GMQL Introduction to the language

Introduction

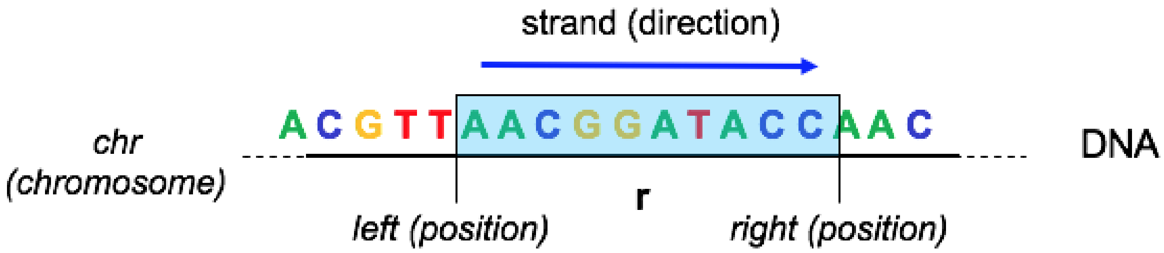
This document contains a list of reference instructions for using all GMQL basic operators, together with relevant examples of single statements and notable combination of them. It also showcases some examples of more complex GMQL queries involving various operators (COVER, MAP, JOIN), all of which have discernible biological meaning.

After a short presentation of the Genomic Data Model adopted by GMQL (Section A), the document contains a description of the basic operators of the language (Section B). After listing syntactic conventions, Section B reports the list of all operators in GMQL along with their parameters, syntax and general usage. Each operator also contains a list of basic examples, showcasing how to combine different parameters and describing the associated semantics.

A. GENOMIC DATA MODEL (GDM)

GMQL is based on a representation of the genomic information known as GDM - Genomic Data Model. **Datasets** are composed of **samples**, which in turn contains two kinds of data:

1. **Region values** (or simply **regions),** aligned w.r.t. a given reference, with specific left-right ends within a chromosome:



   Regions can store different information regarding the "spot" they mark in a particular sample, such as region length or statistical significance. Regions of the model describe processed data, e.g. mutations, expression or bindings; they have a **schema**, with 5 common attributes (*id*, *chr*, *left*, *right*, *strand*) including the id of the region and the region coordinates, along the aligned reference genome, and then arbitrary typed attributes. This provides interoperability across a plethora of genomic data formats;

2. **Metadata**, storing all the knowledge about the particular sample, are arbitrary attribute-value pairs, independent from any standardization attempt; they trace the data provenance, including biological and clinical aspects.

B. BASIC OPERATORS

After illustrating GMQL syntactic conventions, this section reports the list of all operators in GMQL along with their parameters, syntax and general usage. Each operator also contains a list of basic examples, showcasing how to combine different parameters and describing the associated semantics.

Within this document, by convention, operator parameters and conditions which are **mandatory** are written in **bold**.

Foreword: Syntactic conventions and other observations

- **Region** and **metadata attribute names**, as well as **dataset names**, are **case sensitive**: for instance, pvalue != pValue != PVALUE. **GMQL keywords**, however, are **not case sensitive**: e.g. UPSTREAM == upstream == UpStReAm;
- **Logical predicates on metadata** consist of concatenations of atomic predicates by means of the OR, AND and NOT logical operators. Atomic predicates have the following general form: attribute_name (==, o >, or <, or >=, or <=) value. For **region attributes**, if the value is a numeric literal, it is automatically parsed to the related numeric format. Standard numeric ordering is used in order to evaluate >, < and the LIKE; otherwise, if the region attribute value is a string literal, a lexicographic order is used;
- In all operators that allow for **metadata comparisons**, the language recognizes **substrings of metadata attribute names** (for instance, 'age' in 'LEFT.age'). So, if a metadata selection or meta-join is made, the language searches for all metadata which contain the queried attribute name substring with the queried attribute value, e.g. the query *age == '45'* selects samples with metadata 'LEFT.*age 45*" or 'P1.LEFT.*age 45*', but also with metadata 'day_of_mari*age 45*';
- In all operators that allow the *groupby* and *joinby* option, it is possible to express only the suffix part of a (metadata or region) attribute name.
- Assigning different values to an existing variable (i.e. dataset) name is not allowed by the language;
- The following are all the **aggregate functions** available for GMQL operators:
  - COUNT (requires no argument, counts number of regions, and is computed by default in the MAP operation);
  - BAG (applicable to attributes of any type, creates comma-separated strings of distinct attribute values);
  - SUM, AVG (average), MIN, MAX, MEDIAN, STD (standard deviation) (applicable to attributes of numeric types).
- In GMQL queries **comments** can be introduced only as an entire line, starting with the character #.

- **Note**: the only operator in the current release which allows to **edit region coordinates** as region attributes is the PROJECT operator.
- **Note:** it is possible to use the PROJECT command to copy region coordinates into separate region attributes and employ them as any other attribute (including use in aggregations).

- When evaluating the effect of every GMQL operator, **all t**he following **five characteristics** of the output dataset must be considered:
  1. **Number of samples** (called *dataset cardinality*), based on input dataset(s) cardinality;
  2. **Dataset schema**, i.e. region attributes;
  3. **Samples region coordinates**, based on input regions (also when these overlap in a single sample);
  4. **Samples region attribute values**;
  5. **Samples metadata.**

1) SELECT

The SELECT operation creates a new dataset from an existing one (considering also an additional dataset if a semijoin clause is specified, see below) by extracting a subset of samples from the input dataset; each sample in the output dataset has the same region attributes and metadata as in the input dataset.

The general syntax for SELECT is:

DS_{out} = SELECT(p_m ; region: p_r ; semijoin: $p_{sj}(DS_{ext})$) DS_{in} ;

where

- DS_{in} is the input dataset;
- DS_{out} is the resulting output dataset;
- p_m is a logical predicate on metadata;
- p_r is a logical predicate on genomic regions within each sample in DS_{in} ;
- $p_{sj}(DS_{ext})$ is a semi-join predicate, with form: $attr_1$, $attr_2$, ..., $attr_N$ ***IN*** (or *NOT **IN***) DS_{ext}

This operation (hereafter called selection) can therefore be based on three kinds of criteria, of which at least one must be specified:

1. Metadata predicates: selection based on the existence and values of certain metadata attributes in each sample. For instance, *antibody_target* == *'POLR2A'* will extract only samples whose metadata contain the attribute *antibody_target* with associated value *POLR2A*. In predicates, attribute-value conditions can be composed using logical predicates AND, OR and NOT; in the latter case attribute-value conditions must be within parentheses, e.g. *NOT(antibody_target == 'POLR2A')*;
2. Region predicates: selection based on the characteristic of the genomic regions of each sample. For instance, *strand* == + will extract only samples that include regions with *strand* attribute (defined in the dataset schema) *equal* to + and only those regions whose strand is *equal to* +. Notice that the use of metadata attributes in predicates on region attributes is enabled. For instance, a condition such as AccIndex == META(maxCount) can be used;
3. Semi-join clauses: selection based on the existence of certain metadata attributes and the matching of their values with those associated with at least one sample in an external dataset DS_{ext}. In particular, a semi-join predicate in the form a_1, a_2,. .., a_N IN DS_{ext} is true for a given sample s^k_{in} of DS_{in} if and only if there exists at least one sample in dataset DS_{ext} with metadata attributes named a_1, a_2,. .., a_N and these attributes of DS_{ext} has at least one value in common with the same attributes a_1, a_2,. .., a_N in s^k_{in}. For instance, s*emijoin: cell, antibody_taget IN CTCF_RAW* will extract only those samples of DS_{in} that have both cell and antibody_taget values found in at least one sample of CTCF_RAW. NOT IN condition is evaluated accordingly.

Clearly, a user might define complex SELECT statements with more than one selection type at the same time: if this happens, it is intended that clauses of heterogeneous type are connected by an AND condition.

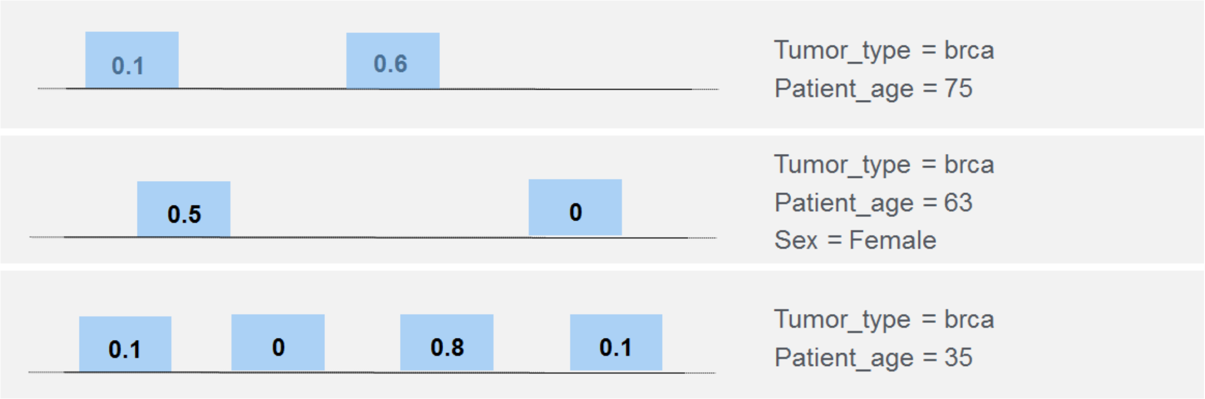
Note 1: SELECT() DS_{in} selects all samples in dataset DS_{in} and copies them in the output.

Note 2: The wild char '*' can be used in a SELECT statement to indicate all values of an attribute, e.g. SELECT(NOT(*attribute_name* == '*')) DS_{in} selects all samples in dataset DS_{in} which do not include in their metadata the attribute named *attribute_name* (with any value) and copies such samples in the output.

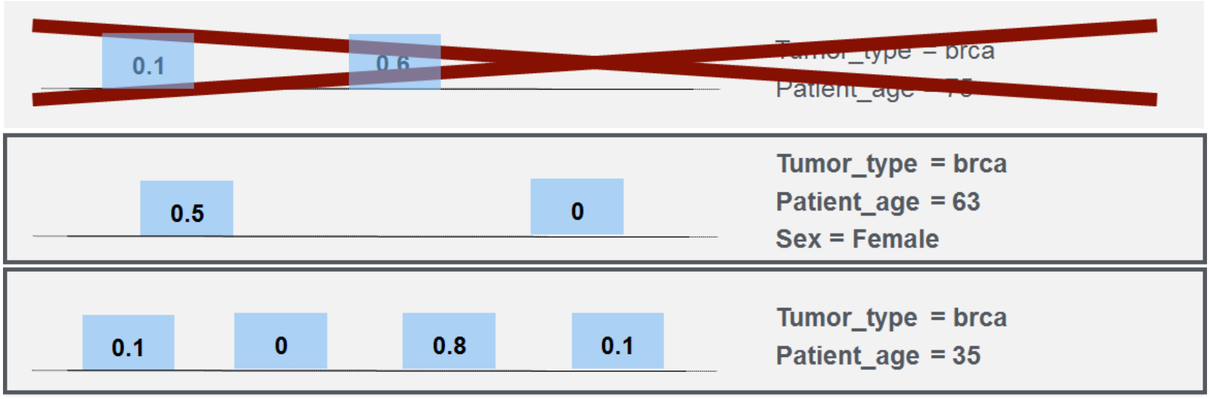
Example 1:
OUTPUT_DATASET= SELECT(Patient_age < 70) INPUT_DATASET;

This GMQL query selects from INPUT_DATASETdata samples of patients younger than *70* years old, based on filtering on sample metadata attribute *Patient_age* (see following figure).

INPUT_DATASET:



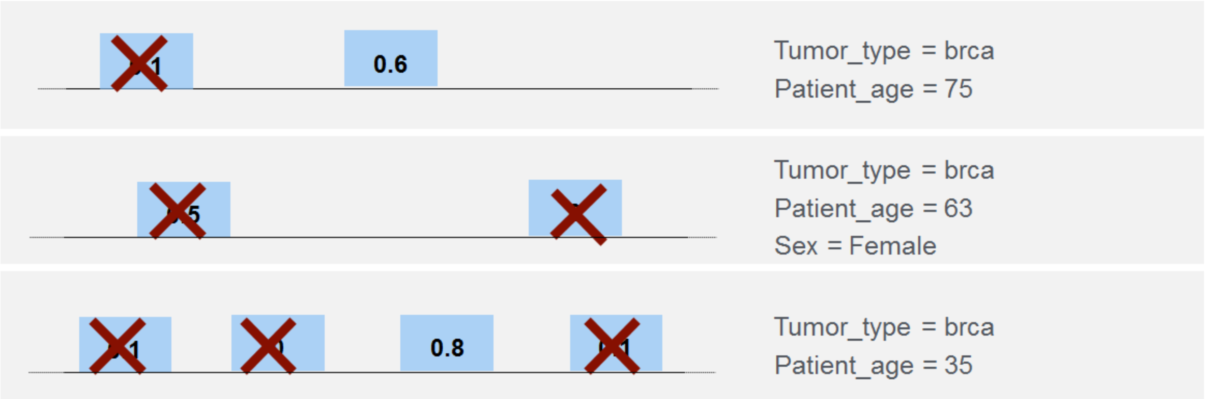
OUTPUT_DATASET:



Example 2:
OUTPUT_DATASET = SELECT(region: score > 0.5) INPUT_DATASET;

This query selects, in all samples in INPUT_DATASET, those regions which have a value greater than *0.5* for their attribute *score*. The resulting OUTPUT_DATASET contains a copy of the samples of INPUT-DATASET, with the same metadata, but with only the remaining regions, in case none.

OUTPUT_DATASET:



<u>Example 3</u>*:*
DATA = SELECT(cell == ‘Urothelia’; region: left > 100000) HG19_ENCODE_NARROW;

This GMQL statement creates a new output dataset DATA which only includes samples from the input dataset HG19_ENCODE_NARROW that present the metadata attribute-value pair *(cell Urothelia)*. Moreover, in each sample, only the regions whose left coordinate value is greater than *100000* are included in the output dataset DATA.

<u>Example 4:</u>
DATA = SELECT(region: NOT(variant_type == 'SNP')) HG19_TCGA_dnaseq;

This statement produces a dataset that contains all the samples of the input dataset (with all their metadata) which have at least one region which has not of *variant_type* ‘*SNP*’. Inside each sample the regions that, for the region attribute *variant_type*, have a value different from “SNP” (Single-Nucleotide Polymorphism) are preserved; the regions that have *variant_type* ‘*SNP*’, instead, are excluded.
Note that in case all regions belonging to a sample have been excluded through the NOT condition, that empty sample is not produced in the output dataset.

<u>Example 5:</u>
DATA = SELECT(manually_curated|tissue_status == "tumoral" AND
    (manually_curated|tumor_tag == "gbm" OR manually_curated|tumor_tag == "brca"))
    HG19_TCGA_dnaseq;

This statement shows how it is possible to combine multiple conditions on the metadata attributes by using the boolean operators AND and OR. In this particular case, the output dataset contains all the samples that have *manually_curated|tissue_status* = "*tumoral*" and *manually_curated|tumor_tag* = "*gbm*", and also all the samples that have *manually_curated|tissue_status* = "*tumoral*" and *manually_curated|tumor_tag* = "*brca*". Notice that *gbm* corresponds to data concerning patients with *Glioblastoma multiforme*, instead *brca* refers to *Breast Invasive Carcinoma*.

<u>Example 6</u>*:*
JUN_POLR2A_TF = SELECT(antibody_target == ‘JUN’; region: pValue < 0.01;
    semijoin: cell NOT IN POLR2A_TF) HG19_ENCODE_NARROW;

This statement creates a new dataset called JUN_POLR2A_TF by selecting those samples and their regions from the existing HG19_ENCODE_NARROW dataset (a collection of narrowPeak data from the ENCODE repository) such that:

A. each output sample has a metadata attribute called *antibody_target* with value *JUN*;
B. each output sample also has not a metadata attribute called *cell* that has the same value of at least one of the values that a metadata attribute equally called *cell* has in at least one sample of the POLR2A_TF dataset;
C. for each sample satisfying A and B, only its regions that have a region attribute called *pValue* with the associated value less than *0.01* are conserved in output.

Example 7:
T = SELECT(region: score > META(avg_score)) GRCh38_ENCODE_BROAD_MAY_2017;

This statement allows to select all those regions for which the region attribute *score* has a value which is greater than the metadata attribute value *avg_score*.

2) PROJECT

The PROJECT operator creates, from an existing dataset, a new dataset with all the samples in the input one, but keeping for each sample in the input dataset only those metadata and/or region attributes expressed in the operator parameter list. Region coordinates and values of the remaining metadata and region attributes remain equal to those in the input dataset. Differently from the SELECT operator, PROJECT allows to:

- Remove existing metadata and/or region attributes from a dataset;
- Create new metadata and/or region attributes to be added to the result.

The general syntax for PROJECT is:

DS_{out} = PROJECT($RA_1, \dots, RA_m$;
metadata: MA_1, ..., MA_n ;
region_update: NR_1 AS $g_1, \dots, NR_h$ AS g_h;
metadata_update: NM_1 AS f_1, ..., NM_k AS f_k) DS_{in} ;

where:

- DS_{in} is the input dataset;
- DS_{out} is the resulting output dataset;
- $RA_1, \dots, RA_m$ are the conserved genomic region attributes;
- MA_1, ..., $MA_{n;}$ are the conserved metadata attributes;
- NR_1, ..., $NR_{h;}$ are new genomic region attributes generated using functions $g_1, ..., g_h$ on existing region attributes;
- NM_1, ..., $NM_{k;}$ are new metadata attributes generated using functions $f_1, ..., f_k$ on existing metadata attributes.

Note 1: The default form of this operator has no parameter. PROJECT() DS_{in} applies the projection only on the regions. It removes all the region fields which are not coordinates (i.e., *chr*, *start*, *stop*, and *strand* are kept).

Note 2: It is possible to use the special keywords ALLBUT to retain all existing genomic region attributes apart from a specified set.

Note 3: If the names of existing region attributes are used in place of new region names, the operation updates such region attributes to the new specified values.
To specify the new values, the following options are available:
- All aggregation functions already defined;
- All basic mathematical operations (+ - * /), including usage of parenthesis;
- Whenever possible, the metadata values are cast to numeric. If the cast fails (i.e. the metadata value is a not castable string) the resulting metadata should contain "GMQL Casting Exception: Could not parse".

Note 4: To express which set of region attributes should be considered, the wildcard "?" can be used in the RA_i place of the syntax (at most one per attribute). For instance, the user can write statements such as:

```
D = PROJECT(?.score) DATASET;
E = PROJECT(?.score, ?.name) DATASET;
F = PROJECT(DS.?) DATASET;
G = PROJECT(my.?.score) DATASET;
H = PROJECT(S.?, ?.att, S.?.att);
```

Note that PROJECT(?.S.?) is incorrect.

Note 5: It is possible to create a new textual region attribute with a defined value, e.g. RES = PROJECT(region_update: label AS "class1") INPUT;

Example 1:

```
RES = PROJECT(region_update: length AS right - left) DS;
```

This GMQL statement creates a new dataset called RES by preserving all region attributes and creating a new region attribute called *length* by subtracting the left coordinate of a region from its right coordinate. This simple operation computes the length of the region in terms of number of bases. Notice that the length is always positive regardless of the strand of the region, because *right* and *left* coordinates already take into account the direction. Also notice that, in case DS was a dataset of gene TSS, all the new length attributes would turn out to be unitary.

Example 2:

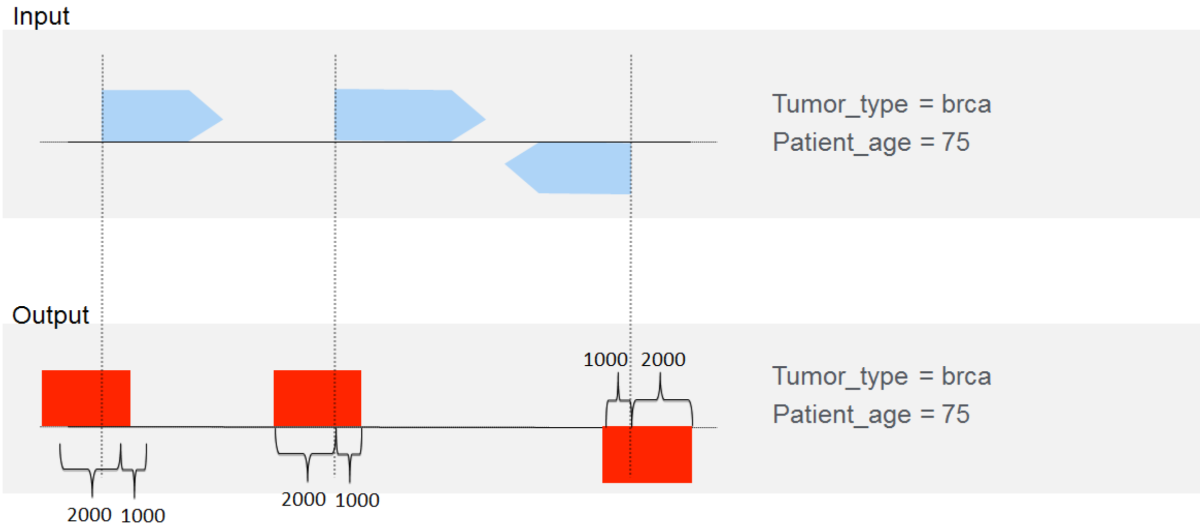
```
RES = PROJECT(region_update: new_right AS right) DS;
```

This GMQL statement creates a new dataset called RES by preserving all region attributes and creating a new region attribute called *new_right* by which contains a copy a the values of the coordinate attribute *right*. This allows to subsequently aggregate regions by their right coordinate value using the *new_right* attribute.

Example 3:

```
RES = PROJECT(region_update: START AS START -  2000,
                                              STOP AS START + 1000) GENES;
```

The first PROJECT statement considers an input dataset of genes. To define a promotorial region it is necessary to start from a transcription start site (TSS) (a single base of DNA at the beginning of the gene, conventionally taken as a starting point for the gene transcription) and extend it upstream/downstream by a number of given bases. As an example, here, the region coordinate attributes *left* and *right* are redefined by shifting them of 2kbs upstream and 1kbs downstream, respectively, by using the START/STOP option that takes the region strand into account.



Example 4:
CTCF_NORM_SCORE = PROJECT(ALLBUT score; region_update: new_score AS (score / 1000.0) + 100; metadata_update: normalized AS 1) CTCF_RAW;

This GMQL statement creates a new dataset called CTCF_NORM_SCORE by preserving all region attributes apart from *score*, and creating a new region attribute called *new_score* by dividing the existing score value of each region by 1000.0 and incrementing it by 100.
It also generates, for each sample of the new dataset, a new metadata attribute called *normalized* with value 1, which can be used in future selections.

Example 5:
DS_out = PROJECT(variant_classification, variant_type;
metadata: manually_curated__tissue_status, manually_curated__tumor_tag) DS_in;

This statement produces an output dataset that contains the same samples as the input dataset. Each output sample only contains, as region attributes, the four basic coordinates (chr, left, right, strand) and the specified region attributes *variant_classification* and *variant_type*, and as metadata attributes only the specified ones, i.e. *manually_curated__tissue_status* and *manually_curated__tumor_tag*.

Example 6:
D2 = PROJECT(metadata_update: age AS age + 10) D1;
D3 = PROJECT(metadata_update: age_plus AS age + 100) D1;

The first statement produces an output dataset that contains the same samples as the input dataset. Each output sample contains the same region attributes as the samples in D1. As metadata attributes, each sample contains the same ad D1 samples with the exception of the attribute *age*, which is incremented by 10.
The second statement has the same effect but instead of substituting the *age* attribute, it adds the *age_plus* attribute, which corresponds to the former age incremented by 100.

Example 7:
RES = PROJECT(ALLBUT score, pValue) INPUT;

This example shows how to use the ALLBUT option to exclude multiple attributes. This statement returns as output the same samples of the INPUT dataset, which have the same region attributes as the INPUT samples with the exception of *score* and *pValue*.

3) EXTEND

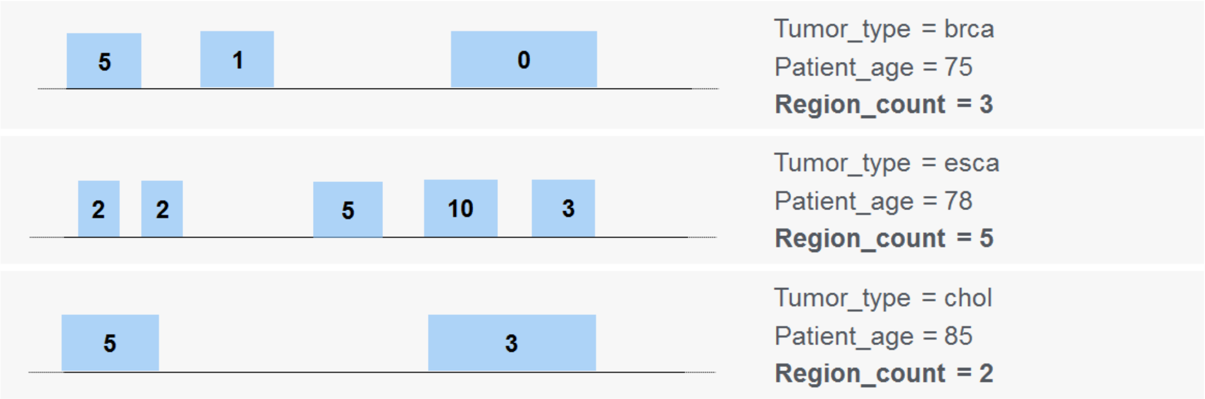
For each sample in an input dataset, the EXTEND operator builds new metadata attributes, assigns their values as the result of arithmetic and/or aggregate functions calculated on sample region attributes, and adds them to the existing metadata attribute-value pairs of the sample. Sample number and their genomic regions, with their attributes and values, remain unchanged in the output dataset.
The general syntax for EXTEND is:
DS_{out} = EXTEND(NM_1 AS g_1, ..., NM_k AS g_k) DS_{in};
where:

- DS_{in} is the input dataset whose sample region attribute values are used to compute the new sample metadata;
- DS_{out} is the output dataset, a copy of the input dataset with additional metadata calculated by EXTEND;
- NM_1, ..., $NM_{k;}$ are new metadata attributes generated using arithmetic and/or aggregate functions $g_1, ..., g_k$ on the sample region attributes in DS_{in}. In addition to the usual aggregate functions, additional ones are available: q1(*region_attribute*), q2(*region_attribute*), q3(*region_attribute*), which are respectively the first, second, and third quartile of the values of the specified *region_attribute*.

Example 1:
RES = EXTEND(RegionCount AS COUNT()) EXP;

This GMQL statement counts the regions in each sample and stores their number as value of the new metadata *RegionCount* attribute of the sample.

RES:



<u>Example 2</u>:
RES = EXTEND(RegionCount AS COUNT(), MinP AS MIN(Pvalue)) EXP;

This GMQL statement copies all samples of EXP into RES dataset, and then calculates two new metadata attributes for each of them:

1. *RegionCount* is the number of sample regions;
2. *MinP* is the minimum *Pvalue* of the sample regions.

RES sample regions are the same as the ones in EXP.

<u>Example 3</u>:
OUT = EXTEND(allScores AS BAG(score)) DATA;

This GMQL statement copies all samples of DATA into OUT dataset, and then for each of them adds another metadata attribute, *allScores*, which is the aggregation comma-separated list of all the distinct values that the attribute *score* takes in the sample.

<u>Example 4:</u>
RES = EXTEND(quar1 AS q1(score)) my_ds;

This statement copies all the samples of my_ds into the RES dataset and, for each of them, it adds an additional metadata attribute *quar1*, calculate as the first quartile value of the sample's scores distribution.

4) ORDER

The ORDER operator is used to order either samples, sample regions, or both, in a dataset according to a set of metadata and/or region attributes, and/or region coordinates. The number of samples and their regions in the output dataset is as in the input dataset, as well as their metadata and region attributes and values, but a new ordering metadata and/or region attribute is added with the sample or region ordering value, respectively.
The general syntax of ORDER is the following:
DS_{out} = ORDER(MA_1 DESC, ..., MA_n DESC;
meta_top: *k* OR meta_topg: *k*;

region_order: RA_1 DESC,...,RA_m DESC;
region_top: *k* OR region_topg: *k*) DS_{in};

where:

- DS_{in} is the input dataset;
- DS_{out} is the output sorted dataset;
- MA_1, ..., $MA_{n;}$ are the ordering metadata attributes;
- $RA_1, \ldots, RA_m$ are the ordering genomic region attributes;
- DESC is an optional parameter to be put after each ordering attribute that reverses the ordering with respect to that attribute (default is ascending, becomes descending);
- *k*, where specified, is the number of samples (or regions) to be extracted from the ordered dataset (or from each sample), starting from the top (with respect to the final ascending/descending ordering).

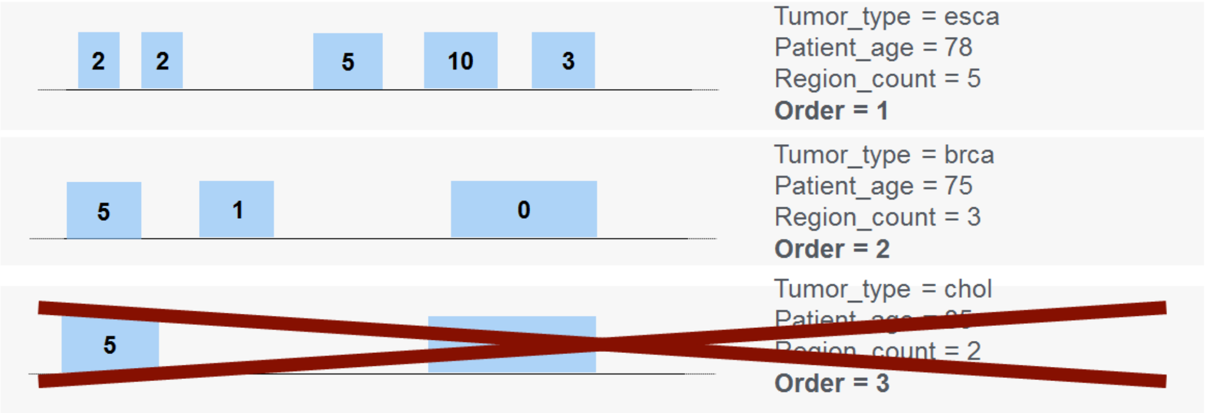
Sorted samples, or sample regions, have a new attribute *order* added to either metadata, regions, or both of them; the value of *order* reflects the result of the sorting. The clauses meta_top: *k* and region_top: *k* extract the first k samples and regions, respectively, according to the final ordering. The clauses meta_topg: *k* and region_topg: *k* implicitly consider the ordering defined by first grouping identical values of the first n − 1 ordering attributes, and then sorted by the remaining attributes; they then select the first k samples, or regions, of each group.

Note: ORDER does not have a default form (the statement ORDER() DS_{in} does not compile). At least one parameter is required.

Example 1:
OUTPUT_DS = ORDER(Region_count DESC; meta_top: 2;) INPUT_DS;

This GMQL statement orders the samples according to the *Region_count* metadata attribute and takes the two samples that have the highest count. As shown in the following figure, the sample with attribute *Order* = *3* is excluded from the output.

OUTPUT_DS:



Example 2:
OUTPUT_DS = ORDER(RegionCount; meta_top: 5;

region_order: MutationCount DESC; region_top: 7) INPUT_DS;

This GMQL statement extracts the first 5 samples on the basis of their region counter (those with the smaller *RegionCount*) and then, for each of them, 7 regions on the basis of their mutation counter (those with the higher *MutationCount*).

Example 3:
OUTPUT_DS = ORDER(treatment_type, ID DESC; meta_top: 2) INPUT_DS;

This GMQL statement first sorts the samples in INPUT_DS dataset by ascending order with respect to their metadata *treatment_type*, then it sorts them by descending order based on the values of their metadata *ID* attribute (the new metadata attribute *_order* is added to all samples). Finally, only the samples with *_order* = 1 or *_order* = 2 are extracted in the output dataset OUTPUT_DS.

Example 4:
OUTPUT_DS = ORDER(region_order: pvalue, length, name; region_topg: 1) INPUT_DS;

This GMQL statement groups the regions in each sample of INPUT_DS dataset according to their ascending *pvalue* and *length* order and then, for each group, only outputs the first region based on alphabetical order of the attribute *name*.

5) GROUP

The GROUP operator is used for grouping both regions and/or metadata of input dataset samples according to distinct values of certain attributes (known as *grouping attributes*); new grouping attributes are added to samples in the output dataset, storing the results of aggregate function evaluations over metadata and/or regions in each group of samples.
Samples having missing values for any of the grouping attributes are discarded.
The general syntax for GROUP is:
DS_{out} = GROUP($\boldsymbol{MA_1}$**, ...,** $\boldsymbol{MA_n}$;
meta_aggregate: GM_1 AS f_1, ..., GM_k AS f_k;
region_group: RA_1, ..., RA_m;
region_aggregate: GR_1 AS g_1, ..., GR_h AS g_h) DS_{in};
where:

- DS_{in} is the input dataset;
- DS_{out} is the output dataset;
- MA_1, ..., $MA_{n;}$ are the grouping metadata attributes;
- GM_1, ..., $GM_{k;}$ are new grouping metadata attributes generated using arithmetic and/or aggregate functions $f_1, ..., f_k$ on the metadata attributes in DS_{in};
- $RA_1, …, RA_m$ are the conserved genomic region attributes;
- GR_1, ..., $GR_{h;}$ are new grouping region attributes generated using arithmetic and/or aggregate functions $g_1, ..., g_h$ on the attributes of regions in DS_{in}.

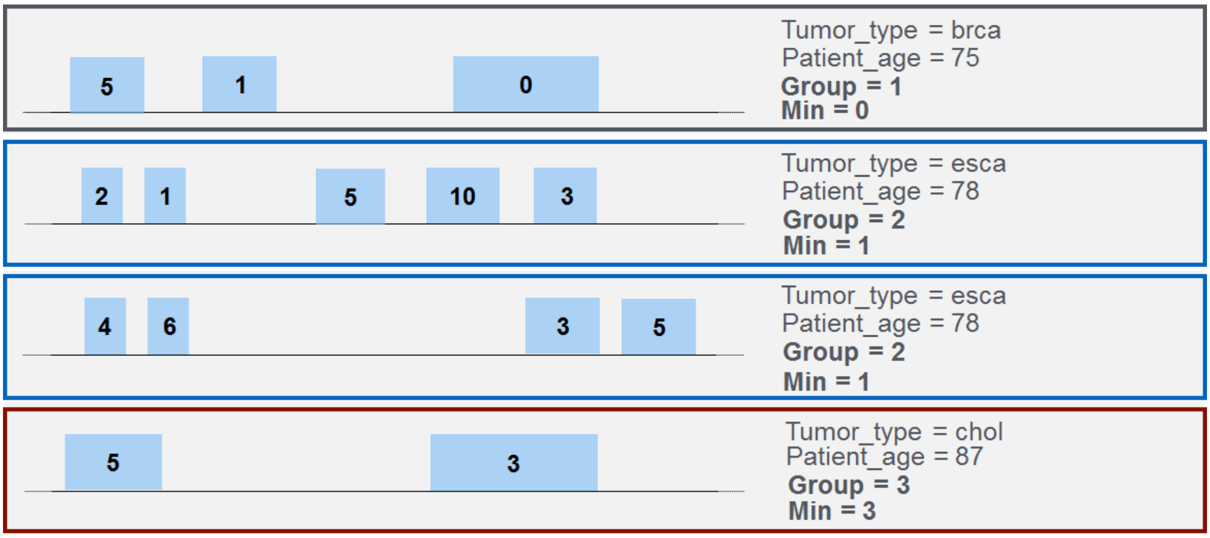
Several observations can be made on the effect of GROUP:

- The metadata of output samples, each corresponding to a given group, are constructed as the union of metadata of all the samples contributing to that group; consequently, metadata include the attributes storing the grouping values, that are common to all samples in the group.
- Should a grouping attribute be multi-valued, samples are partitioned by each subset of their distinct values (e.g., samples with a Disease attribute set both to ‘Cancer’ and ‘Diabetes’ are within a group which is distinct from the groups of the samples with only one value, either ‘Cancer’ or ‘Diabetes’).
- When grouping applies to regions, by default it includes as grouping attributes the region coordinates *chr*, *left*, *right*, *strand*. This choice corresponds to the biological procedure of removing duplicate regions, i.e. regions with the same coordinates, ensuring that the result is a legal GMQL sample.
- Other attributes may be added to grouping attributes (e.g., RegionType); aggregate functions can then be applied to each group. The resulting schema includes the attributes used for grouping and possibly new attributes used for the aggregate functions.

Example 1:
GROUPS = GROUP(Tumor_type; region_aggregate: Min AS MIN(score)) EXP;

This GMQL statement groups samples according to the value of *Tumor_type* and computes the minimum *score* of each group.

GROUPS:



Example 2:
GROUPS = GROUP(cell; region_aggregate: Pvalue AS MIN(Pvalue)) EXP;

This GMQL statement first groups the samples of EXP by *cell* values, then calculates the minimum *Pvalue* over (non-duplicated) regions in each group and uses this value to further group the results.

Example 3:

GROUPS = GROUP(metadata_aggregate: cells AS BAG(cell); region_group: Pvalue) EXP;

This statement first groups by the region coordinates *chr*, *left*, *right*, *strand* and then considers, as additional region grouping attribute, the *Pvalue*. In addition, in the output samples, the new metadata attribute *cells* is added, computed as the concatenation of all the values that the attribute *cell* takes in the samples of the grouped regions.

6) MERGE

The MERGE operator builds a new dataset consisting of a single sample having

- as regions all the regions of all the input samples, with the same attributes and values
- as metadata the union of all the metadata attribute-values of the input samples.

A *groupby* clause can be specified on metadata: the samples are then partitioned in groups, each with a distinct value of the grouping metadata attributes, and the MERGE operation is applied to each group separately, yielding to one sample in the result dataset for each group. Samples without the grouping metadata attributes are disregarded.

The general syntax for MERGE is:

DS_{out} = MERGE(groupby: M_1, ..., M_n) DS_{in};

where:

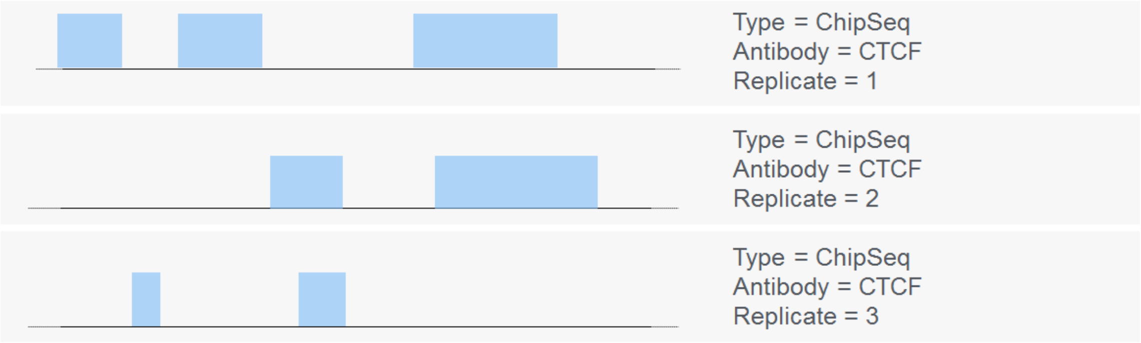
- DS_{in} is the input dataset to be merged;
- DS_{out} is the output dataset;
- M_1, ..., M_n are the (optional) metadata attributes used in the *groupby* clause (see below).

<u>Example 1</u>:

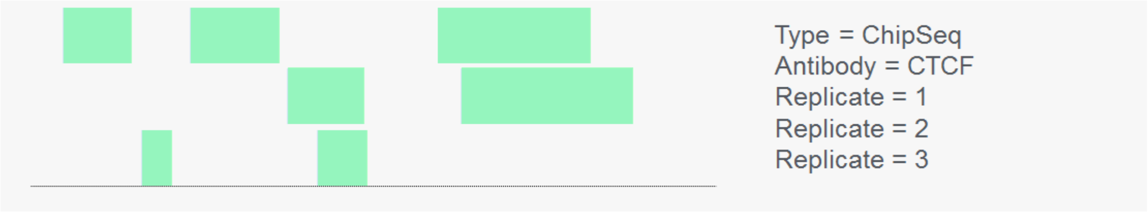
MERGED_ALL = MERGE() INIT_DATA;

This GMQL statement collapses a bunch of samples (both regions and metadata) into a single one. More in detail, it creates a new dataset MERGED_ALL consisting of a single sample having as regions all the regions in the INIT_DATA dataset, with the same attributes and values, and as metadata the union of all the metadata attributes values of the samples of INIT_DATA.

For instance, we may have this INIT_DATA input dataset:



And would get this MERGED_ALL result:



Example 2:
MERGED = MERGE(groupby: antibody_target) EXPERIMENT;

This GMQL statement creates a dataset called MERGED which contains one sample for each antibody_target value found within the metadata of the EXPERIMENT dataset sample; each created sample contains all regions from all EXPERIMENT samples with a specific value for their *antibody_target* metadata attribute.

7) UNION

The UNION operation is used to integrate homogeneous or heterogeneous samples of two datasets within a single dataset; for each sample of either one of the input datasets, a sample is created in the result as follows:

- its metadata are the same as in the original sample;
- its schema is the schema of the first (left) input dataset; new identifiers are assigned to each output sample;
- Its regions are the same (in coordinates and attribute values) as in the original sample. Region attributes which are missing in an input dataset sample (w.r.t. the merged schema) are set to null.

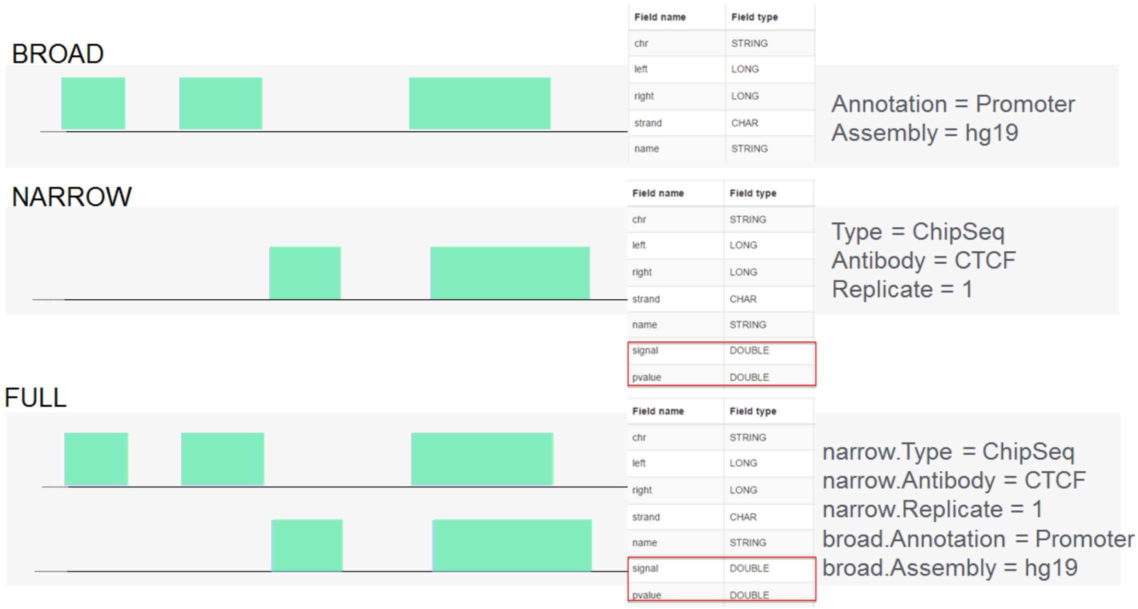
The general syntax for UNION is:
DS_{out} = UNION() DS_1 DS_2;
where:

- DS_1 and DS_2 are the input datasets to be unified;
- DS_{out} is the unified output dataset;

The merging of two schemas is performed by adding to the schema of the first dataset the region attributes of the second dataset which are not identical to those of the first dataset; two region attributes are considered identical if they have the same name and type. For what concerns metadata, attributes of samples from the first (second) input dataset are enriched with an additional attribute _provenance so as to trace the dataset to which they originally belonged.

Example:
FULL = UNION() BROAD NARROW;

This statement creates a dataset called FULL which contains all samples from the datasets BROAD and NARROW (that could include broadPeak and narrowPeaks data samples from ENCODE experiments), whose schema is defined by merging BROAD and NARROW dataset schemas (union of all the attributes present in the two input datasets).



Field name	Field type
chr	STRING
left	LONG
right	LONG
strand	CHAR
name	STRING

Annotation = Promoter
Assembly = hg19

Field name	Field type
chr	STRING
left	LONG
right	LONG
strand	CHAR
name	STRING
signal	DOUBLE
pvalue	DOUBLE

Type = ChipSeq
Antibody = CTCF
Replicate = 1

Field name	Field type
chr	STRING
left	LONG
right	LONG
strand	CHAR
name	STRING
signal	DOUBLE
pvalue	DOUBLE

narrow.Type = ChipSeq
narrow.Antibody = CTCF
narrow.Replicate = 1
broad.Annotation = Promoter
broad.Assembly = hg19

8) DIFFERENCE

DIFFERENCE is a binary, non-symmetric operator that produces one sample in the result for each sample of the first operand, by keeping the same metadata of the first operand sample and only those regions (with their schema and values) of the first operand sample which do not intersect with any region in the second operand sample (also known as *negative regions*).
The general syntax for DIFFERENCE is:
DS_{out} = DIFFERENCE(joinby: M_1,..., M_n) DS_{ref} DS_{neg};
where:

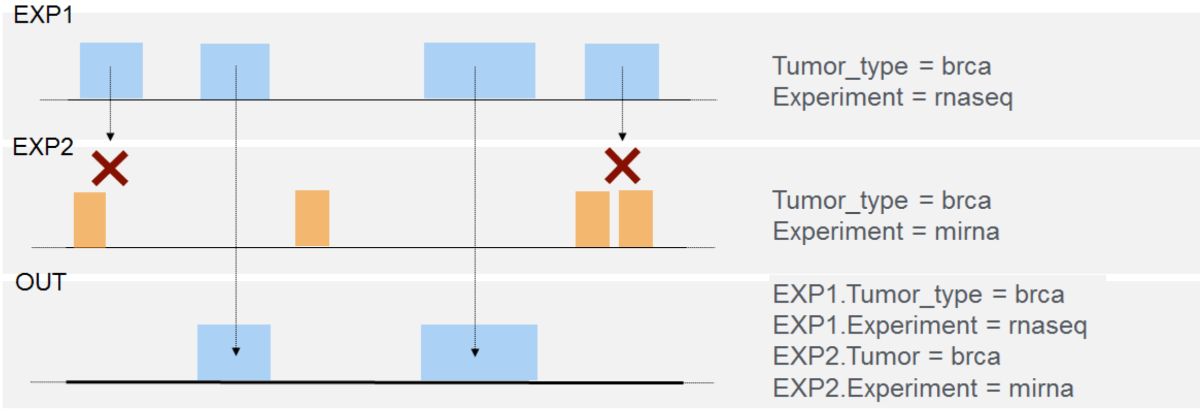
- DS_{ref} is the *reference* dataset, i.e. the dataset which is copied in the output and from which regions of DS_{neg} are "subtracted";
- DS_{neg} is the *negative* dataset, i.e. the dataset whose regions are checked for intersection against the reference regions. If any reference region is found to have intersection with a region in DS_{neg}, it is removed from the output dataset;
- DS_{out} is the output dataset;
- M_1, ..., $M_{n;}$ are the (optional) metadata attributes used in the joinby clause.

The optional **joinby** clause is used to extract a subset of couples from the cartesian product DS_{ref} x DS_{neg} on which to apply the DIFFERENCE operator: only those samples $s_1 \in DS_{ref}$ and $s_2 \in DS_{neg}$ that have the same value for each attribute M_1 through M_n are considered when performing the DIFFERENCE.

Note: DIFFERENCE operates in two different modes based on region intersection: the default behavior (i.e., DIFFERENCE() DS_{ref} DS_{neg}), and the exact matching (i.e., DIFFERENCE(exact:true) DS_{ref} DS_{neg}).

Example 1:
OUT = DIFFERENCE() EXP1 EXP2;

This GMQL statement returns all the regions in the first dataset that do not overlap any region in the second dataset.



Example 2:
RES = DIFFERENCE(exact:true) mydsaJ1 mydsaJ2;

This statement extracts all regions in the first dataset that do not coincide (exactly from the start to the end coordinate) with at least a region in the second dataset.

Example 3:
OUT = DIFFERENCE(joinby: antibody_target) EXP1 EXP2;

This GMQL statement extracts for every pair of samples $s_1 \in$ EXP1 and $s_2 \in$ EXP2 having the same value of the metadata attribute **antibody_target** the regions that appear in s_1 but do not overlap any region in s_2; metadata of the result are the same as the metadata of s_1.

9) MAP

MAP is a non-symmetric operator over two datasets, respectively called **reference** and **experiment**. The operation computes, for each sample in the experiment dataset, aggregates over the values of the experiment regions that intersect with a region in a reference sample, for each region of each sample in the reference dataset; we say that experiment regions are *mapped* to the reference regions. The number of generated output samples is the Cartesian product of the samples in the two input datasets; each output sample has the same regions as the related input reference sample, with their attributes and values, plus the attributes computed as aggregates over experiment region values. Output sample metadata are the union of the related input sample metadata, whose attribute names are prefixed with their input dataset name.
For each reference sample, the MAP operation produces a matrix like structure, called **genomic space**, where each experiment sample is associated with a row, each reference region with a column, and each matrix row is a vector of numbers - the aggregates computed during MAP execution. When the features of the reference regions are unknown, the MAP helps in extracting the most interesting regions out of many candidates.

The general syntax for MAP is:
DS_{out} = MAP(NR_1 AS $g_1, …, NR_h$ AS g_h;
joinby: MA_1,..., MA_n) DS_{ref} DS_{exp};
where:

- DS_{ref} is the *reference* dataset;
- DS_{exp} is the *experiment* dataset;
- DS_{out} is the output dataset;
- NR_1, ..., $NR_{h;}$ are new genomic region attributes (optionally) generated using functions $g_1, ..., g_h$ on existing experiment region attributes;
- MA_1, ..., $MA_{n;}$ are the (optional) metadata attributes used in the *joinby* clause (see below).

Note: the COUNT() aggregate (counting the number of experiment regions intersecting a certain reference region) is always computed; results are stored in an attribute named *count_[DSrefName]_[DSexpName]*, where *DSrefName* and *DSexpName* are the names of DS_{ref} and DS_{exp}, respectively.
To rename the default name to an example custom name *myCountName*, use the following syntax: Z = MAP(count_name: *myCountName*) X Y;

We first describe the effect of the basic MAP operation (without **joinby** clause). Let:

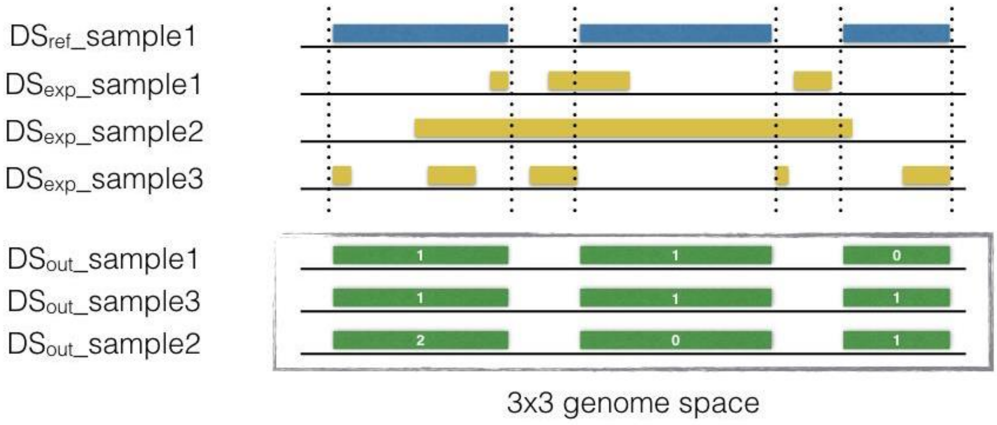
- $s_1 \in DS_{ref}$ be a given reference sample with R_1 the set of its regions and M_1 its metadata;
- s_2 be a generic sample of the experiment dataset DS_{exp} with R_2 the set of its regions and M_2 its metadata.

A new sample s_3 is constructed as follows:

- the metadata M_3 are obtained by merging metadata M_1 and M_2, taking track of their provenance by prefixing their attribute names with the name of their original dataset;
- the regions R_3 are created such that, for each region $r_1 \in R_1$, there is exactly an equal region $r_3 \in R_3$, with the same coordinates and having as attributes the attributes of r_1 and, in addition, the new attributes computed by the aggregate functions g_i specified in the operation; such aggregate functions are applied to the attributes of all the regions $r_2 \in R_2$ having a non-empty intersection with r_1.

The operation is iterated for each experiment samples and each reference sample, and it generates a reference sample-specific genomic space at each iteration.

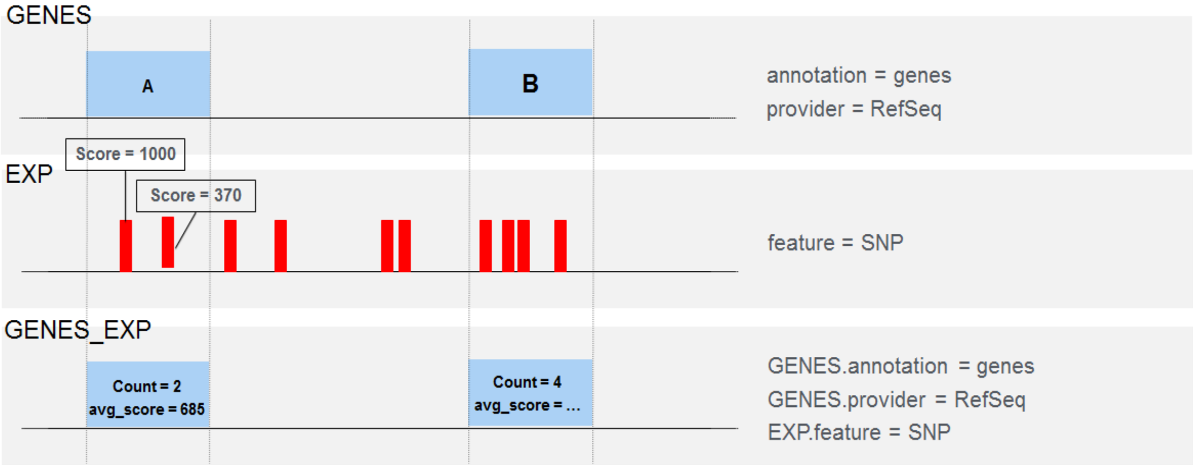
When the **joinby** clause is present, only pairs of samples s_1 of DS_{ref} and s_2 of DS_{exp} with metadata M_1 and M_2 that satisfy the *joinby* condition are considered. Syntactically, the clause consists of a list of metadata attribute names (or their suffixes) that must be present with equal values in both M_1 and M_2 (attribute names specified in the *joinby* clause can also refer to only the last suffix of actual attribute names in M_1 and M_2 for the s_1 - s_2 matches to be detected and considered).



The above figure shows the result of the MAP operation (with no *joinby* clause) on a small portion of the genome. The input consists of one reference sample with 3 regions, in the DS_{ref} dataset, and three experiment samples in the DS_{exp} dataset; the output consists of the DS_{out} dataset with three samples, each with the same regions as in the reference sample, which contain a feature called **count_DS_{ref}_DS_{exp}** (where DS_{ref} and DS_{exp} are the input dataset names) counting the number of experiment regions which intersect with the specific reference region. The result can be interpreted as a (3 × 3) genome space.

Example 1:
GENES_EXP = MAP(avg_score AS AVG(score)) GENES EXP;

Given a dataset GENES, containing a single sample with a known set of genes, and another dataset EXP containing results from a genomic experiment on the same species, this GMQL statement counts the number of regions (e.g., peaks) in each sample from the experiment which overlap with a known gene, and computes the average (AVG) *score* value across such regions, saving results in the output as a region attribute (feature) called *avg_score*.

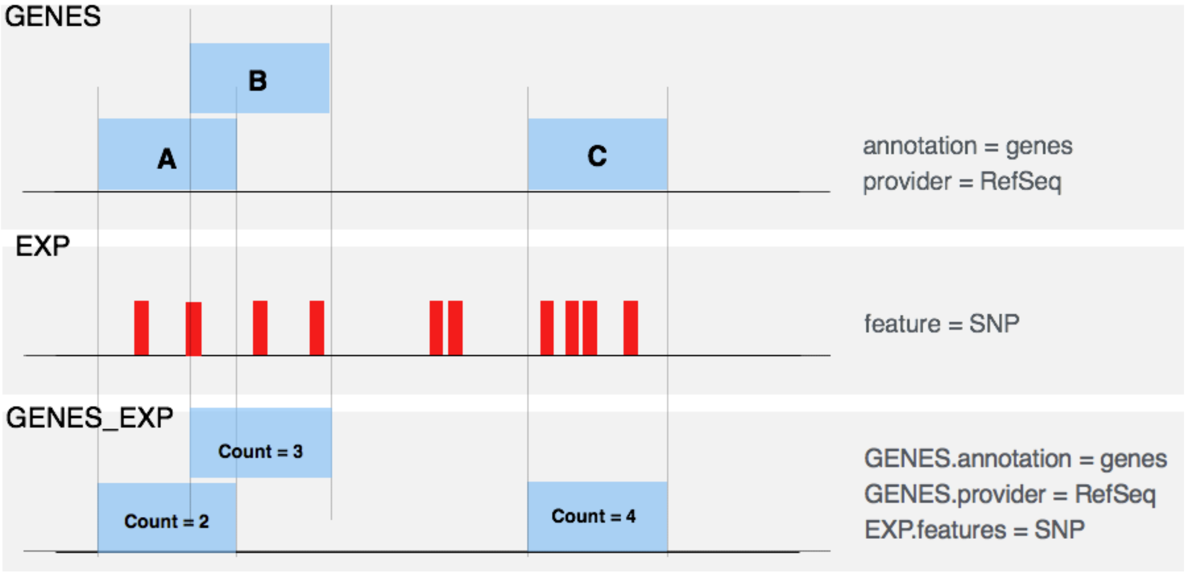


Example 2:
OUT = MAP (minScore AS MIN(score); joinby: cell_tissue) REF EXP;

This GMQL statement counts the number of regions in each sample from EXP that overlap with a REF region, and for each REF region it computes the minimum *score* of all the regions in each EXP sample that overlap with it. The MAP *joinby* option ensures that only the EXP samples referring to the same cell_tissue of a REF sample are mapped on such REF sample; EXP samples with no *cell_tissue* metadata attribute, or with such metadata but with a different value from the one(s) of REF sample(s), are disregarded.

Observation:

Notice that when a reference sample include (partially) overlapping regions, all these regions are included in the result regions, as it can be seen in the following figure.



10) JOIN

The JOIN operator takes in input two datasets, respectively known as *anchor* (the first/left one) and *experiment* (the second/right one) and returns a dataset of samples consisting of regions extracted from the operands according to the specified condition (known as *genometric predicate*). The number of generated output samples is the Cartesian product of the number of samples in the anchor and in the experiment dataset (if no joinby close if specified). The attributes (and their values) of the regions in the output dataset are the union of the region attributes (with their values) in the input datasets; homonymous attributes are disambiguated by prefixing their name with their dataset name. The output metadata are the union of the input metadata, with their attribute names prefixed with their input dataset name.

The general syntax for JOIN is the following:

DS_{out} = JOIN(pred: equipredicate;
genpred: genometric_predicate;
output: coord-param;
joinby: MA_1, ..., MA_n) DS_{anc} DS_{exp};

where:

- DS_{anc} and DS_{exp} are respectively the *anchor* and *experiment* datasets;
- DS_{out} is the output dataset;
- *equipredicate* is an optional list of pairs of attributes used for equi-joins of regions, where the first attribute is from the left operand and the second attribute is from the right operand;

- *genometric_predicate* is an optional concatenation of *distal* conditions by means of logical ANDs (see later for details);
- *coord-param* is one of four different values that declare which region is given in output for each input pair of anchor and experiment regions satisfying the genometric predicate:
    - LEFT outputs the anchor regions from DS_{anc} that satisfy the genometric predicate; LEFT can be prefixed by the keyword DISTINCT which calls from the duplicate elimination of regions of DS_{anc} with the same value, corresponding to distinct values of DS_{exp} *;*
    - RIGHT outputs the experiment regions from DS_{exp} that satisfy the genometric predicate; RIGHT can be prefixed by the keyword DISTINCT which calls from the duplicate elimination of regions of DS_{exp} with the same value, corresponding to distinct values of DS_{anc} *;*
    - INT outputs the overlapping part (intersection) of the anchor and experiment regions that satisfy the genometric predicate; if the intersection is empty, no output is produced;
    - CAT (or CONTIG) outputs the concatenation between the anchor and experiment regions that satisfy the genometric predicate, i.e. the output region is defined as having *left* (*right*) coordinates equal to the minimum (maximum) of the corresponding coordinate values in the anchor and experiment regions satisfying the genometric predicate;
- MA_1, ..., MA_n are the (optional) metadata attributes used in the *joinby* clause (see below).

The *joinby* condition (also called *meta-join* predicate) is used to select sample pairs satisfying certain conditions on their metadata (e.g., regarding the same cell line or antibody target); syntactically, it is expressed as a list of metadata attribute whose names and values must match between samples in DS_{anc} and DS_{exp} in order for such samples to very the condition and be considered for the join.

Genometric predicates are discussed in detail at the end of the section.

The join result is constructed as follows:

- The meta-join predicate initially selects all pairs s_i of DS_{anc} and s_j of DS_{exp} that satisfy the *joinby* condition. If the clause is omitted, then the complete Cartesian product between samples of DS_{anc} and of DS_{exp} is selected;
- For each such pair, a new sample s_{ij} is generated in the result, having metadata given by the union of metadata of s_i and s_j;
- The genometric predicate is tested for all the pairs (r_i,r_j) of regions, for each $r_i \in s_i$ and $r_j \in s_j$. This is done by giving (in turn) to each $r_i \in s_i$ the role of anchor region and then evaluating the genometric predicate condition with all the regions r_j of s_j.
- The equijoin predicate is tested for all the pairs (r_i,r_j) of regions, for each $r_i \in s_i$ and $r_j \in s_j$.
- For every pair (r_i,r_j) that satisfies the conjunction of the genometric and equijoin predicates, a new region is generated in s_{ij}, according to the *coord-param* value; the attributes and their values of the new region are all those of the r_i*, and* r_j regions, and

homonymous attributes are disambiguated by prefixing their input dataset name to their name.

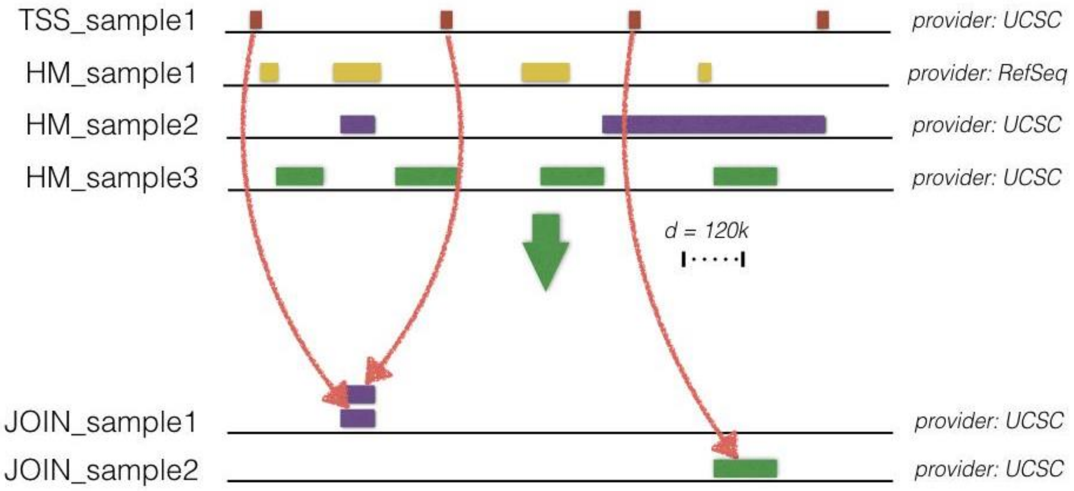
Note 1: by construction, the JOIN operation yields results whose number can grow quadratically both in the number of samples and of regions; hence, it is the most computationally intensive of all GMQL operations.

Note 2: The default behavior with syntax JOIN() *DS*$_{anc}$ *DS*$_{exp}$ is equivalent to specify the genometric predicate option DISTANCE<0 (i.e., A = JOIN(distance<0) *DS*$_{anc}$ *DS*$_{exp}$).

Example 1:
HM_TSS = JOIN(MD(1), DGE(120000); output: RIGHT; joinby: provider) TSS HM;

Given a dataset HM of ChIP-seq experiment samples regarding Histone Modifications and one called TSS with a sample including Transcription Start Site annotations, this GMQL statement searches for those regions of HM that are at a minimal distance from a transcription start site (TSS) and takes the first/closest one for each TSS, provided that such distance is greater than 120K bases and joined TSS and HM samples are obtained from the same *provider* (*joinby* clause).

Assume that sample metadata are as shown in the following picture.

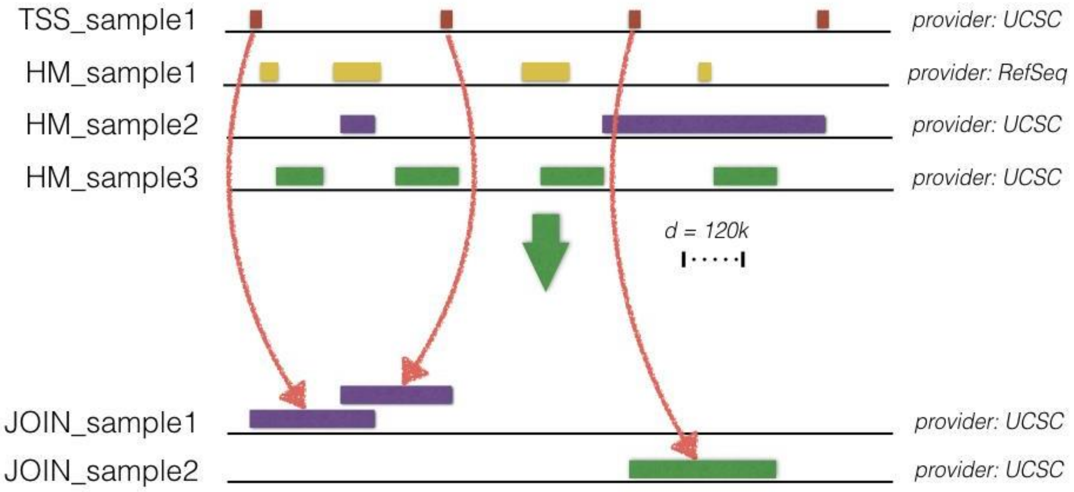


The first sample (HM_sample1) will be excluded from JOIN genometric predicate scan because of mismatching metadata: the cardinality of the result dataset will be 2x1 = 2 samples. The result includes only the selected regions of the right input dataset (in this case HM), with their attributes and values together with the attributes and values of the joined region in the other input dataset (in this case the left one, HM).

Regions from HM_sample2 (purple) are overlapping in JOJN_sample1 because both of them are valid JOIN results (for the first two red regions, respectively); from HM_sample3 only one green region is selected since MD(1) condition is evaluated first (see 12.1 for details.)

Example 2:
HM_TSS = JOIN(MD(1), DGE(120000); output: CAT; joinby: provider) TSS HM;

This example includes the same input datasets, genometric predicate and joinby attribute as Example 1, but the output is produced as the concatenation of regions selected by the genometric predicate (see CAT description above). In the following picture we report the output JOIN samples in this scenario:

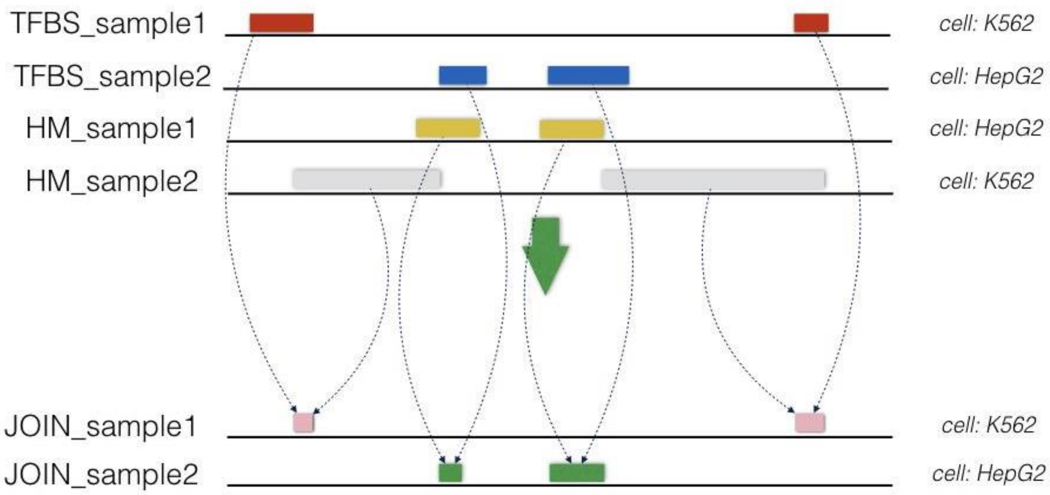


Example 3:
TFBS_TSS = JOIN(DGE(5000), DLE(100000); output: LEFT) TFBS TSS;

Given a dataset TFBS that contains peak regions of transcription factor binding sites (TFBSs), and another dataset named TSS that contains 1 bp-long transcription start sites (TSSs), this GMQL statement returns as output all those TFBSs that are far no more than 100k bp, but no less than 5000 bp from a TSS (i.e. in possible enhancer regions).
If one would instead be interested in the TSSs that have at least one TFBS in such regions, this statement could be changed by using *output: right* instead of *output: left* as parameter (see example 1).

Example 4:
TF_HM_OVERLAP = JOIN(DLE(-1); output: INT; joinby: cell) TFBS HM;

Given a dataset TFBS that contains peak regions of transcription factor binding sites (TFBSs) for a certain TF, and another dataset named HM that contains regions resulting from experiments targeting specific histone modifications (for instance methylations), this JOIN statement returns as output the intersection of histone modification regions with the transcription factor binding sites that overlap in the same cell line (indicated by the joinby parameter).
For suitable histone modifications, this can be a potential indicator of chromatin openness used as evidence of actual TF bindings to the DNA.



Genometric predicates are fundamental for JOIN commands: they allow the expression of a variety of distal conditions all based on the concept of **genomic distance**. The genomic distance is defined as the number of base pairs (i.e., nucleotides) between the closest opposite ends of two regions belonging to the same chromosome, measured from the right-end of the region with left-end lower coordinate.

Note: In the GMQL framework, overlapping regions have negative distance while adjacent regions have distance equal to 0.

A genometric predicate is a sequence of distal conditions (i.e., evaluated using genomic distance) defined as follows:

- MD(K) (or MINDIST(K), MINDISTANCE(K)) denotes the *minimum distance clause*, which selects the first K regions of an experiment sample at minimal distance from an anchor region of an anchor dataset sample. In case of ties (i.e., regions at the same distance from the anchor region), all tied experiment regions are kept in the result, even if they would exceed the limit of K;
- DLE(N) (also DIST <= N, DISTANCE <= N) denotes the *less-equal distance clause*, which selects all the regions of the experiment such that their distance from the anchor region is less than, or equal to, N bases. There are two special less-equal distances clauses: DLE(-1) searches for regions of the experiment which overlap with the anchor region (regardless the extent of the overlap), while DLE(0) searched for experiment regions adjacent to, or overlapping, the anchor region;
- DGE(N) (also DIST >= N, DISTANCE >= N) denotes the *greater-equal distance clause*, which selects all the regions of the experiment such that their distance from the anchor region is greater than, or equal to, N bases.

An additional clause that can be specified in a genometric predicate is UP/DOWN (or UPSTREAM/DOWNSTREAM), called the *upstream/downstream clause*, which refers to the upstream and downstream directions of the genome. This clause requires that the rest of the predicate hold only on the upstream (downstream) genome with respect to the anchor region. More specifically:

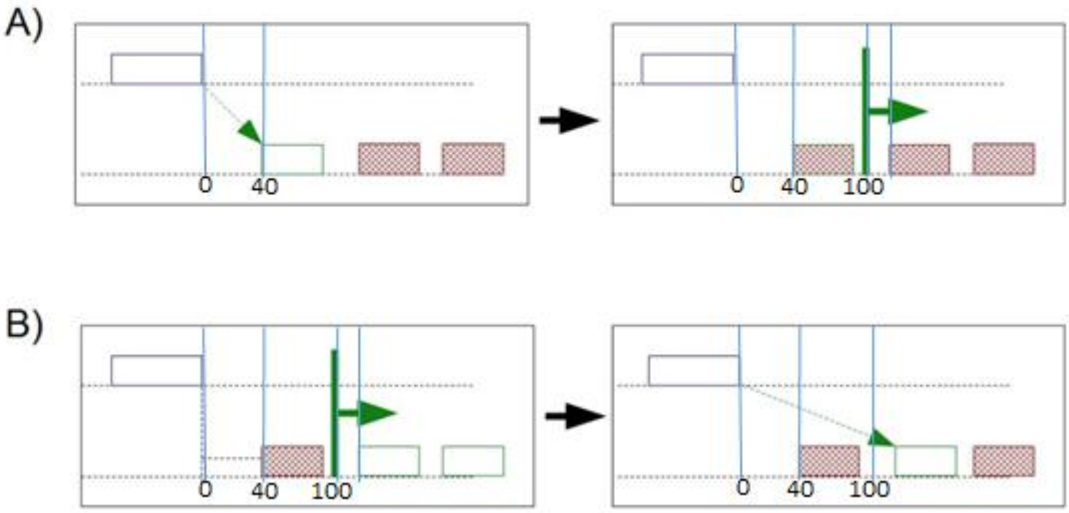
- in the positive strand (or when the strand is unknown), UP is true for those regions of the experiment whose right-end is lower than, or equal to, the left-end of the anchor, and DOWN is true for those regions of the experiment whose left-end is higher than, or equal to, the right-end of the anchor;
- in the negative strand inequalities are exchanged;
- remaining regions of the experiment must be overlapping with the anchor region.

When this clause is not present, distal conditions apply to both directions of the genome indifferently.

Genometric clauses are strings composed of concatenations of distal conditions; we say that a genometric clause is *well-formed* if and only if it includes at least one less-equal distance, or a minimum distance clause. Genometric predicates (clauses) used in JOIN statements must be well-formed.

Examples:
The following strings are legal, well-formed, genometric predicates:

- DGE(500), UP, DLE(1000), MD(1);
- DLE(2000), MD(1), DOWN;
- MD(100), DLE(3000)



Note: different orderings of the same distal clauses may produce different results. In the above figure, we show an evaluation of the following two clauses relative to an anchor region:

A. MD(1), DGE(100);
B. DGE(100), MD(1).

In case A, the MD(1) clause is computed first, producing one region which is next excluded by computing the DGE(100) clause; therefore, no region is produced as result. In case B, the DGE(100) clause is computed first, producing two regions, and then the MD(1) clause is computed, producing one region as result.

Similarly, the clauses:

A. MD(1), UP

B. UP, MD(1)

may produce different results: in case A, the minimum distance region is selected regardless of its up/down stream position to the anchor, and then it is retained if and only if it belongs to the upstream of the anchor, while in case B only upstream regions are considered, and the one at minimum distance is selected.

11) COVER

COVER is a GMQL operator that takes as input a dataset (of usually, but not necessarily, multiple samples) and returns another dataset (with a single sample, if no *groupby* option is specified) by "collapsing" the input samples and their regions according to certain rules specified by the COVER parameters. The attributes of the output regions are only the region coordinates, plus in case, when aggregate functions are specified, new attributes with aggregate values over attribute values of the contributing input regions; output metadata are the union of the input ones.

The COVER operation is used to:

- reduce the regions of multiple samples in a single sample (particularly when the samples are replicas of the same experiment);
- deal with overlapping regions;
- compute aggregate on the overlapping regions.

The general syntax of the COVER operator is:

DS_{out} = COVER(**minAcc, maxAcc**; groupby: $M_1, \dots, M_n$;
aggregate: NR_1 AS $g_1, \dots, NR_h$ AS g_h) DS_{in};

where:

- DS_{in} is the input dataset;
- minAcc (maxAcc) is the minimum (maximum) accumulation value, i.e. the minimum (maximum) number of overlapping regions to be considered during COVER execution;
- DS_{out} is the output dataset;
- M_1, ..., $M_{n;}$ are the (optional) metadata attributes used in the *groupby* clause (see below);
- NR_1, ..., $NR_{h;}$ are new genomic region attributes (optionally) generated using aggregate functions $g_1, \dots, g_h$ on existing region attributes.

The special keywords *ANY* and *ALL* can be used instead of numbers for *minAcc* and *maxAcc*. In particular:

- ALL sets the minimum (and/or maximum) to the number of samples in the input dataset;
- ANY acts as a wildcard and can be used only as *maxAcc* value; in this case, the COVER extracts all regions with any maximum accumulation value. For instance, COVER(2, ANY) will consider all areas defined by a minimum of two overlapping regions in the input, up to any amount of overlapping (3, 4, 5, and so on).

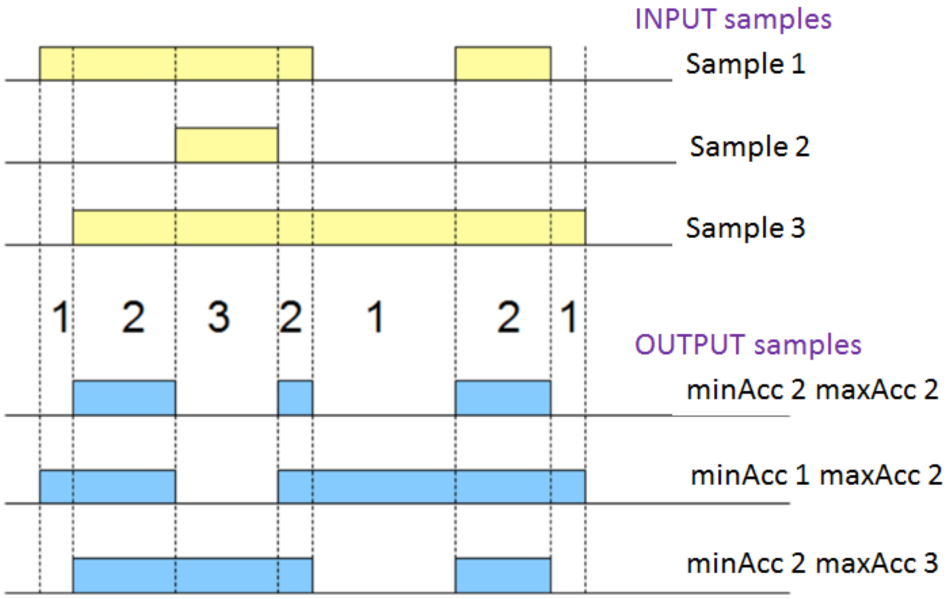
Note 1: default COVER and its three variants (FLAT, SUMMIT, and HISTOGRAM), which are described in the following, do not have default arguments (i.e., COVER() DS_{in} does not compile). *minAcc* and *maxAcc* must be always specified.

Note 2: *minAcc* and *maxAcc* can be optionally expressed as functions of ALL, with the following possible structures:

- ALL / n;
- (ALL + k) / n.

We first describe the COVER operation with no grouping. In such case, the operation produces a single output sample, and all the metadata attribute-values of the contributing input samples in DS_{in} are assigned to the resulting single sample in DS_{out}. Regions of the resulting sample are built from DS_{in} in the following way:

- Each resulting region *r* in DS_{out} is the contiguous intersection of at least *minAcc* and at most *maxAcc* contributing regions r_i in the samples of DS_{in};
- When regions are stranded, COVER is separately applied to positive and negative strands. In this case unstranded regions are accounted both as positive and negative;
- Resulting regions may have new attributes NR_i, calculated by means of aggregate expressions over the attributes of the contributing regions: for instance, *Jaccard Indexes* are standard measures of similarity of the contributing regions r_i, added as default region attributes. The *JaccardIntersect* index is calculated as the ratio between the lengths of the intersection and of the union of the contributing regions; the *JaccardResult* index is calculated as the ratio between the lengths of the result and of the union of the contributing regions.

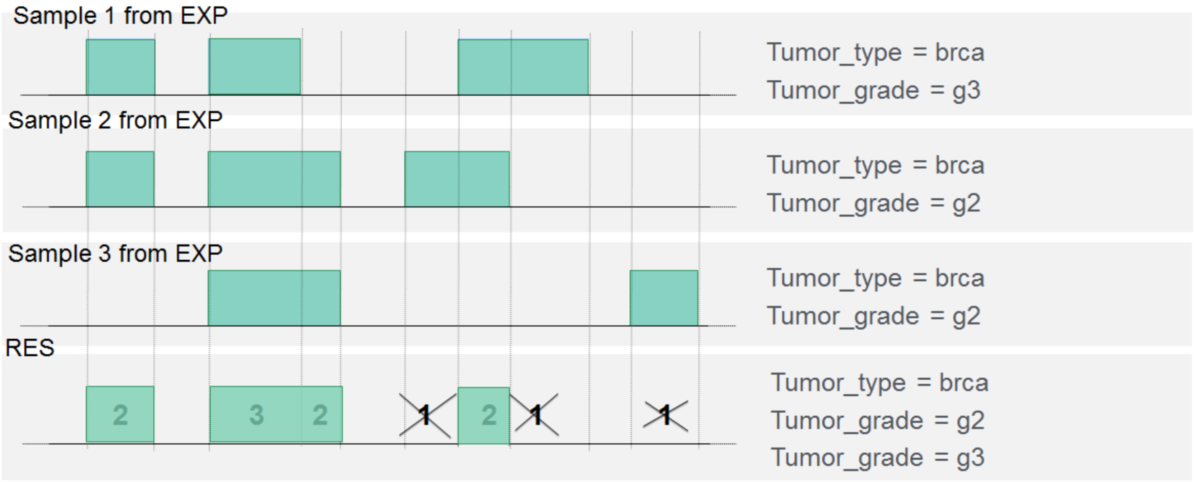


In the above figure we show the results of COVER with *minAcc* and *maxAcc* parameter values set respectively to (2, 2), (1, 2), and (2, 3). Note that in the figure case ALL = 3, so for instance COVER(2, 3) provides the same result as COVER(2, ALL).

When a *groupby* clause is specified, the input samples are partitioned in groups, each with distinct values of the grouping metadata attributes, and the COVER operation is separately applied (as described above) to each group, yielding to one sample in the result for each group (input samples that do not satisfy the *groupby* condition are disregarded).

Example 1:
RES = COVER(2, ANY) EXP;

This GMQL statement produces an output dataset with a single output sample. The COVER operation considers all areas defined by a minimum of two overlapping regions in the input samples, up to any amount of overlapping regions. In the figure below we show how no regions are created in the output where only one or no region in the input samples is present. Output region attributes include only region coordinates and Jaccard indexes (*JaccardIntersect* and *JaccardResult*). Metadata are the union of the input metadata, as shown in figure.



Example 2:
RES = COVER(2, 3; groupby: cell; aggregate: min_pValue AS MIN(pValue)) EXP;

This GMQL statement computes the result grouping the input EXP samples by the values of their *cell* metadata attribute, thus one output RES sample is generated for each cell type; output regions are produced where at least 2 and at most 3 regions of grouped EXP samples overlap, setting as attributes of the resulting regions the minimum pValue of the overlapping regions (*min_pValue*) and their Jaccard indexes (*JaccardIntersect* and *JaccardResult*).

Example 3:
CELL_TF = COVER(1, ANY; groupby: cell, antibody_target) NARROW_PEAK;

Given a dataset NARROW_PEAK, containing transcription factor binding site (TFBS) regions from a repository (e.g. ENCODE), for each antibody target of each cell line, this GMQL statement produces output regions where at least a binding site of the given transcription factor for the given cell exists, grouping cells (first) and antibody targets (then); output regions have only their Jaccard indexes as their attributes. This statement is typically used to extract any possible regions where a TFBS for a given cell line can exist; by rising the *minAcc* parameter (e.g. to 2, 3 or more), the same statement can be used to extract consensus regions (i.e. regions with higher probability of containing actual signal, in the example case a TFBS for a given cell line).

Three variants are available in GMQL for the COVER operation, which vary the coordinates of the returned regions as follow:

- FLAT returns the union of all the regions which contribute to the COVER. More precisely, it returns the contiguous regions that start from the first end and stop at the last end of the regions which would contribute to each region of a COVER;
- SUMMIT returns only those portions of the COVER result where the maximum number of regions overlap (this is done by returning only regions that start from a position after which the number of overlaps does not increase, and stop at a position where either the number of overlapping regions decreases or violates the maximum accumulation index);
- HISTOGRAM returns all regions contributing to the COVER divided in different (contiguous) parts according to their accumulation index value (one part for each different accumulation value), which is assigned to the *AccIndex* region attribute.

The syntax for all variants is the same as for the COVER statement, only replacing COVER with FLAT, HISTOGRAM, or SUMMIT, respectively, as required.

Example 1:
RES = FLAT(2, 4; groupby: cell) EXP;

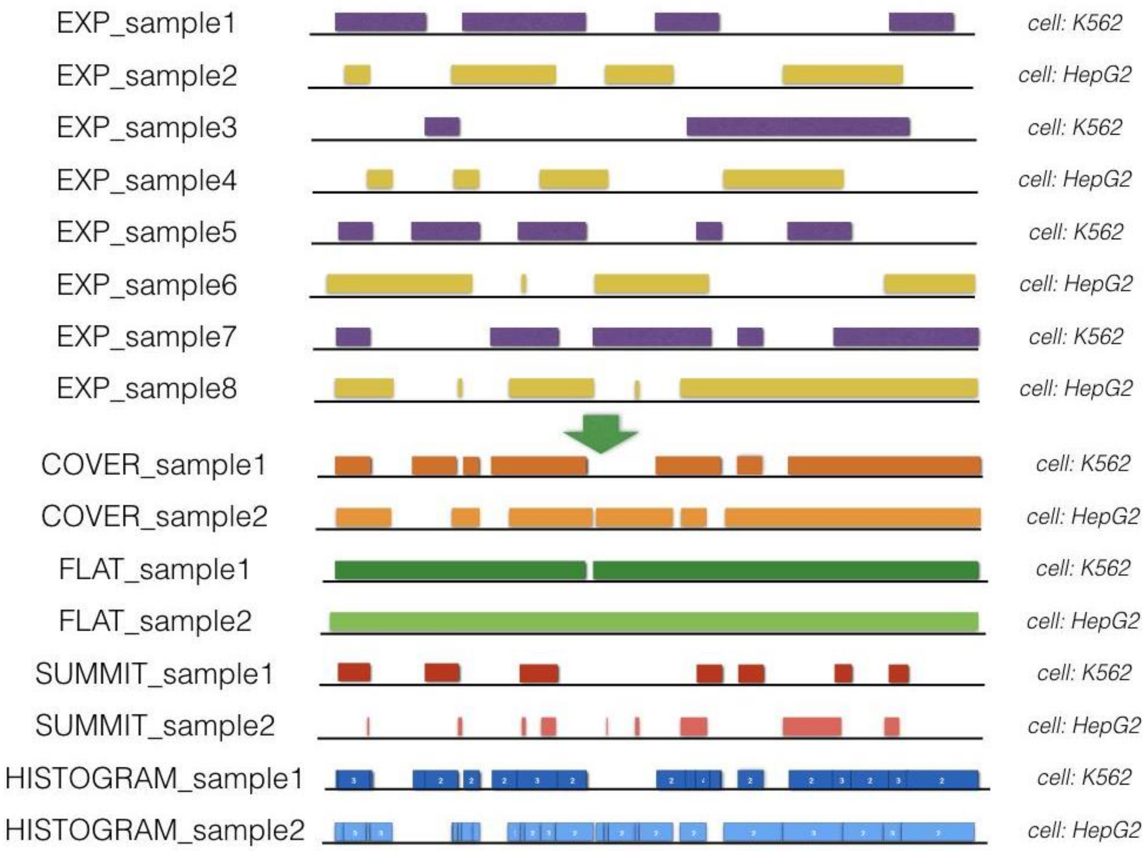
This GMQL statement computes the result grouping the input EXP samples by the values of their *cell* metadata attribute, thus one output RES sample is generated for each cell type.
Output regions are produced by concatenating all regions which would have been used to construct a COVER(2,4) statement on the same dataset; Jaccard indexes (*JaccardIntersect* and *JaccardResult*) are set as in the basic case.

Example 2:
RES = SUMMIT(2, 4; groupby: cell) EXP;

This GMQL statement computes the result grouping the input EXP samples by the values of their *cell* metadata attribute, thus one output RES sample is generated for each cell type.
Output regions are produced by extracting the highest accumulation overlapping (sub)regions according to the methodologies described above; Jaccard indexes (*JaccardIntersect* and *JaccardResult*) are set as in the basic case.

Example 3:
RES = HISTOGRAM(2, 4; groupby: cell) EXP;

This GMQL statement computes the result grouping the input EXP samples by the values of their *cell* metadata attribute, thus one output RES sample is generated for each cell type.
Output regions are produced by dividing results from COVER in contiguous subregions according to the varying accumulation values (from 2 to 4 in this case): one region for each accumulation value (see figure for a visual explanation); Jaccard indexes (*JaccardIntersect* and *JaccardResult*) are set as in the basic case.



Example 4:
RES = HISTOGRAM(ALL/2,(ALL+1)/2; groupby: antibody_target) EXP;

This statement computes the result grouping the input EXP samples by the calues of their *antibody_target* metadata attribute, thus one output sample is generated for each type of antybody target.
Assuming that the cardinality of the EXP dataset is of 8 samples, then ALL = 8. By computing the simple arithmetic operations, we retrieve that *minAcc* = 4 and *maxAcc* = floor(4.5).
Therefore, the output regions are produced exactly as if the user had performed an HISTOGRAM(4,4), after applying the *groupby* option.

12) MATERIALIZE

The MATERIALIZE operation saves the content of a dataset in a file, whose name can be specified, and registers the saved dataset in the system to make it usable in other GMQL queries.
The general syntax for MATERIALIZE is the following:
MATERIALIZE ***DS*** **INTO** ***file-name***;
where:

- *DS* is the dataset (temporary and local to the query) to be saved on the file system;
- *file-name* is the required name of the file into which the dataset *DS* must be saved.

Note: the actual GMQL implementation materializes *DS* into a file with a name in the form job_[queryname]_[username]_[timestamp]_file-name.
All datasets defined in a GMQL query are, by default, temporary; to store and access the content of any dataset generated during a GMQL query such dataset must be materialized. Any dataset can be materialized; however the operation is time expensive, so for better performance it is suggested to materialize only relevant datasets, such as the final output.

Example:
MATERIALIZE HM_TFS INTO res;

This GMQL statement saves the content of the temporary HM_TFS dataset into a file named *job_[queryname]_[username]_[timestamp]_res*.