY`**" **expressions** "`Capital`" and "`Cal_Year`". If you check the Table "`T_Capital_Temps`" you will see that there are **three** distinct **arrays** of values for "`Capital`" and "`Cal_Year`":

Capital	Cal_Year
Beijing	2018
Brasilia	2019
Washington	2018

What are the output field names of this example?

The **names** of the "**`SELECT`**" **fields** are "`Cap_City`", "`C_Year`" and

"`StDev_T_Max_Year`", as indicated in the "**`SELECT`**" clause.

The **names** of the "**`PIVOT`**" **fields** are "**Q1**", "**Q2**", "**Q3**" and "**Q4**", because these are the **distinct** results of the "**`PIVOT`**" **expression** "**Quart**", when computed over **all** the (retained) **input records**.

What is the output field order of this example?

The "**`SELECT`**" **fields** are produced in that order because this is how they appear in the "**`SELECT`**" clause.

The "**`PIVOT`**" **fields** are produced in that order because the data type of the "**`PIVOT`**" **expression** is ***Strings***, and ***Strings*** are ordered in **alphabetical** order.

What are the output field values of this example?

The **values** of the "**`SELECT`**" **fields** are the result of the "**`SELECT`**" **expressions** "`Capital`", "`Cal_Year`" and "`StDev(Temp_Max)`" computed over the **groups** of (retained) **input records** produced by the "**`GROUP BY`**" **expressions**.

The (retained) **input records** in each **group** are the ones that produce the same values in the "**`GROUP BY`**" **expressions**. In this example, there are three **groups**, each characterized by the **results** "(**Beijing**, **2018**)", "(**Brasilia**, **2019**)" and "(**Washington**, **2018**)". You may check that the **values** of the "**`SELECT`**" **fields** above correspond to the explanation I have just given.

The **values** of the "**`PIVOT`**" **fields** are the result of the "**`TRANSFORM`**" **expression** "`StDev(Temp_max)`" computed over the **groups** of (retained) **input records** produced by the "**`GROUP BY`**" **expressions** **jointly with** the "**`PIVOT`**" **expression**.

In more detail, the **value** of **field** "**j**" from the **output record** "**i**" is produced by applying the "**`TRANSFORM`**" **expression** "`Avg(Temp_max)`", to **all** the records in the **group** "(**i**, **j**)" of (retained) **input records** corresponding to **record** "**i**" and **field** "**j**". Each **group** "(**i**, **j**)" contains the (retained) **input records** that produce the **same results** in **all** the **1** to **k** "**`GROUP BY`**" **expressions** as the **values** in **row** "**i**" **and the same result** in the "**`PIVOT`**" **expression** as the **field name** of column "**j**".

I want to point out **two relevant** aspects from this example, that apply to **all Transform** operations:

- Notice that the "**`SELECT`**" **expression** "`StDev(Temp_Max)`" is **exactly the same** as the "**`TRANSFORM`**" **expression** "`StDev(Temp_Max)`". However, being the same expression it produces **different results** because it is computed over **different groups** of (retained) **input records**. The "**`SELECT`**" **expression** is computed over **groups** produced by the "**`GROUP BY`**" **expressions**, while the "**`TRANSFORM`**" **expression** is computed over **groups** produced the "**`GROUP BY`**" **expressions** **jointly with** the "**`PIVOT`**" **expression**.
- Notice that the field **values** of "`StDev_T_Max_Year`" are **not** the result of SQL aggregate function "`StDev()`" over the **values** of **fields** "**Q1**", "**Q2**", "**Q3**" and "**Q4**", in **each row**. If they were computed like that, the result would be: "**7.00**", "**1.19**" and "**1.93**, which is different from the actual **output** of the Query: "**9.28**", "**3.17**" and "**3.98**".

The reason is that the field **values** of "**StDev_T_Max_Year**" are the result of the "**SELECT**" **expression** "**StDev(Temp_Max)**" computed over the **groups** of (retained) **input records** produced by the "**GROUP BY**" **expressions**. In this example, this means computing the function "**StDev()**" over **all** the values of "**Temp_Max**" for each "**Capital**" and each "**Cal_Year**".

F.10.6 What is the "TRANSFORM" clause of a Transform?

In a **Transform** operation (click *F.10.1*), the mandatory "**TRANSFORM**" clause determines the "**PIVOT**" data/field **types** (click *F.10.3.4*) and the "**PIVOT**" **field values** (click *F.10.4.1*), as follows:

```
TRANSFORM PIVOT-field-values-expression()
```

If you want to write (syntax) a correct "**TRANSFORM**" clause, you may click *F.10.14*.

F.10.7 What is the "SELECT" clause of a Transform?

In a **Transform** operation (click *F.10.1*), the mandatory "**SELECT**" clause determines the "**SELECT**" **fields** (click *F.10.3*), the "**SELECT**" **field names** (click *F.10.3.2*), the "**SELECT**" **field order** (click *F.10.3.3*), the "**SELECT**" data/field **types** (click *F.10.3.4*) and the "**SELECT**" **field values** (click *F.10.4.1*), as follows:

```
SELECT    Output-expression_1() [ AS Output-field-name_1 ]
      [ , ...
        , Output-expression_n() [ AS Output-field-name_n ] ]
```

If you want to write (syntax) a correct "**SELECT**" clause, you may click *F.10.14*.

F.10.8 What is the "ORDER BY" clause of a Transform?

In a **Transform** operation (click *F.10.1*), the optional "**ORDER BY**" clause determines the **order** of the **output** records, as follows:

```
ORDER BY [    Group_by-exp_x(Input-field-names) [DESC]
           ,  ...
           ,  Group_by-exp_y(Input-field-names) [DESC] ]
         [[,] PIVOT-field-names-exp(Input-field-names) [DESC]]
```

Each "**ORDER BY**" **expression** must be **exactly the same** as one of the "**GROUP BY**" **expressions** or as the "**PIVOT**" **expression**, which is a very strong restriction.

Aside from this restriction on the **Transform** "**ORDER BY**" **expressions**, the "**ORDER BY**" of a **Transform** works exactly the same as the "**ORDER BY**" of a **Select**: you may click "*F.7.12 How do I use "ORDER BY" to order the output records of a Select?*".

If you want to write (syntax) a correct "**ORDER BY**" clause, you may click *F.10.14*.

F.10.9 What is the "WHERE" clause of a Transform?

In a **Transform** operation (click *F.10.1*), the optional "**WHERE**" clause indicates what are the **retained input records**, as follows:

```
WHERE Where-Boolean-expression(Input-field-names)
```

The "**WHERE**" ***Boolean*** **expression** is built using the "**Input-field-names**"

combining them with functions (excluding SQL aggregate), value operators and constants.

The "**WHERE**" clause of a **Transform** works **exactly the same** as the "**WHERE**" clause of a **Select**. You may see "*F.7.7 What is the "WHERE" clause of a Select?*".

If you want to write (syntax) a correct "**WHERE**" clause, you may click *F.10.14*.

F.10.10 What is the "GROUP BY" clause of a Transform?

In a **Transform** operation (click *F.10.1*), the mandatory "**GROUP BY**" clause determines the **output records** (click *F.10.4.2*), as follows:

```
GROUP BY     Group_by-expression_1(Input-field-names)
          [ , ...
            , Group_by-expression_k(Input-field-names) ]
```

Each "**GROUP BY**" **expression** is built using the "**Input-field-names**" combining them with functions (excluding SQL aggregate), value operators and constants.

The "**GROUP BY**" of a **Transform** works **exactly the same** as the "**GROUP BY**" of a **Select**. You may click "*F.7.9 What is the "GROUP BY" clause of a Select-group by aggreg?*".

There is also a very subtle, but very interesting, difference: the "**GROUP BY**" clause is optional in a **Select**, but it is mandatory in a **Transform**. This is because you can only do a cross table if there is **only one value** for each **row** and **column**. If you had **several values** for **each row** and **column**, there would be **no criterion** to choose one of them to be displayed. Therefore, the **Transform** operation **guarantees** that there is **only one value** for each **row** and **column**. The way to **guarantee** this is producing the cross table **values** from record **aggregation** of the **rows** and **columns**. Remind that record aggregation produces **only one value** from **each group** of records.

If you want to write (syntax) a correct "**GROUP BY**" clause, you may click *F.10.14*.

F.10.11 What is the "PIVOT" clause of a Transform?

In a **Transform** operation (click *F.10.1*), the mandatory "**PIVOT**" clause determines the "**PIVOT**" **fields** (click *F.10.3*), the "**PIVOT**" **field names** (click *F.10.3.2*) and the "**PIVOT**" **field order** (click *F.10.3.3*), as follows:

```
PIVOT PIVOT-field-names-expression(Input-field-names)
```

The "**PIVOT**" clause contains the **expression** that produces the "**PIVOT**" **field names** (click *F.10.3.2*), **unless** the optional "**IN**" clause (click *F.10.12*) is used. If the optional "**IN**" clause is used, then the "**PIVOT**" **field names** are produced by the **list** of **values** in the "**IN**" clause.

The "**PIVOT**" **expression** is built using the "**Input-field-names**" combining them with functions (excluding SQL aggregate), value operators and constants.

If you want to write (syntax) a correct "**PIVOT**" clause, you may click *F.10.14*.

F.10.12 What is the "IN" clause of a Transform?

In a **Transform** operation (click *F.10.1*), the optional "**IN**" clause determines the "**PIVOT**" **fields**, the "**PIVOT**" **field names** (click *F.10.3.2*) and the "**PIVOT**" **field order** (click *F.10.3.3*), as follows:

```
IN ( List-of-PIVOT-values )
```

The "**IN**" keyword is followed by a **list** of **constants**. I will call this **list** the "**IN**" **list**. The "**IN**" **list** is enclosed between parentheses and the **constants** are separated with commas.

The "**IN**" clause is most frequently used to specify the **order** of the "**PIVOT**" **fields** and/or to **discard** some of them. The way to do this is just by writing in the "**IN**" **list** the "**PIVOT**" **field names** that you want (i.e., excluding some if you do not want them), and writing them in the **specific order** that you want them.

In the general case, when the "**IN**" clause is used, the "**PIVOT**" **fields**, their **field names** and their **field order** will be **exactly the ones** and with the **same order** as they appear in the "**IN**" clause (except if they are the same as a "**SELECT**" **field name**). I now explain the possible cases in more detail:

- If the "**IN**" **list** **does not** include a **value** that is produced by the "**PIVOT**" **expression**, then the "**PIVOT**" **field** corresponding to that **value** **does not** appear in the **output** of the **Transform**.
- If the "**IN**" **list** **includes** a **value** that is produced by the "**PIVOT**" **expression**, **and** that **value** (converted to ***String***) is **not** one of the "**SELECT**" **field names**, then the corresponding "**PIVOT**" **field** will be in the **output** of the **Transform**, in the order indicated in the "**IN**" **list**, with the corresponding **field values** produced by the "**TRANSFORM**" **expression**.
- If the "**IN**" **list** **includes** a **value** that is **not** produced by the "**PIVOT**" **expression**, **and** that **value** (converted to ***String***) is **not** one of the "**SELECT**" **field names**, then that **value** (converted to ***String***) will be a "**PIVOT**" **field**, in the order indicated in the "**PIVOT**" **list**, with all its values being **Null**.
- If the "**IN**" **list** **includes** a **value** that (converted to ***String***) **is one of** the "**SELECT**" **field names**, then a "**PIVOT**" **field** will be produced with the name "**FieldN**", where "**N**" is an integer value assigned by MS-Access. The order of that field is the one of the corresponding **value** in the "**IN**" **list**. The **values** of this field will be exactly the same ones as the ones of "**SELECT**" **field** whose **name** was the same.

The "**PIVOT**" **field name** corresponding to a **Null** value (i.e., "**PIVOT**" **field name** "**<>**") can also be reordered by including it in the "**IN**" clause. I strongly recommend that you **do not use** a "**PIVOT**" **expression** that produces the string "**<>**", to avoid confusion with a "**PIVOT**" **field name** arising from **Null** (which most likely will be unintentional).

The "**IN**" **list** **cannot** contain **duplicated** values: if it does, the Query will **crash** with a syntax error message.

As I indicated at the beginning of this section, the "**IN**" clause is most frequently used

to specify the **order** of the "**PIVOT**" **fields** and/or to **discard** some of them. As I have explained just above, the "**IN**" clause can **also** be used to add **copies** of "**SELECT**" **fields** and/or to **create new** "**PIVOT**" **fields**. However, the two features in the previous sentence do not seem very useful to me, because the added fields would have either replicated **values** or **Null** (respectively), in **all** the **output** records.

If you want to write (syntax) a correct "**IN**" clause, you may click *F.10.14*.

F.10.13 How do the clauses from Transform and Select compare?

To better understand **Select** and **Transform**, I think it is useful to compare the characteristics of the clauses of a **Select** operation and of a **Transform** operation:

- In the **Transform** (mandatory) "**SELECT**" clause, each "**SELECT**" **expression** may include as its **elements** the "**PIVOT**" **expression**, in addition to the **elements** in the (mandatory) "**SELECT**" clause of a **Select-group_by_aggreg**. This allows to use the "**PIVOT**" **field names** in the "**SELECT**" **expressions**.
  Remind that in a **Select-group_by_aggreg**, the "**SELECT**" **expressions** can contain **other** "**Output-field-names**" (as long as you **do not** create a **circular** reference, click *F.7.14*), any number of "**GROUP BY**" **expressions**, and any number of **SQL aggregate functions** each having as argument its specific **expression** over the "**Input-field-names**" and **other** "**Output-field-names**" (as long as you **do not** create a **circular** reference, **nor** a **nested** SQL aggregate function, click *F.7.14*).
- The **Transform** (optional) "**DISTINCT**" clause does not produce any effect in the results of the **Transform** operation, while in the **Select** operation the (optional) "**DISTINCT**" clause removes **all** the redundant **output** duplicate records.
- The **Transform** (optional) "**ORDER BY**" clause can only use as its **expressions** the "**GROUP BY**" **expressions** (over the **Input-field-names**) or **the** "**PIVOT**" **expression** (over the **Input-field-names**). Aside from this, its functionality is the same as the (optional) "**GROUP BY**" clause from a **Select**.
- **Transform** does **not** have the (optional) "**HAVING**" and "**TOP**" clauses, that the **Select** operator has.
- **Select** does **not** have the (mandatory) "**TRANSFORM**" and "**PIVOT**" clauses nor the (optional) "**IN**" clause, that the **Transform** operator has.
- Both **Transform** and **Select** have the (**not advisable**) (optional) "**DISTINCTROW**" clause.

F.10.14 How do I write a correct (syntax) Transform?

You may click:

- "*F.10.14.1 What is a syntax-example of a Transform?*"
- "*F.10.14.2 What are the formal rules (syntax) to write a Transform?*"
- "*F.10.14.3 Can I nest Transform operations?*"

Lightning Guide to Databases with Microsoft Access and SQL

© Prof. Dr. Arturo Azcorra

August 2021 – First edition v1.0

Buy me at: lightningguide.net

PARTS IN THIS LIGHTNING GUIDE

Copyright warning!

PART N. CONTENTS AND ACKNOWLEDGEMENTS

Table of Contents

Lightning Guide to MS Access & SQL

A practical guide to Database Design with MS Access and SQL

Prof. Arturo Azcorra