FlexibleSearch Samples

This document discusses a number of FlexibleSearch samples. As the FlexibleSearch is a key component of SAP Commerce, reading this document is recommended for all developers.

This document does not discuss the FlexibleSearch in general, please refer to FlexibleSearch for general information on FlexibleSearch.

Basic SELECT Statements

This section discusses FlexibleSearch statements with one single **SELECT** operator. Some of these samples overlap in terms of operators, this is hard to avoid for such a subject.

SELECT Statements with Negation

The following FlexibleSearch statements are samples for using the negation operator **NOT**.

- **Getting all Products whose code is not empty**

  The following FlexibleSearch statement returns Primary Keys (PK) of every **Product** whose **code** attribute is different from **null**. Be aware that in SQL syntax the empty string **""** is not considered to be **null**. In other words: the following FlexibleSearch statement finds **Products** whose code is set to **""**.

```
SELECT {p.pk} FROM {Product AS p} WHERE {p.code} IS NOT NULL
```

- **Getting all Categories whose code does not contain a certain string**

  The following FlexibleSearch statement returns the PKs of every **Category** whose **code** attribute does not contain the string **test**.

```
SELECT {c:pk} FROM {Category AS c} WHERE {c:code} NOT LIKE '%test%'
```

SELECT Statements with Several Return Columns

The arguments passed for the **SELECT** operator specify the columns from the database that are to be returned by the FlexibleSearch query.

- **Returning every database column of every Category**

  The following FlexibleSearch statement returns every database column of every Category in SAP Commerce. The list of returned database columns could be something along the lines of: **hjmpts**, **modifiedts**, **createdts**, **typepkstring**, **pk**, **ownerpkstring**, **aclts**, **propts**, **p_showemptyattributes**, **p_normal**, **p_thumbnails**, **p_revision**, **p_code**, **p_data_sheet**, **p_logo**, **p_catalogversion**, **p_picture**, **p_detail**, **p_catalog**, **p_others**, **p_order**, **p_thumbnail**, and **p_externalid**.

```
SELECT * FROM {Category}
```

- **Getting the point of time when a Category was last modified, Category code and PK**

  The following FlexibleSearch statement returns three database columns of every **Category** in SAP Commerce. Note that the **code** attribute is enclosed by curly braces.

```
SELECT {cat:modifiedtime}, {cat:code}, {cat:pk} FROM {Category AS cat}
```

SELECT Statements Over Several Attributes

A FlexibleSearch statement allows narrowing down the list of search results by specifying several attributes in a search condition. A SAP Commerce item must match the search condition in the respective attribute to become included in the search result list. For example, if the FlexibleSearch statement queries for the **code** and the **name** attribute, the list of search results contains only items that have a matching value in both the **code** and the **name** attribute.

- **Getting all Products whose code or name contains a certain string**

  The following FlexibleSearch returns the PKs of all Products whose **code** attribute or **name** attribute contains a string that ends with **myProduct**.

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:code} LIKE '%myProduct'
      OR {p:name} LIKE '%myProduct'
   ORDER BY {p:code} ASC
```

  The percent sign (**%**) works as a wildcard character:

  - **a%** finds all strings that start with an **a**,
  - **%a** finds all strings that end with an **a**,
  - **%a%** finds all strings that contain an **a**.

    By introducing a parameter (**?name** in the following FlexibleSearch query) into the query, you can search for any search string:

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:code} LIKE ?name
      OR {p:name} LIKE ?name
   ORDER BY {p:code} ASC
```

    Be aware that the search condition **`WHERE LIKE ?name`** only finds products whose **name** attribute matches the value of the **?name** parameter exactly. To find **Products** whose **name** attribute contains the value of the **?name** parameter only partially, you need to use wildcard characters, as in **`WHERE LIKE CONCAT('%', CONCAT(?name, '%'))`** or enclosing parameter by wildcard characters like:

    `...LIKE ?name; query.addQueryParameter("name", "%h%")` **. Both solutions are SQL-injection safe.**

SELECT Statements Over Several Languages

SAP Commerce allows for attributes to be localized, that is, to have an individual value for each language in SAP Commerce. By specifying the language code enclosed by square brackets (**[** and **]**, respectively) after the attribute name, such as **`{description[de]}`**. Not specifying a language code for a localized attribute causes SAP Commerce to use the default language for the current session.

- **Getting all Products with an empty name in the current SAP Commerce language**

  The following FlexibleSearch query returns the PKs of all **Products** whose **name** attribute is not set (**IS NULL**). The **name**attribute is localized, but the FlexibleSearch query does not specify a language explicitly, therefore the FlexibleSearch defaults to the current session language.

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:name} IS NULL
```

- **Getting all Products with an empty name in German or an empty description in English**

  The following FlexibleSearch query returns the PKs of all **Products** and whose...

- **name** attribute is not set for the German language (**IS NULL**) or
- **description** attribute is not set for the English language (**IS NULL**).

Both the **name** attribute and the **description** attributes are localized. The FlexibleSearch query specifies the language explicitly (via de and en, respectively). If no language is specified, the FlexibleSearch would default to the current session language.

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:name[de]} IS NULL
      OR {p:description[en]} IS NULL
```

- **Searching several languages at once**

  By specifying different language codes, you can search localized attributes in various languages at a time. The following FlexibleSearch statement searches both the English and German values of the **description** attribute:

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:description[en]:o} LIKE '%text%'
   OR {p:description[de]:o} LIKE '%text%'
```

Here, **:o** (outer join) parameter is used to include matches with missing rows in the **ProductsLP** table (=. the table that holds localized products) as well. Otherwise , the query would only return products with an existing row in **ProductsLP** table, because it would only use JOIN.

> **Tip**
>
> **Add OR Clause to Search Additional Attributes or Languages**
>
> To search another language, add the attribute to be searched with an explicit specification of the language to be searched to the **WHERE** clause via an **OR** clause. For example, the following FlexibleSearch statement searches the **description** attribute of the **Product** in the three languages English, German, and French (**en**, **de**, and **fr**, respectively) and the **name** attribute in German:
>
> ```
> SELECT {p:PK}
>    FROM {Product AS p}
>    WHERE {p:description[en]:o} LIKE '%text%'
>    OR {p:description[de]:o} LIKE '%text%'
>    OR {p:name[de]:o} LIKE '%text%'
>    OR {p:description[fr]:o} LIKE '%text%'
> ```

It is also possible to replace the hard-coded search string with a parameter. Please refer to the SELECT statements with parameters section below for details.

SELECT Statements with Parameters

A parameter in a FlexibleSearch query allows inserting varying search patterns. This is a common field of use for applications where one single query is intended to be used for various searches, such as:

- - Search fields in the store frontend.
  - Search fields in Backoffice.
  - Item retrieval in the application business code.
- **Using one parameter in a FlexibleSearch statement**

  The following FlexibleSearch statement queries the **description** attribute in three SAP Commerce languages (**en**, **de**, **fr**) for one single parameter, **?param**. In other words, the FlexibleSearch finds all **Product** instances whose description in English, German, or French contains the search pattern specified by **?param**.

  ```
  SELECT {p:PK}
     FROM {Product AS p}
     WHERE {p:description[en]:o} LIKE ?param
     OR {p:description[de]:o} LIKE ?param
     OR {p:description[fr]:o} LIKE ?param
  ```

  Note, that there are **:o** characters in **WHERE** clause. They are used to force the related table to be outer-joined (in case of localized properties the xxxLP table).
- **Using two parameters in a FlexibleSearch statement**
  - Getting every **Product** that is in at least one of two **Categories**:

    ```
    SELECT {cpr:target}
       FROM {CategoryProductRelation AS cpr}
       WHERE {cpr:source} LIKE ?param1
          OR {cpr:source} LIKE ?param2
    ```
  - Getting every **Product** that was changed between two dates:

    ```
    SELECT {pk}
       FROM {Product}
       WHERE  {modifiedtime} >= ?startDate
          AND {modifiedtime} <= ?endDate
    ```

SELECT Statements with Concatenation

The FlexibleSearch feature allows concatenating strings within a statement. Be aware that each **CONCAT** operator call allows only two

parameters. To concatenate more than two parameters, you need to run more than one **CONCAT** operator calls.

- **Enclosing a search string by percent signs %**

  The following FlexibleSearch statement gives an example on the **CONCAT** operator by concatenating the leading **%** character with the concatenation of **myProduct** and the **%** character for a total of **%myProduct%**:

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:description[de]}
      LIKE
         CONCAT(
            '%',
            CONCAT(
               'myProduct',
               '%'
            )
         )
   OR {p:description[en]}
      LIKE
         CONCAT(
            '%',
            CONCAT(
               'myProduct',
               '%'
            )
         )
   ORDER BY {p:code} ASC
```

  This function is useful in combination with parameters, as in the following FlexibleSearch statement:

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:description[de]} LIKE
      CONCAT(
         '%',
         CONCAT(
            ?param,
            '%'
         )
      )
   OR {p:description[en]} LIKE
      CONCAT(
         '%',
         CONCAT(
            ?param,
            '%'
         )
      )
   ORDER BY {p:code}
```

SELECT Statements with DISTINCT Operator

The **DISTINCT** operator makes sure that duplicate results are returned only once. Duplicate return results may occur from sub-selects, **JOIN** clauses or from identical parameters, for example.

- **Finding every Product that is in at least one of two given Categories**

  The following FlexibleSearch statement returns every **Product** that is assigned to at least one of the two **Categories** provided by the two parameters **?param1** and **?param2**. The **DISTINCT** operator ensures that every **Product** is returned only once even if it were assigned to both **Categories**.

```
SELECT DISTINCT {cpr:target}
   FROM {CategoryProductRelation AS cpr}
   WHERE {cpr:source} LIKE ?param1
      OR {cpr:source} LIKE ?param2
```

SELECT Statements with GROUP BY Operator

- **Getting every Product which has been ordered, grouped by the Product**

```
SELECT {oe:product}
FROM {OrderEntry AS oe}
GROUP BY {oe:product}
```

Subselects

A subselect is a **SELECT** statement within a **SELECT** statement. Via a subselect, a **SELECT** statement can affect (narrow down or expand, for example) a search result list. The basic syntax looks as follows:

```
SELECT *
   FROM ${type}
   WHERE

   {{
      SELECT *
         FROM ${other_type}
         WHERE ${subselect_search_condition}
   }}
```

Note

Subselects in **FROM** clause of the query are also allowed (the example above presents subselect in the WHERE clause of the query). See Subselect with Parameters section below for more details.

Subselect Over Several Types

- **Getting every Product that has a directly or indirectly assigned PriceRow**

  The following FlexibleSearch statement lists every Product that:

  - has at least one **DiscountRow** directly assigned to it (**subselect 1**) or
  - is assigned to a **ProductDiscountGroup** that thas a **DiscountRow** assigned to it (**subselect 2**)

```
SELECT DISTINCT {p:PK}, {p:name}, {p:code}
   FROM {Product AS p}
   WHERE {p:PK} IN
   (
      {{
         -- subselect 1
         SELECT {dr:product}
            FROM {DiscountRow AS dr}
      }}
   )
   OR {p:PK} IN
   (
      {{
         -- subselect 2
         SELECT {prod:PK}
            FROM
               {
                  Product AS prod
                     LEFT JOIN DiscountRow AS dr
                     ON {prod:Europe1PriceFactory_PDG} = {dr:pg}
               }
            WHERE {prod:Europe1PriceFactory_PDG} IS NOT NULL }}
   )
   ORDER BY {p:name} ASC, {p:code} ASC
```

- **Getting every Product that is in at least 3 Categories**

  The following FlexibleSearch statement returns every **Product** that is in more than three **Categories** (specified by the **WHERE howmany > 3** clause in **subselect 1**). The **subselect 2** returns the number of categories a **Product** is in (via searching the **CategoryProductRelation**). The **subselect 1** returns only those products which are in more than three **Categories** (**WHERE howmany >3**).

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:PK} IN
   (
```

```
   -- subselect 1
   SELECT prod
      FROM
         (
            {{
               -- subselect 2
               SELECT {cpr:target} AS prod, count({cpr:target}) AS howmany
                  FROM {CategoryProductRelation AS cpr}
                  GROUP BY {cpr:target}
            }}
         )
         temptable
         WHERE howmany > 3
)
ORDER BY {p:name} ASC, {p:code} ASC
```

Note

Add a Parameter for Flexibility

By replacing the hard-coded **3** with a parameter, you could use the statement to find every **Product** that is in a specified number of categories, such as:

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:PK} IN
   (
      -- subselect 1
      SELECT prod
         FROM
            (
               {{
                  -- subselect 2
                  SELECT {cpr:target} AS prod, count({cpr:target}) AS howmany
                     FROM {CategoryProductRelation AS cpr}
                     GROUP BY {cpr:target}
               }}
            )
            temptable
            WHERE howmany > ?number
   )
   ORDER BY {p:name} ASC, {p:code} ASC
```

Please also refer to the Subselect with Parameters section below for additional information.

Subselect with Parameters

- **Getting all Products ordered on or after a certain date**

  The following FlexibleSearch statement returns every **Product** that was ordered on or after a certain date. Via **subselect 2**, the FlexibleSearch statement retrieves every **Order** that was created on or after the value specified by the **?date** parameter. Of these search results, **subselect 1** retrieves the OrderEntries that belong to these **Orders**. The outermost **SELECT** statement gets the **Products** referred by the **OrderEntries**.

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:PK} IN
   (
      {{
         -- subselect 1
         SELECT DISTINCT {oe:product}
            FROM {OrderEntry AS oe}
            WHERE {oe:order} IN
            (
               {{
                  -- subselect 2
                  SELECT {o:PK}
                     FROM {Order AS o}
                     WHERE {o:date} >= ?date
               }}
            )
      }}
)
```

- **Getting every Product without a PriceRow in a specified Currency**

  The following FlexibleSearch statement returns every **Product** that does not have a **PriceRow** assigned for the specified **Currency**. The subselect returns every **PriceRow** for the specified currency. This search result is then **negated** via the outer FlexibleSearch statement, which returns every **Product** that is not included in the search results that are returned by the subselect.

```
SELECT {p:PK}
   FROM {Product AS p}
   WHERE {p:PK} NOT IN
   (
      {{
         -- subselect
         SELECT {pr:product}
            FROM {PriceRow AS pr}
            WHERE {pr:currency} = ?currency
      }}
   )
   ORDER BY {p:name} ASC, {p:code} ASC
```

- **Reporting query with subselect in FROM clause and SQL aggregate functions.**

  This query calculates average **Order** value and average **Order** unit count within specified date range.

  Result of this query is a pair of numeric values. The first one is the average **Order** value, and the second one is the average order unit count.

  The term **unit count** of the order means the sum of quantities of the order entries. For example if an order consists of:

  - 1 red T-shirt
  - 1 blue T-shirt
  - 2 yellow T-shirt

  Then the order unit count for this order is 4.

```
SELECT AVG(torderentries.totprice), AVG(torderentries.totquantity)
   FROM (
   {{
      SELECT SUM({totalPrice}) AS totprice, SUM({quantity}) AS totquantity FROM {OrderEntry}
         WHERE {creationtime} >= ?startDate AND {creationtime} < ?endDate GROUP BY {order}
   }}
   ) AS torderentries
```

Combined SELECT Statements with UNION Operator

The following FlexibleSearch statement uses the **UNION** operator to retrieve the set of results for two **SELECT** statements:

```
SELECT x.PK
FROM
(
{{SELECT {PK} as PK FROM {Chapter}
WHERE {Chapter.PUBLICATION} LIKE 6587084167216798848
}}
UNION ALL
{{
SELECT {PK} as PK FROM {Page}
WHERE {Page.PUBLICATION} LIKE 6587084167216798848
}}) x
```

Combined SELECT Statements with UNION Operator