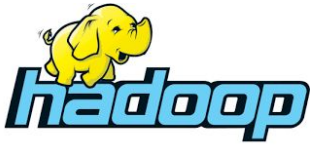
Stream processing et SQL

Bruno Bonnin

@_bruno_b_

#devoxxfr

About me



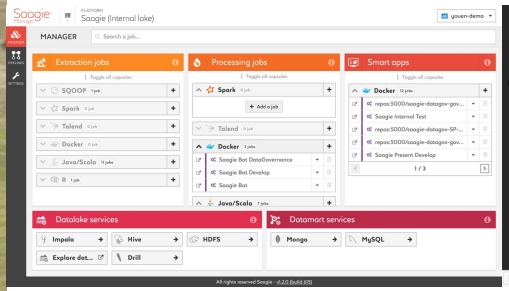
https://github.com/bbonnin

DO YOU REALLY WANT TO

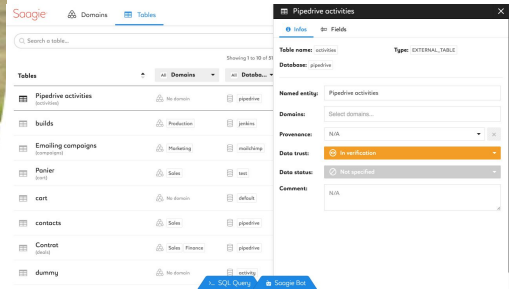
BUILD YOUR OWN BIG DATA PLATFORM?

Saagie

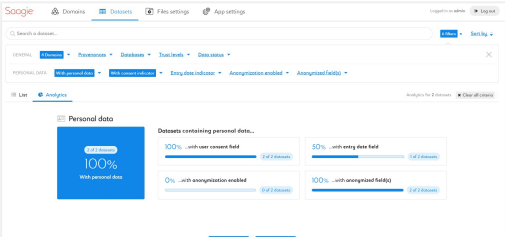
Data Fabric



Data Governance



GDPR



Flot continue et sans fin de données

Sources:

- IoT
- Monitoring
- Click streams
- Finance
- Jeux en ligne
- ...

Pourquoi SQL in 2017 ?

C'est un standard !

Largement adopté par tous (même si on peut/veut l'éviter…)

- Développeur
- Architecte data
- Data scientist
- …

Les streams sont des données comme les autres !

Alors pourquoi ne pas utiliser SQL pour les requêter ?

SQL vs. Stream processing

Relational Algebra / SQL	Stream Processing
Relations (or tables) are bounded (multi-)sets of tuples.	A stream is an infinite sequences of tuples.
A query that is executed on batch data (e.g., a table in a relational database) has access to the complete input data.	A streaming query cannot access all data when is started and has to "wait" for data to be streamed in.
A batch query terminates after it produced a fixed sized result.	A streaming query continuously updates its result based on the received records and never completes.

Source: https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/table/streaming.html

SQL et Stream processing

Apache Calcite

Apache Calcite

- Catalogue des metadatas
- Parsing SQL
- Validation des requêtes SQL
- Optimisation des requêtes SQL
  - Plan d'exécution
- Adaptateurs pour différentes sources de données (MongoDB, Elastic, …)

L'exécution des requêtes est à la charge du système utilisant Calcite

Pour les **streams**: définition d'un minimum de mots-clés et de fonctions pour les requêter

THE keyword !

```
SELECT STREAM * FROM weblogs;
```

weblogs est un stream

Requêtes ne se terminant pas

Sur quelles données ? de maintenant à ...

```
SELECT STREAM url, status_code, nb_bytes FROM weblogs;
SELECT STREAM url, nb_bytes FROM weblogs WHERE status = 500;
```

Jointure avec une table

```
SELECT STREAM c.id_pizza, p.prix
FROM commandes_pizza AS c
JOIN pizzas AS p
ON c.id_pizza = p.id_pizza;
```

Simple pour une table qui ne change pas

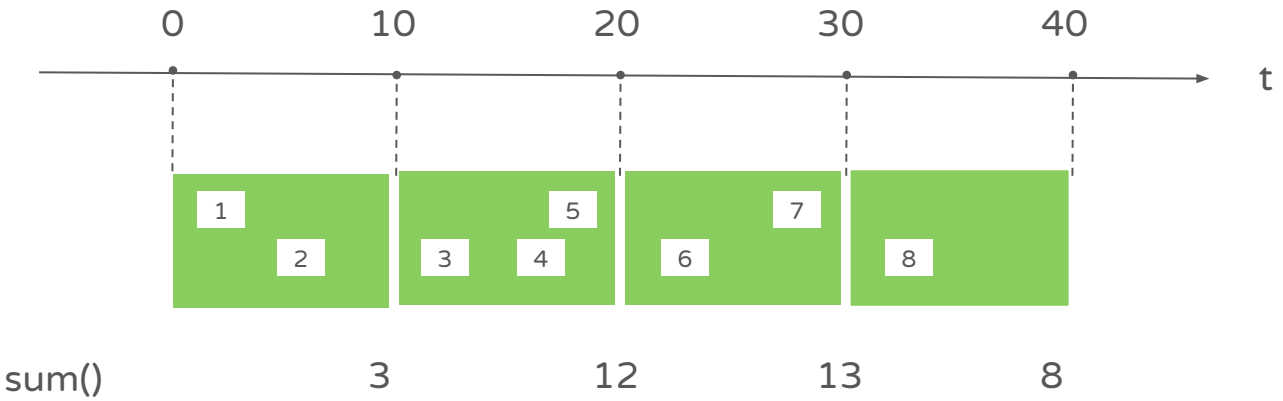
Et si la table bouge ?

Une solution: stocker l'historique dans la table pour l'utiliser dans la requête

```
SELECT STREAM c.id_pizza, p.prix
FROM commandes_pizza AS c
JOIN pizzas AS p
ON c.id_pizza = p.id_pizza
AND c.rowtime
    BETWEEN p.dateDebut AND p.dateFin;
```

Windowing

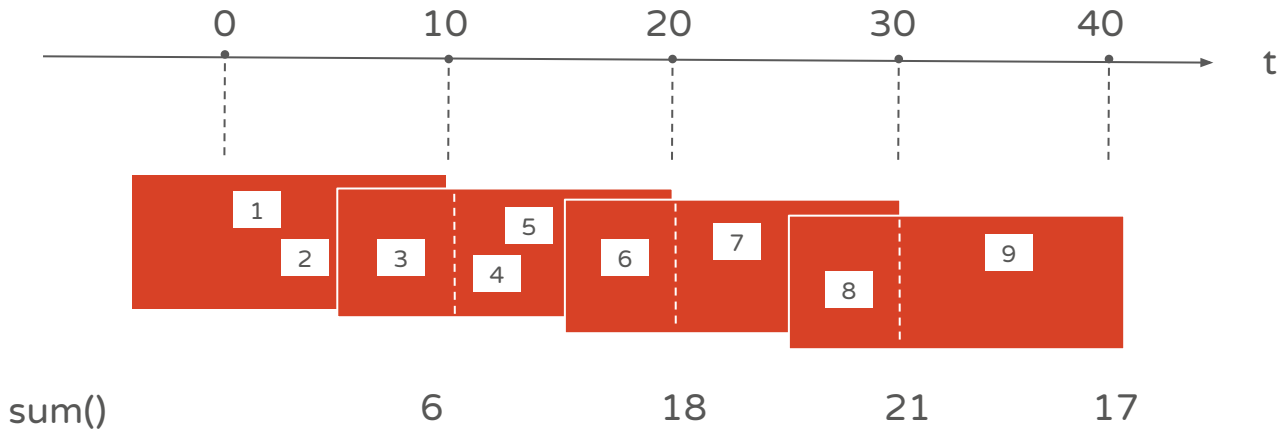
Tumbling window




```
SELECT STREAM
     TUMBLE_END(rowtime, INTERVAL '10' SECOND),
     url,
     SUM(nb_bytes) AS total_bytes
FROM weblogs
GROUP BY TUMBLE(rowtime, INTERVAL '10' SECOND), url;
```

Windowing

Hopping window




```
SELECT STREAM
    HOP_END(rowtime, INTERVAL '10' SECOND, INTERVAL '15' SECOND) AS rowtime,
    SUM(nb_bytes) AS total_bytes
FROM weblogs
GROUP BY HOP(rowtime, INTERVAL '5' SECOND, INTERVAL '10' SECOND);
```

Démo

https://github.com/bbonnin/talk-stream-processing-et-sql

Que préférez-vous ?

```
final Table table = tableEnv
  .fromDataStream(dataset,
    "ts, ip_address, url, status, nb_bytes, rowtime.rowtime" )
  .window(
    Tumble
      .over("10.second").on("rowtime").as("tenSecWindow"))
  .groupBy("tenSecWindow, url")
  .select(
     "url, tenSecWindow.end as time, url.count as nb_requests" );

tableEnv.toAppendStream(table, Row.class).print();

execEnv.execute();
```

Que préférez-vous ?

```
tableEnv.registerDataStream("weblogs", dataset,
       "ts, ip_address, url, status, nb_bytes, rowtime.rowtime" );

final String query =
       "SELECT url, " +
       "     TUMBLE_END(rowtime, INTERVAL '10' SECOND), "  +
       "     COUNT(*) AS nb_requests "  +
       "FROM weblogs " +
       "GROUP BY TUMBLE(rowtime, INTERVAL '10' SECOND), url" ;

final Table table = tableEnv.sql(query);
tableEnv.toAppendStream(table, Row.class).print();
execEnv.execute();
```

Conclusion

La partie stream de Calcite offre beaucoup d'autres possibilités comme les jointures entre streams, update/delete/insert, ...

- Détails: https://calcite.apache.org/docs/stream.html

Tout n'est pas implémenté !

Mais le but est de faire avancer le standard SQL.

Le SQL permet de réunir tous les acteurs autour d'un outil commun pour traiter les données, streams ou pas !

Merci !

@_bruno_b_

#devoxxfr

Group by, order by: quelques contraintes

```
SELECT STREAM status, count(*) FROM weblogs
        GROUP BY status;
```

Non autorisé par Calcite

Le GROUP BY doit inclure une valeur monotone (par ex, rowtime)

```
SELECT STREAM status, count(*) FROM weblogs
        GROUP BY rowtime, status;
```