

Foreign data wrapper study for schemaless databases

Hiroki Kumagai

February 6th, 2021

FOSDEM 2021

PostgreSQL devroom

Self-introduction

- Name: Hiroki Kumagai
- Live in Yokohama of Japan
- Engineer at Toshiba Corporation (Advanced Collaborative Software Development and Technology Department Corporate Software Engineering & Technology Center)
 - Software development of embedded devices based on open source software for a while such as digital TV.
 - Since 2019 I have joined in current team and we are focused on developing database technologies required for accessing various data.



Today's talk

- Study of FDW applying for schema-less database
 - Motif for schema-less database : **InfluxDB**
 - In this talk, schema-less database means database like not requiring column definition before inserting data.

FDW (Foreign Data Wrapper) *

FDW is a standardized way of handling access to data stored in external data sources from SQL databases.

PostgreSQL can implement FDW as extension modules.

* https://wiki.postgresql.org/wiki/Foreign_data_wrappers

Agenda

- What is schema database
- What is InfluxDB
- Current InfluxDB FDW
- Goal
- Design
- Implementation
- Demonstration
- Consideration
- Conclusion

What is schema database

- In database terms, a schema is the organization and structure of database.
- A schema contains schema objects, which could be tables, **columns**, data types, views, stored procedures, relationships, primary keys, foreign keys, etc.

<https://database.guide/what-is-a-database-schema/>

There are many meanings for the term 'schema'.

But in this talk, I focuses only on **columns** as schema objects.

What is InfluxDB

- Timeseries database
 - It is easy to manage sensor data of IoT devices, log data etc.
- Elements of data

<https://docs.influxdata.com/influxdb/cloud/reference/key-concepts/data-elements/>

RDBMS		InfluxDB	Explanation
Table		Measurement	
Record		Point	
Column	have timestamp type	Timestamp (time)	Primary key. Point has always one timestamp.
Column	have index	Tag set	A pair data of tag key = tag value. All tags has index. Tags are optional, and string value.
Column		Field set	A pair data of field key = field value. Point should have at least one field set. Fields have integer/float/string/boolean value.

 **Column**

 **influxdb keys**

What is InfluxDB (cont.)

- Release versions
 - There are two major versions 1.x and 2.x.
 - But we are focusing on version 1.x at this moment.
- Query language (NoSQL)
 - **InfluxQL** is an SQL-like query language for InfluxDB.
 - There is yet another language **Flux** in InfluxDB.
 - We use InfluxQL because we think it is primary language on 1.x.
- Schema-less feature
 - Application can write new tag(s) and field(s) at anytime without changing schema operation like '**ALTER TABLE**'.

What is InfluxDB - Schema-less operation in InfluxDB

Step 1

```
INSERT s,DEVICE_ID=DEVICE1 SIG_A=1 0
```

time	tag	field
	DEVICE_ID	SIG_A
1970-01-01T00:00:00Z	DEVICE1	1



There are only **two** keys of DEVICE_ID and SIG_A.



Step 2

InfluxDB client application can add new columns without changing data definition explicitly.

```
INSERT s,DEVICE_ID=DEVICE1,SUB_ID=A SIG_A=2,SIG_B=0.4 1000000000
```

time	tag	tag	field	field
	DEVICE_ID	SUB_ID	SIG_A	SIG_B
1970-01-01T00:00:00Z	DEVICE1		1	
1970-01-01T00:00:01Z	DEVICE1	A	2	0.4



The schema is updated by **INSERT** operation.

Current InfluxDB FDW

- We have one FDW implementation of InfluxDB.
 - https://github.com/pgspider/influxdb_fdw
- Current spec
 - Support only scan operation (**SELECT**)
 - Support push-down WHERE clause and some aggregation
 - The tags and fields are mapped into columns of foreign table 1:1.
- Current problems
 - Support InfluxDB version 2.x
 - Support modify (insert/delete) operation
 - Improve usability for the InfluxDB schema change

Current InfluxDB FDW - Schema change

- When user writes a data point into InfluxDB, it might be added new tag and/or field keys.
- In this case, user has to update explicitly the foreign table corresponding to the latest measurement of InfluxDB.

Step1

Measurements in **InfluxDB**

time	tag	field
	DEVICE_ID	SIG_A
1970-01-01T00:00:00Z	DEVICE1	1

Step2

A data point is added having a new field key **SIG_B**

time	tag	field	field
	DEVICE_ID	SIG_A	SIG_B
1970-01-01T00:00:00Z	DEVICE1	1	
1970-01-01T00:00:01Z	DEVICE1	2	0.4

Create a table in **PostgreSQL**

CREATE FOREIGN TABLE

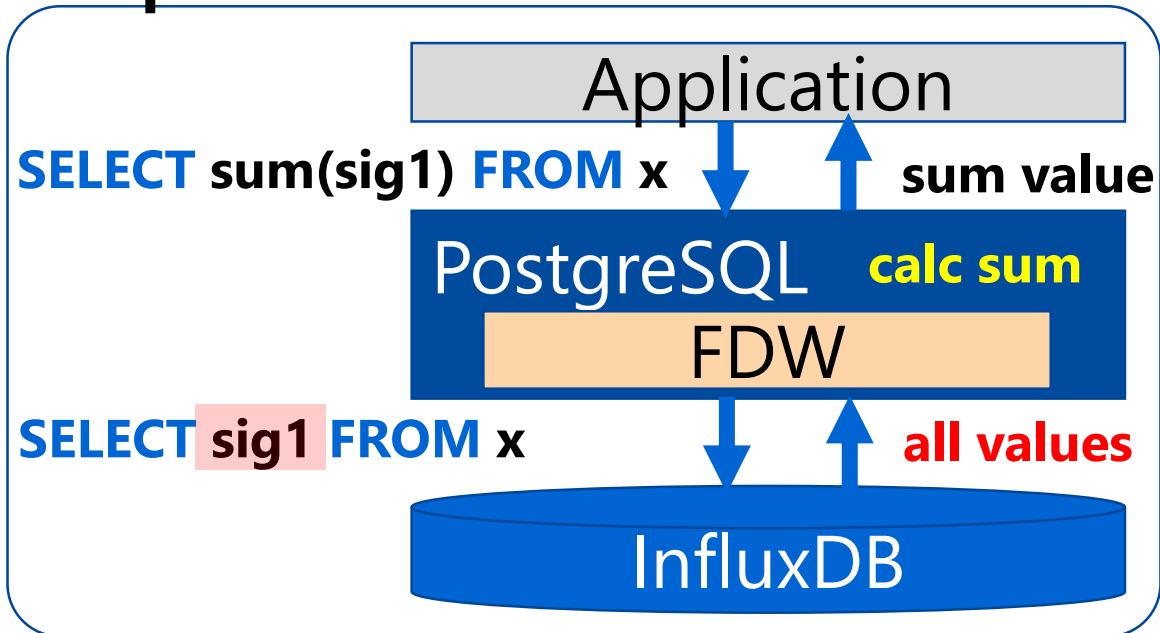
```
st (  
  time timestamp with time zone,  
  DEVICE_ID text,  
  SIG_A bigint  
) SERVER influxdb_svr OPTIONS (table 's');
```

The table is missing a column for SIG_B field.

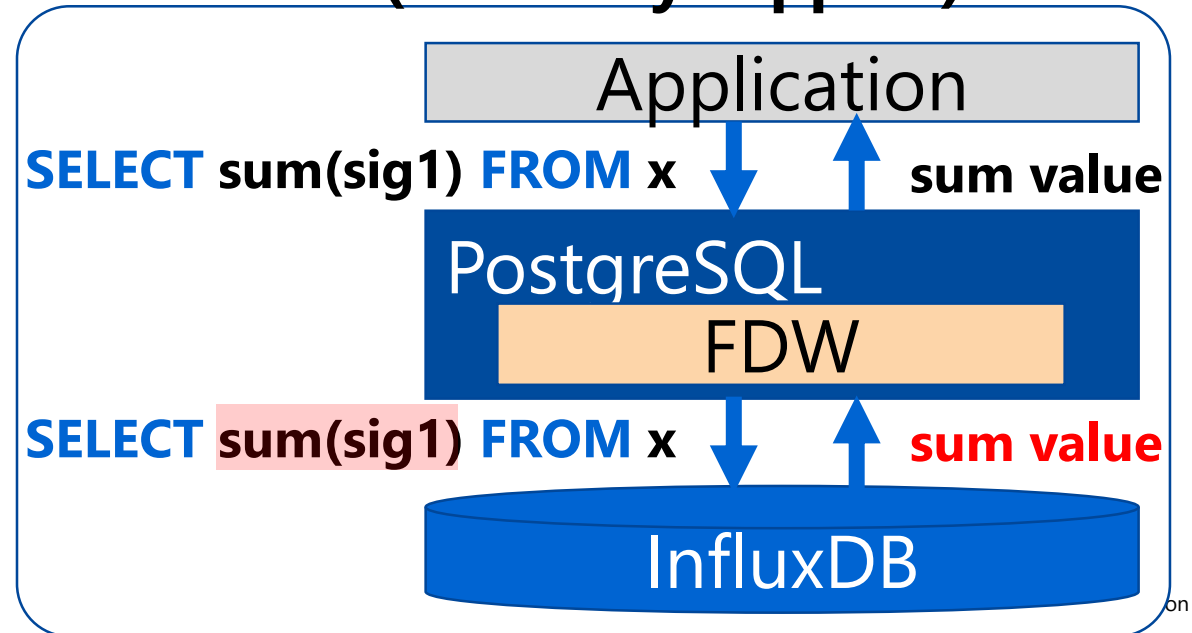
Goal

- Design and implementation of schema-less support in InfluxDB FDW
 - FDW should be able to access data from InfluxDB without knowing actual schema on InfluxDB and changing schema after table creation.
 - FDW should be able to push-down as possible as to prevent performance degradation even on schema-less design.

No push-down



Push-down (Currently support)



Design

- Fix the foreign table schema regardless of the InfluxDB measurement.
 - Against for tag and field set, we map each data set into unstructured data types based on **hstore** type (Two new types of **influxdb_tags** and **influxdb_fields**).
 - InfluxDB FDW needs to define two new types to distinguish between tags and fields during deparsing process because there are spec gaps between them on InfluxQL.

CREATE FOREIGN TABLE

```
st (  
    time timestamp with time zone,  
    tags influxdb_tags, -- This can refer any tags of data point.  
    fields influxdb_fields -- This can refer any fields of data point.  
) SERVER influxdb_svr OPTIONS (table 's');
```

Design - Definition of unstructured data types

<https://www.postgresql.org/docs/13/hstore.html>

- About **hstore** type
 - The **hstore** data type can be used to store sets of **key/value pairs** within a **single** PostgreSQL value as string.

ex.) `"col1"=>"1", "col2"=>"a"`

"col1" and "col2" are keys.

"1" is value of "col1" key and "a" is value of "col2" key.

Overview of definition of `influxdb_tags` (same for `influxdb_fields`)

```
CREATE TYPE influxdb_tags;
```

```
CREATE FUNCTION influxdb_tags_in(cstring)
```

```
RETURNS influxdb_tags AS '$libdir/hstore', 'hstore_in' -- use hstore function
```

```
LANGUAGE C STRICT IMMUTABLE PARALLEL SAFE;
```

Design - Definition of unstructured data types (cont.)

```
CREATE TYPE influxdb_tags (  
  ...  
  INPUT = influxdb_tags_in,  
  ...  
);
```

```
CREATE FUNCTION fetchval(influxdb_tags, text)  
RETURNS text AS '$libdir/hstore', 'hstore_fetchval' -- use hstore function  
LANGUAGE C STRUCT IMMUTABLE PARALLEL SAFE;
```

```
CREATE OPERATOR -> (  
  LEFTARG = influxdb_tags,  
  RIGHTARG = text,  
  PROCEDURE = fetchval -- use hstore function  
);
```

If possible, I thought it is better to have a way to define alias type using hstore like **"CREATE TYPE influxdb_tags ALIAS hstore"**.

Design - How to refer the influxdb keys

- Using arrow operator, "key"->"value" expression can be used to refer the "value" of the specified "key" within influxdb_tags and or influxdb_fields variables.

In existing design

```
SELECT  
time,  
DEVICE_ID,  
SIG_A,  
SIG_B
```

FROM

st;

Foreign table

```
time timestamp with time zone,  
DEVICE_ID text,  
SIG_A bigint,  
SIG_B double precision
```

In schema-less design

```
SELECT  
time,  
tags->'DEVICE_ID' DEVICE_ID,  
fields->'SIG_A' SIG_A,  
fields->'SIG_B' SIG_B
```

FROM

st;

Foreign table

```
time timestamp with time zone,  
tags influxdb_tags,  
fields influxdb_fields
```

The amount of description increases...

Design - How to get all data from foreign table

- In SQL "**SELECT ***" statement can get all data from a foreign table.
- In this query, we do not specify tag and/or field keys, so FDW is difficult to output values of keys in separate columns.
- This is not compatible output behavior with existing InfluxDB FDW.

ex.) **SELECT * FROM st;**

time	tags	fields
1970-01-01 00:00:00+00	"DEVICE_ID" => "DEVICE1"	"SIG_A" => "1", "SIG_B" => "0.4"

Design - How to support aggregation

- Support push-down aggregate function having arguments if influxdb_tags and/or influxdb_fields refer a const string using arrow operator '->'.
operator '->'.

```
ex.) SELECT COUNT(fields->'SIG_A') FROM st;
```

- Support also push-down aggregate function having arguments if influxdb_tags and/or influxdb_fields refer a const string using arrow operator '->' and they are **explicitly casted** as bigint, double precision and boolean. GROUP BY can be specified in the same way.

```
ex.) SELECT SUM((fields->'SIG_A')::bigint) FROM st  
GROUP BY (tags->'DEVICE_ID');
```

Design - Handling wrong referring keys

- InfluxDB behavior for non-existing influxdb keys.
 - InfluxDB does not treat as errors and do not return data.
 - FDW also apply the same policy as InfluxDB itself.

SELECT fields->'not_exist' **FROM** st; ➡ (no error, no data)

- If influxdb_tags and influxdb_fields are specified in reverse
 - Because InfluxDB behaves differently depending on tag or field, FDW should also change its behavior depending on whether it is tag or field.

Design - Handling wrong referring keys (cont.)

- However if the keys are specified incorrectly, FDW may not be able to get correct results or execute push-down efficiently.

```
SELECT fields->'tag' FROM st;
```



'tag' named field keys not existed in InfluxDB.

(no data)

```
SELECT count(tags->'field')  
FROM st GROUP BY tags->'field';
```



GROUP BY field key is ignored in InfluxDB,
so the result will not be expected.

wrong results

Design - Distinguishing tag and field keys in FDW

- There is a way to get **tag key names** by using dedicated query **SHOW TAG KEYS**. And FDW will get this **tag key names** by using this query only when **IMPORT FOREIGN SCHEMA** is executed.
- FDW can also determine the **tag key names** by **table option 'tags'** manually. But it is not easy to use.
- In the future, we'd like to automate detecting the tags key names.

Query result

time	DEVICE_ID	SIG_A	SIG_B
1970-01-01T00:00:01Z	DEVICE1	2	0.4

tag key names list

'DEVICE_ID', 'SUB_ID'

FDW recognize
DEVICE_ID is a tag key

Implementation

Callback routines required for scanning support in FDW

PostgreSQL

FDW

GetForeignRelSize

GetForeignPaths

GetForeignUpperPaths

GetForeignPlan

BeginForeignScan

IterateForeignScan

EndForeignScan

Need to know actual influxdb key names to use in remote query.

User query

```
SELECT fields->'SIG_A' FROM st;
```

Variable in foreign table

Remote query

```
SELECT "SIG_A" FROM st;
```

tag/field keys in InfluxDB measurement

1. Extract influxdb key names
2. Modify push-down decision to support unstructured expressions
3. Construct remote query (fields->'SIG_A' → "SIG_A")
4. Construct result data into unstructured type variable

Implementation - Extract influxdb key names

- Extract influxdb key names from target list
 - In GetForeignRelSize, FDW can get **influxdb key names** by extracting const string values which are referred as influxdb_tags and influxdb_fields variables from PlannerInfo's **processed_tlist**.
 - And the result is stored as a list in FDW private data later use.

```
SELECT tags->'DEVICE_ID', fields->'SIG_A', fields->'SIG_B' FROM st;
```

influxdb key names list = "DEVICE_ID", fields = "SIG_A", "SIG_B"

fields -> **'SIG_A'** VAR must be influxdb_tags or influxdb_fields type

```
{TARGETENTRY {OPEXPR ({VAR}{CONST})}}
```

Implementation - Modify push-down decision

- Modify push-down decision to support unstructured expressions

Expressions to allow push-down for arrow operator ->

ex.) **SELECT** fields->'SIG_A' **FROM** st;

{**OPEXPR** ({**VAR**}) {**CONST**}}

->

tags / fields

'SIG_A'

Operator

Variable influxdb_tags
 influxdb_fields

Constant

Expressions to allow push-down for arrow operator -> with explicit casting

ex.) **SELECT** SUM((fields->'SIG_A')::bigint) **FROM** st;

{{**COERCEVIAIO** {**OPEXPR** ({**VAR**}) {**CONST**}}}}

()::bigint

->

tags / fields

'SIG_A'

Implementation - Construct remote query

- Construct remote query for unstructured expressions

Scanning for simple base relation

ex.) `SELECT tags->'DEVICE_ID', fields->'SIG_A' FROM st;`



This can be realized by using **influxdb key names list** obtained former in `GetForeignRelSize`.

Remote query: `SELECT "DEVICE_ID", "SIG_A" FROM s;`

Scanning with aggregation

ex.) `SELECT SUM((fields->'SIG_A')::bigint) FROM st;`



This can be realized by deparsing expression "**{AGGREF ({TARGETENTRY {COERCEVIAIO {OPEXPR ({VAR}{CONST})})})}**" into "**Aggregate Function name ("Const")**".

Remote query: `SELECT SUM("SIG_A") FROM s;`

Implementation - Construct result row data


- Constructing result data into unstructured type variable
 - In IterateForeignScan, FDW queries to InfluxDB and obtained values of tags and fields are stored into influxdb_tags variable and influxdb_fields variable in each by using **tag key names** list.

ex.) **SELECT * FROM st;**



Remote query:

SELECT * FROM s;



time	tag	field	field
	DEVICE_ID	SIG_A	SIG_B
1970-01-01T00:00:01Z	DEVICE1	2	0.4

When we get row data like this.

Packing all tag set into influxdb_tags variable as key => value pairs separated by commas. It is same manner for influxdb_fields.

time	tags	fields
1970-01-01 00:00:01+00	"DEVICE_ID" => "DEVICE1"	"SIG_A" => "2", "SIG_B" => "0.4"

```
[rkuma@rkcent7 demo]$ ./psql.sh
```

```
psql (13.0)
```

```
Type "help" for help.
```

```
testdb=#
```

```
[rkuma@rkcent7 demo]$ ./influx.sh
```

```
2021/01/14 12:47:27 Processed 2 commands
```

```
2021/01/14 12:47:27 Processed 36 inserts
```

```
2021/01/14 12:47:27 Failed 0 inserts
```

```
Connected to http://localhost:8086 version 1.8.3
```

```
InfluxDB shell version: 1.8.3
```

```
>
```

Consideration

- Further verifications
 - Now I could execute only simple query variables and aggregates in PostgreSQL 13.0 with modified InfluxDB FDW.
- Improve updating of tag names list
 - FDW needs to know tag names in order to distinguish tags and fields, but FDW currently does not update automatically. So there is a room for improvement at this point.
- Improve defining of key-value types based on existing types
 - The new key-value types `influxdb_tags` and `influxdb_fields` are just copies of `hstore` type. So if we can define alias type for existing data types straightforward, I think it is easier to maintain.

```
CREATE TYPE influxdb_tags ALIAS hstore;
```

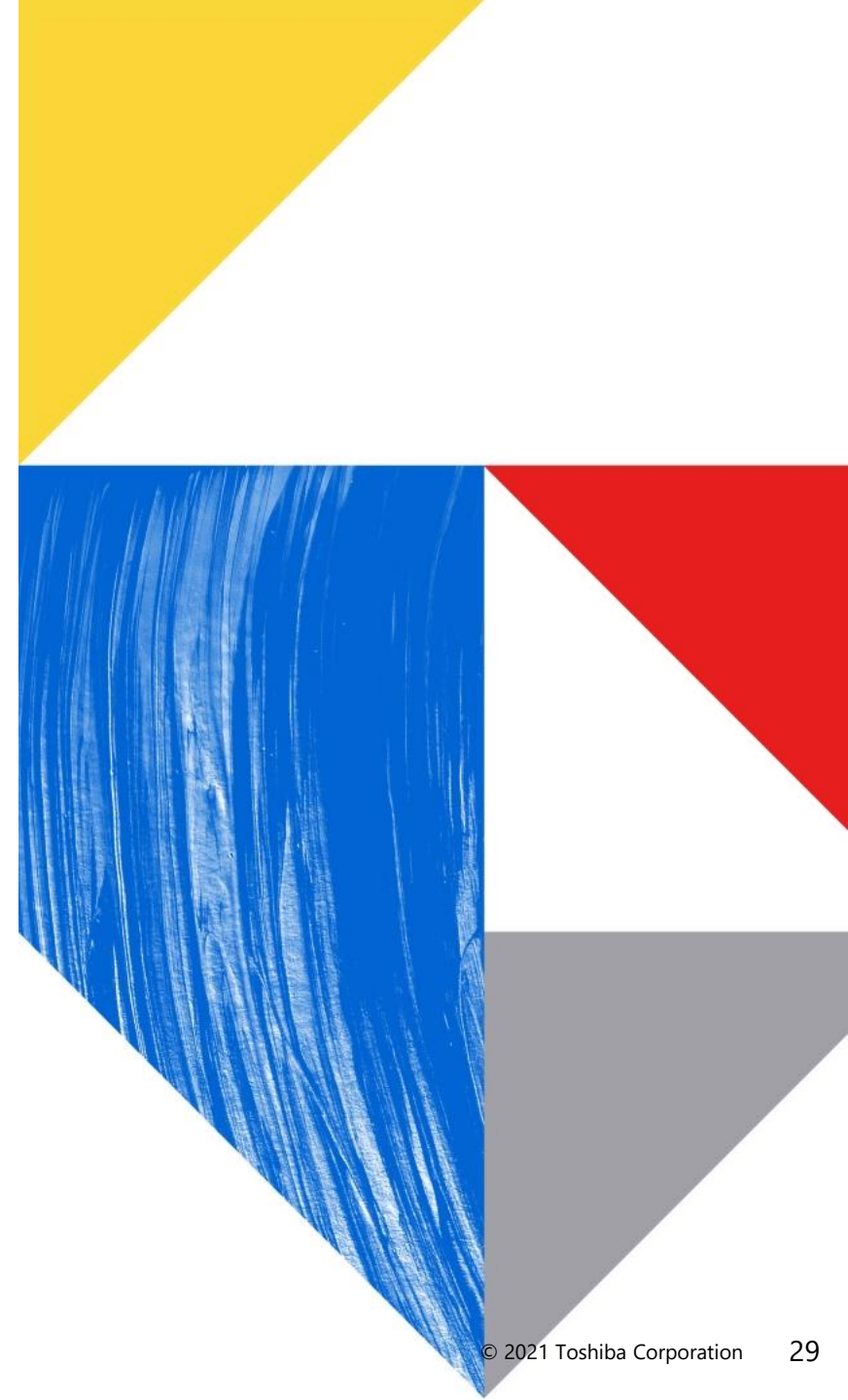
Conclusion

- Schema-less FDW design for InfluxDB
 - InfluxDB FDW can be designed for schema-less database by using unstructured data type based on **hstore**.
 - We do not need to **ALTER TABLE** anymore.
 - I think this design can be applicable to FDWs for other schema-less databases. But there will be cases it is suitable **json** type rather than **hstore** type for nested data structure.
- Schema-less push-down implementation
 - InfluxDB FDW could support schema-less feature based on existing FDW implementation without losing push-down functionality. It is important from performance point of view.

Thank you for listening.

Our project site:

<https://github.com/pgspider/>



TOSHIBA