Subject Store

Guía de usuario

Marzo 2025

1. Introducción

`SubjectStore` es una biblioteca para la gestión de datos en torno a un *Subject* (sujeto o asunto), permitiendo asociar atributos a marcas temporales. Su diseño se centra en *cómo* se alimentan los datos, y no en *cómo* se presentan, lo que posibilita fusionar distintas visiones parciales de un mismo *subject* sin imponer restricciones en su diseño sobre la forma de visualización final.

Para entender mejor el funcionamiento y utilidad de `SubjectStore`, consideremos un caso de uso típico: la gestión de información médica de un paciente, identificado como `12345`. Este paciente posee atributos que cambian cada vez que se realiza una prueba médica.

Por ejemplo, cuando al paciente se le realiza un hemograma, se tienen que registrar nuevos datos sobre diferentes parámetros sanguíneos (glóbulos rojos, glóbulos blancos, hemoglobina, plaquetas, etc.). El número de estas mediciones es variable y no siempre se registran todos los parámetros. Por otra parte, se podrían registrar diagnósticos o tratamientos específicos desde diversas especialidades médicas (como cardiología, neurología, etc.), cada una aportando información parcial sobre el estado del paciente.

Con `SubjectStore`, cada fuente aporta datos de forma independiente, registrando únicamente aquellos atributos de los que dispone información en cada instante. Esta flexibilidad permite almacenar conjuntamente atributos altamente dinámicos (como las mediciones del hemograma) y atributos menos variables (como diagnósticos) sin necesidad de definir previamente una estructura rígida de datos. La librería se encarga posteriormente de conciliar todas estas fuentes, ofreciendo una visión integral y fácilmente consultable del historial médico completo del paciente.

2. Prerrequisitos

Antes de comenzar a utilizar `SubjectStore` en tu proyecto, es necesario cumplir con los siguientes requisitos básicos:

- **Java 19 o superior**: Asegúrate de contar con una versión compatible del JDK instalada en tu entorno de desarrollo. Puedes verificar tu versión ejecutando:

  ```
  java -version
  ```

- **Apache Maven**: Es necesario disponer de Maven como gestor de dependencias y herramienta de compilación para integrar fácilmente la biblioteca en tu proyecto. Puedes comprobar si está instalado con el siguiente comando:

  ```
  mvn -version
  ```

- **Conocimientos básicos sobre series temporales y atributos categóricos**: Dado que `SubjectStore` maneja datos temporales, es recomendable entender conceptos elementales sobre series de tiempo y datos categorizados.

3. Instalación

Puedes instalar `SubjectStore` en tu proyecto Java utilizando un gestor de dependencias como Maven o Gradle.

Para incorporar `SubjectStore` mediante Maven, añade la siguiente dependencia al archivo `pom.xml` de tu proyecto:

```
<dependency>
    <groupId>systems.intino.datamarts</groupId>
    <artifactId>subject-store</artifactId>
    <version>1.0.0</version>
</dependency>
```

4. Carga de datos

Para iniciar la carga de datos en un `SubjectStore`, es necesario instanciar la clase proporcionando un nombre en el formato `id:tipo`, que identifica de forma única al sujeto y su tipo. Adicionalmente, se puede indicar un archivo o una conexión a base de datos donde almacenar la información. Si no se especifica ninguno, los datos se almacenan en memoria. El nombre del sujeto debe seguir el formato `id:tipo`, por ejemplo: `"12345:paciente"`.

- `new SubjectStore(name)` crea un almacén en memoria.
- `new SubjectStore(name, file)` almacena los datos en el archivo indicado.
- `new SubjectStore(name, connection)` utiliza una conexión SQL existente.

Los datos se introducen mediante el método `feed`, al que se le indica un instante temporal y una cadena que describe la fuente o el contexto del dato. Luego, se añaden atributos con el método `add`, especificando un par `tag-value`. Finalmente, se cierra la entrada con `terminate`. Los valores numéricos se introducen como `double` (coma flotante), y los atributos categóricos como cadenas de texto.

En el siguiente ejemplo se crea un `SubjectStore` asociado a un archivo. Si el archivo no existe, será creado automáticamente.

```
File file = new File("...");
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
    Instant instant = Instant.parse("2025-03-23T09:30:00Z");
    store.feed(instant, "Laboratorio.Hemograma.0512301")
         .add("hemoglobina", 14.2)
         .add("plaquetas", 250_000)
         .add("globulos-blancos", 5_600)
         .add("grupo-sanguineo", "A+")
         .terminate();
}
```

Si una segunda fuente quiere aportar atributos adicionales en otro instante, simplemente se repite el procedimiento:

```
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
    Instant instant = Instant.parse("2025-03-25T11:00:00Z");
    store.feed(instant, "Cardiologia.DrCabrera")
```

```
        .add("presion-sistolica", 125)
        .add("presion-diastolica", 80)
        .add("riesgo-cardiovascular", "Moderado")
        .terminate();
}
```

Este el siguiente ejemplo se muestra cómo crear un `SubjectStore` en memoria, sin necesidad de un archivo ni una conexión a base de datos. Es especialmente útil en contextos donde no se requiere persistencia, como pruebas unitarias, simulaciones, o procesos de análisis temporal intermedios. Al no depender del sistema de archivos, permite crear, modificar y descartar datos de forma rápida y eficiente.

```
try (SubjectStore store = new SubjectStore("12345:paciente")) {
    store.feed(Instant.now(), "Consulta.Urgencias")
        .add("temperatura", 37.8)
        .add("frecuencia-cardiaca", 88)
        .terminate();
}
```

Cuando se desea registrar múltiples aportaciones en una única operación, se puede utilizar el modo `batch`. Este permite agrupar varias llamadas a `feed`, `add` y `terminate` bajo una única sesión, optimizando la inserción de datos históricos o masivos.

```
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
    SubjectStore.Batch batch = store.batch();

    batch.feed(day1, "HMG-2").add("hemoglobina", 145).terminate();
    batch.feed(day2, "HMG-1").add("hemoglobina", 130).terminate();
    batch.feed(day3, "HMG-B").add("hemoglobina", 115).terminate();
    batch.feed(day4, "HMG-L").add("hemoglobina", 110).terminate();

    batch.terminate();
}
```

5. Referencias temporales

`SubjectStore` requiere que se indiquen instantes temporales para ubicar cronológicamente cada aportación de datos (`feed`). Para facilitar el manejo de

estas referencias temporales, la librería proporciona en la clase `TemporalReferences` algunas referencias especiales predefinidas:

- `thisYear()`: Inicio del año actual (1 de enero).
- `thisMonth()`: Inicio del mes actual (día 1).
- `thisWeek()`: Inicio de la semana actual (lunes).
- `today()`: Día actual, truncado a medianoche UTC.
- `thisHour()`: Hora actual, truncada a la hora exacta.
- `thisMinute()`: Minuto actual, truncado al minuto exacto.
- `thisSecond()`: Segundo actual, truncado al minuto (nota: parece un error, devuelve el minuto actual).

6. Consultas

Una vez que un `SubjectStore` ha sido alimentado con información, es posible consultar su contenido. Para abrir un archivo basta con instanciar la clase `SubjectStore` indicando el archivo (`.oss`) donde está almacenada la información.

```
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
  ...
}
```

La instancia de `SubjectStore` proporciona una serie de métodos que permiten acceder a información estructural y de estado sobre el sujeto y los datos almacenados. A continuación se detallan los principales métodos disponibles:

- `name()`: devuelve el nombre del `Subject`.
- `id()`: devuelve el id del `Subject`.
- `type()`: devuelve el tipo del `Subject`.
- `size()`: retorna el número total de (`feeds`) registrados.
- `ss(feed)`: devuelve la etiqueta asociada a un `feed` concreto, útil para identificar su origen o propósito.

- `from()`: devuelve el instante de tiempo correspondiente al primer `feed` cronológico real.
- `to()`: devuelve el instante de tiempo correspondiente al último `feed` cronológico real.
- `tags()`: devuelve todos los atributos registrados.
- `exists(tag)`: verifica si un atributo específico ha sido registrado.

Además, se pueden realizar consultas para un atributo dado. Hay dos tipos de consultas:

- `currentNumber(tag)`: devuelve directamente el valor numérico más reciente del atributo dado (o `null` si no existe).
- `currentText(tag)`: devuelve directamente el valor categórico más reciente del atributo dado (o `null` si no existe).
- `numericalQuery(tag)`: devuelve un objeto `NumericalQuery` si el atributo es numérico. Este objeto permite acceder a todos los valores históricos o aplicar operaciones estadísticas sobre los datos.
- `categoricalQuery(tag)`: devuelve un objeto `CategoricalQuery` si el atributo es de tipo texto. Permite acceder al conjunto de valores distintos observados.

6.1. Consulta de datos numéricos

Los atributos numéricos se consultan utilizando el método `numericalQuery`, que recibe por parámetros el nombre del atributo que se desea consultar. Para obtener el valor más reciente de un atributo, se utiliza el método `get()`, que devuelve un objeto `Point<Double>` con la información del valor, la fecha y el identificador del `feed` donde fue registrado. Si los datos fueron almacenados con escala (por ejemplo, multiplicando por 10 para representar decimales), será necesario dividir el valor al momento de interpretarlo. El valor devuelto por `get()` corresponde siempre al dato más reciente dentro del historial cronológico real.

```
Point<Double> hemoglobina = store
    .numericalQuery("hemoglobina")
    .get();
```

```
double value = hemoglobina.value() / 10.0;
Instant instant = hemoglobina.instant();
int feed = hemoglobina.feed();
```

Para consultar todos los valores registrados se usa el método `getAll()`. Para consultar los valores en un período determinado, se usa el método `get(from, to)`. Esto devuelve una serie temporal correspondiente al atributo dentro del intervalo indicado. La forma de operar con estas series se explica en la siguiente sección.

```
Signal signal = store
    .numericalQuery("hemoglobina")
    .getAll();
```

6.2. Consulta de datos categóricos

Los atributos categóricos se consultan utilizando el método `categoricalQuery`, que recibe por parámetro el nombre del atributo que se desea consultar. A diferencia de los datos numéricos, los valores categóricos son cadenas de texto y no requieren normalización ni escalado.

Para obtener el valor más reciente de un atributo categórico, se utiliza el método `get()`, que devuelve un objeto `Point<String>` con el valor, el instante y el identificador del `feed` correspondiente. El resultado de `get()` refleja el valor más reciente dentro del historial cronológico.

```
Point<String> riesgo = store
    .categoricalQuery("riesgo-cardiovascular")
    .get();
String value = riesgo.value();
Instant instant = riesgo.instant();
int feed = riesgo.feed();
```

Para consultar todos los valores registrados se utiliza el método `getAll()`. Para consultar los valores registrados en un rango de fechas, se utiliza el método `get(from, to)`. Este método devuelve una serie temporal que contiene los valores del atributo dentro del intervalo especificado. La forma de operar con estas series se explica en la siguiente sección.

```
Sequence sequence = store
    .categoricalQuery("riesgo-cardiovascular")
```

```
.get(thisMonth(-12), today());
```

7. Series temporales

7.1. Numéricas: `Signal`

Una serie temporal numérica se obtiene a partir de una consulta a un atributo mediante métodos de `get()`, que devuelven una instancia de `Signal`. Esta serie contiene todos los valores registrados en un intervalo de tiempo y permite operar directamente sobre ellos. Cada objeto `Signal` representa una serie de puntos de tipo `Point<Double>`, ordenados cronológicamente, e incluye los siguientes atributos y operaciones principales:

- `from()`: instante de tiempo del primer valor de la serie.
- `to()`: instante de tiempo del último valor de la serie.
- `duration()`: duración total de la serie.
- `points()`: lista completa de puntos de la serie, cada uno con su valor, instante y origen (`feed`).
- `values()`: array de `double` con los valores numéricos de todos los puntos, sin metadatos.

Además de acceder a los valores individuales de una serie temporal, también es posible realizar análisis estadístico mediante los métodos `summary()` y `distribution()`.

El método `summary()` calcula estadísticas básicas sobre la serie y devuelve un objeto `Summary` con los siguientes datos:

- `count`: número total de observaciones.
- `sum`: suma total de los valores.
- `mean`: media aritmética.
- `sd`: desviación estándar.
- `min`: punto con el valor mínimo registrado, incluyendo el instante y el feed.
- `max`: punto con el valor máximo registrado, también con metadatos temporales.

Por otro lado, el método `distribution()` construye una distribución probabilística, a través de la cual es posible obtener:

- `quantile(p)`: valor correspondiente al percentil `p`, con `p` entre 0 y 1.
- `q1()`, `q2()` (o `median()`), `q3()`: primer, segundo y tercer cuartil.
- `probabilityLeftTail(value)`: probabilidad acumulada de que un valor observado sea menor o igual al indicado.
- `probabilityRightTail(value)`: probabilidad de observar un valor superior al indicado.

7.2. Categóricas: `Sequence`

También es posible obtener un resumen estadístico sobre los valores categóricos registrados en un intervalo. Este resumen incluye métricas como:

- `count()`: número total de elementos registrados.
- `mode()`: valor más frecuente.
- `categories()`: conjunto de valores distintos observados.
- `frecuency(category)`: frecuencia absoluta de cada valor.

8. Segmentación de series temporales

Las series temporales, tanto numéricas (`Signal`) como categóricas (`Sequence`), pueden dividirse en segmentos más pequeños para facilitar análisis periódicos o localizados. Existen dos métodos principales para segmentar:

- `segments(Duration duration)`: divide la serie en tramos consecutivos de duración fija. Por ejemplo, segmentos diarios, semanales o por hora.
- `segments(int number)`: divide la serie en un número fijo de segmentos de igual longitud temporal.

En Java, la duración se representa mediante objetos que implementan la interfaz `TemporalAmount`. Si la duración es inferior a un día (por ejemplo, horas, minutos o segundos), se debe utilizar la clase `Duration`, con métodos como `Duration.ofHours(1)` o `Duration.ofMinutes(30)`. En cambio, para unidades de tiempo mayores o basadas en el calendario, como meses o

años, se utiliza la clase `Period`, con métodos como `Period.ofMonths(1)` o `Period.ofYears(1)`.

Cada segmento resultante es una serie temporal del mismo tipo que la original, lo que permite aplicar directamente sobre él los mismos métodos de consulta y análisis, así como volver a segmentarlo si es necesario. Por ejemplo, en el caso de una serie temporal numérica (`Signal`), cada segmento también es un `Signal`, por lo que admite operaciones como `summary()`, `distribution()`. Del mismo modo, una secuencia categórica (`Sequence`) segmentada produce otras instancias de `Sequence`, que pueden consultarse con `mode()`, `count()`, `categories()`.

```
Signal[] segments = hemoglobina.segments(Duration.ofDays(1));
for (Signal segment : segments) {
    var median = segment.distribution().median();
    ...
}
```

9. Volcado y restauración de datos

Una vez registrados los datos en un `SubjectStore`, es posible guardarlos en formato de texto plano utilizando el método `dump`. Este método permite volcar el contenido completo del `SubjectStore` a un fichero, generando una representación legible de los eventos registrados, que puede ser utilizado para inspección manual, depuración, trazabilidad o integración con otros sistemas.

```
File file = new File("...");
try (SubjectStore store = new SubjectStore("12345:paciente", file),
    OutputStream os = new FileOutputStream(os)) {
    store.dump(os);
}
```

Cada bloque de datos exportado mediante el método `dump` comienza con una línea de encabezado que indica el tipo del sujeto entre corchetes. El bloque incluye, además de los datos registrados en esa transacción, los metadatos básicos: el instante temporal (`ts`), la fuente que originó el dato (`ss`) y el identificador del sujeto (`id`).

```
[paciente]
ts=2025-03-25T00:00:00Z
ss=HMG-2
id=12345
hemoglobin=145.0
[paciente]
ts=2025-03-28T00:00:00Z
ss=HMG-2
id=12345
hemoglobin=130.0
...
```

El método `restore` permite reconstruir un `SubjectStore` a partir de un archivo previamente generado con `dump`. Interpreta cada bloque como un evento registrado en un instante concreto, restaurando así los eventos almacenados cuyo `id` y `type` coincidan con el del `SubjectStore`; los bloques con identificadores o tipos diferentes se ignoran.

```
File file = new File("backup.oss");
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
    store.restore(file);
}
```

Importante: durante el proceso de restauración, únicamente se importarán los bloques cuyo identificador coincida con el del `SubjectStore`. Cualquier entrada con un `id` distinto será ignorada automáticamente, lo cual garantiza que no se mezclen datos de sujetos diferentes en el mismo almacén.

10. Generación de vistas

La generación de vistas permite transformar los datos registrados en el store en tablas agregadas organizadas por intervalos temporales. Cada fila de la tabla representa un segmento de tiempo (por ejemplo, una semana), y cada columna muestra el resultado de una operación evaluada sobre los datos pertenecientes a ese intervalo.

A continuación, se muestra un ejemplo de creación de una vista a partir de un formato definido como cadena YAML:

```
String format = "...";
SubjectView view = new SubjectView(store, format);
view.exportTo(file);
```

La definición del formato de las vistas se realiza mediante un texto en formato YAML, que describe la estructura temporal y lógica de la transformación de los datos en una tabla agregada. Este texto puede cargarse desde un archivo o definirse directamente como una cadena en el código. Este formato tiene dos secciones principales:

- *Rows*: Permite definir el rango temporal y el periodo de agregación de la vista. Cada elemento de `rows` debe incluir las declaraciones `from`, `to` y `period`, que indican respectivamente el instante inicial, el instante final y la duración de cada intervalo. Las fechas utilizadas en `from` y `to` pueden tener distintos niveles de precisión: solo el año (`YYYY`), el mes (`YYYY-MM`), la fecha completa (`YYYY-MM-DD`) o la fecha con hora en formato ISO 8601. La duración indicada en `period` debe seguir el formato ISO 8601 de cantidades temporales, como `P1M` (un mes), `P7D` (siete días), o `PT1H` (una hora).
- *Columns*: Permite definir las columnas que se incluyen en la vista. Cada columna contiene un nombre (`name`) y una expresión (`expr`) que define cómo se obtiene su valor. Las expresiones pueden referirse a operaciones sobre datos, funciones temporales o estadísticas. De forma opcional, se pueden aplicar transformaciones mediante `filters`, que se indican como una lista de cadenas, con o sin parámetros.

10.1. Expresiones

Las expresiones que definen las columnas de una vista permiten realizar cálculos a partir de diferentes tipos de variables. Estas variables se pueden combinar usando operadores como `+`, `-`, `*` y `/`, así como funciones matemáticas como `sin()`, `cos()`, entre otras. Las variables pueden pertenecer a tres tipos principales:

- *Columnas previas*: Es posible referirse a otras columnas ya definidas dentro de la misma vista. Esto permite construir transformaciones encadenadas y reutilizar cálculos previos sin repetir expresiones. La referencia se realiza directamente mediante el nombre de la columna anterior.
- *Tags del store*: También se pueden utilizar datos almacenados en el *store*, en forma de *tags* que representan series temporales o eventos asociados a una propiedad. Para que un tag sea válido dentro de una

expresión, debe incluir al menos un **field** que especifique la propiedad concreta a evaluar (por ejemplo, `presion-sistolica`, `temperatura`, `frecuencia-cardiaca`, etc.). La sintaxis para acceder a un campo de un tag incluye un sufijo que indica el campo, como por ejemplo `temperatura.sum` o `presion-sistolica.average`.

- *Instantes temporales* `ts`: Las expresiones pueden acceder directamente a las propiedades del instante temporal representado por cada fila de la vista mediante el identificador `ts`. Este objeto contiene campios para extraer partes del tiempo, como por ejemplo `ts.month-of-year`, `ts.day-of-month`, `ts.hour-of-day`, entre otros. Estos campos permiten representar de forma explícita la dimensión temporal de los datos y generar columnas que describen la posición de cada intervalo en el calendario.

10.2. Campos de tags

Esta sección describe los campos de los tags almacenados en el store. Cada operador realiza una agregación o extracción específica sobre los valores registrados durante el intervalo temporal correspondiente. Los campos se ponen después del nombre del campo mediante una notación de punto (`campo.operador`). Por ejemplo, `temperatura.sum` calcula la suma de todos los valores de temperatura en el intervalo. Los campos disponibles cubren funciones estadísticas comunes tanto para datos numéricos como categóricos.

Operador	**Tipo**	**Descripción**
`count`		Número de observaciones
`sum`	Numérica	Suma total
`average`	Numérica	Media aritmética
`sd`, `standard-deviation`	Numérica	Desviación estándar
`first`	Numérica	Primer valor observado
`last`	Numérica	Último valor observado
`min`	Numérica	Valor mínimo
`max`	Numérica	Valor máximo
`mode`	Categórica	Valor más frecuente
`entropy`	Categórica	Dispersión o diversidad de valores

10.3. Campos temporales

Las marcas temporales accesibles mediante el identificador `ts` permiten obtener información sobre la posición temporal de cada intervalo dentro del calendario. Estos campos son especialmente útiles para construir variables temporales como el mes, el trimestre, o la fecha en diferentes formatos compactos, y pueden ser utilizados en expresiones o visualizaciones para identificar patrones estacionales o tendencias temporales. El acceso a estos campos se realiza mediante notación de punto, como `ts.year` o `ts.day-of-week`.

Marca Temporal	**Descripción**
`day-of-week`	Día de la semana (1–7)
`day-of-month`	Día del mes (1–31)
`month-of-year`	Mes del año (1–12)
`quarter-of-year`	Trimestre (1–4)
`year`	Año `YYYYMM`
`year-quarter`	Formato `YYYYQX`
`year-month`	Formato `YYYYMM`
`year-month-day`	Formato `YYYYMMDD`
`year-month-day-hour`	Formato `YYYYMMDDHH`
`year-month-day-hour-minute`	Formato `YYYYMMDDHHmm`
`year-month-day-hour-minute-second`	Formato `YYYYMMDDHHmmss`

10.4. Funciones

Las expresiones también pueden incluir funciones matemáticas aplicadas a los valores calculados, sean estos campos temporales, columnas intermedias o resultados de agregaciones. Estas funciones permiten realizar transformaciones no lineales, redondeos, conversiones trigonométricas y operaciones de análisis más avanzadas. Todas las funciones se aplican utilizando una sintaxis típica de llamadas funcionales, como `sin(x)`, `log10(x)` o `sqrt(x)`. Es posible encadenar múltiples funciones o combinarlas con operadores aritméticos para generar expresiones complejas de forma concisa y legible.

Función	Descripción	Función	Descripción
`abs`	Valor absoluto	`sin`	Seno
`negate`	Opuesto del valor (`-x`)	`cos`	Coseno
`round`	Redondeo	`tan`	Tangente
`floor`	Redondeo hacia abajo	`asin`	Arcoseno
`ceil`	Redondeo hacia arriba	`acos`	Arcocoseno
`signum`	Signo del número	`atan`	Arcotangente
`exp`	Exponencial e^x	`sinh`	Seno hiperbólico
`log`	Logaritmo natural ($\ln x$)	`cosh`	Coseno hiperbólico
`log10`	Logaritmo en base 10	`tanh`	Tangente hiperbólica
`sqr`	Cuadrado de un número		
`sqrt`	Raíz cuadrada	`rad`	Grados a radianes
`cbrt`	Raíz cúbica	`deg`	Radianes a grados

10.5. Filtros

Las columnas pueden incorporar **filtros**, que son transformaciones aplicadas sobre el resultado de una columna numérica. Los filtros permiten modificar, suavizar o normalizar los valores calculados antes de ser representados en la vista. Su uso es especialmente útil para destacar tendencias, eliminar ruido o preparar datos.

Filtro	Parámetros
`BinaryThreshold`	`threshold`
`CumulativeSum`	–
`Differencing`	–
`Lag`	`offset`
`MinMaxNormalization`	–
`ZScoreNormalization`	–
`RollingAverage`	`window`
`RollingSum`	`window`
`RollingStandardDeviation`	`window`
`RollingMax`	`window`
`RollingMin`	`window`

Cuadro 1: Filtros disponibles y sus parámetros

10.6. Ejemplo

A continuación, se muestra un ejemplo de formato para la generación de una vista semanal, combinando atributos clínicos numéricos y categóricos, junto con fórmulas derivadas y filtros.

```
rows:
   from: 2025-01-01
   to: 2025-01-31
   period: P7D

columns:
  - name: year
    expr: year

  - name: month
    expr: month-of-year

  - name: day
    expr: day-of-month

  - name: sistolica-suma
    expr: presion-sistolica.sum

  - name: sistolica-media
    expr: presion-sistolica.average

  - name: sistolica-tendencia
    expr: sistolica-media
    filters: [RollingAverage:3]

  - name: temperatura-media
    expr: temperatura.average

  - name: temperatura-normal
    expr: temperatura-media
    filters: [MinMaxNormalization]

  - name: temperatura-puntos
    expr: temperatura-normal * 100

  - name: temperatura-medidas
    expr: temperatura.count
```

```
- name: frecuencia-cardiaca-ultima
  expr: frecuencia-cardiaca.count
```

El resultado es una tabla semanal sin encabezados, donde cada fila representa una semana y cada columna corresponde a un valor clínico agregado, normalizado o derivado:

2025	1	1	0.0	0.0		0.0	0.0	0.0	0.0	0.0
2025	1	8	240.0	120.0	120.0	36.8	1.0	100.0	3.0	88.0
2025	1	15	0.0	0.0	60.0	0.0	0.0	0.0	0.0	0.0
2025	1	22	0.0	0.0	90.0	0.0	0.0	0.0	0.0	0.0
2025	1	29	120.0	120.0	110.0	36.5	0.75	75.0	2.0	70.0

11. Guía de estilo para nombres

Para mantener la coherencia, la legibilidad y facilitar la interoperabilidad entre sistemas, se recomienda seguir una convención clara y consistente para los nombres utilizados en `SubjectStore`. Esta guía establece el uso de **kebab-case** como convención principal.

Tags (atributos)

Los `tags` representan los nombres de los atributos registrados y deben escribirse usando *kebab-case*: todo en minúsculas, palabras separadas por guiones y sin espacios, acentos ni caracteres especiales.

Correcto: `presion-sistolica`, `frecuencia-cardiaca`.

Incorrecto: `presion_sistolica`, `presionSistolica` (camelCase), `Presión Sistolica` (acentos y espacios).

Identificadores (`id`)

El `id` identifica de forma única al sujeto. Puede ser numérico o alfanumérico.

Correcto: `12345`, `paciente-001`, `sensor-a12`.

Tipo (`type`)

El `type` describe la categoría del sujeto.

Correcto: `paciente`, `sensor`, `monitor-cardiaco`, `equipo-laboratorio`.