Re-written Schema of databases and tables aka buckets and measurements as per InfluxDB Standards

I forgot to mention that InfluxDB is not an RDBMS (Relational Database Management System) and because of that database/table relations, Joints etc are not possible, unlike traditional SQL-like DBMS/RDMS. Also, there are different naming conventions and standards that InfluxDB follows, Here I'll re-write our existing schema as per those.

Some basic terminologies:

- Schema - How the data are organized in InfluxDB. The fundamentals of the InfluxDB schema are databases, retention policies, series, measurements, tag keys, tag values, and field keys. See Schema Design for more information.

- Retention Policy (RP) - Describes how long InfluxDB keeps data (duration), how many copies of the data to store in the cluster (replication factor), and the time range covered by shard groups (shard group duration). RPs are unique per database and along with the measurement and tag set define a series. When you create a database, InfluxDB creates a retention policy called autogen with an infinite duration, a replication factor set to one, and a shard group duration set to seven days.

- Timestamp - The date and time associated with a point. All time in InfluxDB is UTC but can be changed.

- Measurement - The part of the InfluxDB data structure that describes the data stored in the associated fields. Measurements are strings.

- Tag - The key-value pair in the InfluxDB data structure that records metadata. Tags are an optional part of the data structure, but they are useful for storing commonly-queried metadata; tags are indexed so queries on tags are performant. Query tip: Compare tags to fields; fields are not indexed.

- Tag key - The key part of the key-value pair that makes up a tag. Tag keys are strings and they store metadata. Tag keys are indexed so queries on tag keys are performant.

- Tag set - The collection of tag keys and tag values on a point.

- Tag value - The value part of the key-value pair that makes up a tag. Tag values are strings and they store metadata. Tag values are indexed so queries on tag values are performant.

- Field - The key-value pair in an InfluxDB data structure that records metadata and the actual data value. Fields are required in InfluxDB data structures and they are not indexed - queries on field values scan all points that match the specified time range and, as a result, are not performant relative to tags.

- Field key - The key part of the key-value pair that makes up a field. Field keys are strings and they store metadata.

- Field set - The collection of field keys and field values on a point.

- Field value -The value part of the key-value pair that makes up a field. Field values are the actual data; they can be strings, floats, integers, or Booleans. A field value is always associated with a timestamp. Field values are not indexed - queries on field values scan all points that match the specified time range and, as a result, are not performant.

InfluxDB Schema for /data/* and /archive/* feed records

For /data we will be having a bucket names cam_feeds with a retention policy of about 1 week. Under cam_feeds measurements of raw_feeds and calc_feeds will be there. We will be using feed_id as a tag key so that it can be indexable like a primary key. Field key will be used as key=value pair to store our records like feed_title, start_time, end_time, vid_path, and from_cam. After 1 week i.e., our retention policy, all data will be transferred from raw_feeds > arch_raw_feeds and calc_feeds > arch_calc_feeds.

Buckets	cam_feeds (Retention policy = 1 week)
Measurements	raw_feeds, calc_feeds, arch_raw_feeds, arch_calc_feeds
Tags Keys	feed_id (String, index)
Field Keys	• feed_title (String) • start_time (unixtime) • end_time (unixtime) • vid_path (String) • from_cam (String)

Let us take a sample data for only 1 raw feed for /data:

_time	_measurement	feed_id	_field	_value
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	feed_title	brm_stand1_therm_cam_1641062400_1641062700.mp4
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	start_time	1641062400
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	end_time	1641062700
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	vid_path	/data/raw/202209/640975400/
2022-01-01T00:00:00Z	raw_feeds	3eqenu3e3	from_cam	stand_1_thermal_cam

Schema:

```
raw_feeds, feed_id=3eqenu3e3, feed_title=brm_stand1_therm_cam_1641062400_1641062700.mp4, start_time=1641062400, end_time=1641062700, vid_path=/data/raw/202209/640975400/, from_cam=stand_1_thermal_cam
```

Same for calc_feeds:

```
calc_feeds, feed_id=3eqenu3e3, feed_title=brm_stand1_therm_cam_1641062400_1641062700.mp4, start_time=1641062400, end_time=1641062700, vid_path=/data/calc/202209/640975400/, from_cam=stand_1_thermal_cam
```

InfluxDB Schema for sensor data

Here we will be having all the data which are available from the installed sensors. Along with the other fields like _time, sensor_name, sensor_zone, sensor_type, and sensor_value.

Buckets	snsrs_data (Retention policy = 1 week)
Measurements	data
Tags Keys	snsr_id (String, index)
Field Keys	• snsr_name (String) • snsr_zone (String) • snsr_type (String) • snsr_value (String)

Let us take 1 sample data for sensor_data:

_time	_measurement	snsr_id	_field	_value
2022-01-01T00:00:00Z	data	89d5a82s	snsr_name	infrared_sensor_xyz_inc_bsp
2022-01-01T00:00:00Z	data	89d5a82s	snsr_zone	brm_stand_16
2022-01-01T00:00:00Z	data	89d5a82s	snsr_type	infrared_snsr
2022-01-01T00:00:00Z	data	89d5a82s	snsr_value	846.694546221949562

Schema:

```
data, snsr_id=89d5a82s, snsr_name=infrared_sensor_xyz_inc_bsp, snsr_zone=brm_stand_16, snsr_type=infrared_snsr, snsr_value=846.694546221949562
```

As influxDB will automatically input a _time whenever the record is entered, we will not have a different field for a timestamp and I'm assuming that there is no delay between the actual timestamp when a sensor has taken a value and the value of timestamp when the recorded is inserted, although ill makes some tests.

InfluxDB Schema design for abnormal pattern records

We are going to store records which are abnormal or fall under our defined abnormal pattern category. This data will help us in future regarding analysis and study of the data wherever, when and how the cobble or any other incident has happened.

Buckets	abnormal_pattern (Retention policy = 1 week)
Measurements	data
Tags Keys	patt_id (String, index)
Field Keys	• patt_type (String) • patt_snsr (String) • patt_zone (String) • patt_occ_at (unixtime) • patt_occ_till (unixtime)

Let us take 1 sample data for abnormal_pattern:

_time	_measurement	pattern_id	_field	_value
2022-01-01T00:00:00Z	data	5sd5sd68	patt_type	cobble_med_brm
2022-01-01T00:00:00Z	data	5sd5sd68	patt_snsr	infrared_snsr
2022-01-01T00:00:00Z	data	5sd5sd68	patt_zone	brm_stand_16
2022-01-01T00:00:00Z	data	5sd5sd68	patt_occ_at	1641062400
2022-01-01T00:00:00Z	data	5sd5sd68	patt_occ_till	1641062700

Schema:

```
data, patt_id=89d5a82s, patt_type=cobble_med_brm, patt_zone=brm_stand_16,
patt_occ_at=1641062400, patt_occ_till=1641062400
```

Workaround to relate records with InfluxDB

As I have mentioned that influxDB is not an RDBMS and because of that relations aren't possible with influxDB. I was thinking to make a separate index for our available camera, sensors, and defined patterns (events) in our schema based on JSON. This will help us to have an index of those along with their IDs which are also a tag key in our influxDB records.

JSON schema sample for the camera:

```
{
    "cam_id": "5asd55as5",
    "cam_name": "thermal_cam_sony_inc",
    "cam_type": "thermal",
    "cam_zone": "brm_stand_16",
    "usr_note": "This camera is for xyz purpose",
    "specs": {
        "sensor_technology": "CMOS",
        "supported_interface": "GigE Vision",
        "spectrum_capability": "Visible (400-700 nm), Near Infrared (700-1000 nm)",
        "spectrum": "MONO/NIR",
        "pixel_size": "4.8 µm",
        "resolution": "5120 x 5120",
        "max_frame_rate": "51fps",
        "vendor": "Sony Visual Imaging Solutions"
    }
}
```

JSON schema sample for our available sensors:

```
{
      "snsr_id": "9b9t2t5f",
      "snsr_name": "gas_snsr_ada_inc",
      "snsr_type": "Gas Sensor",
      "snsr_zone": "brm_stand_16",
      "snsr_note": "This sensor is for xyz purpose",
      "specs":
            {
            "communication_interface": "I2C/PLC/OPC",
            "measuring_range": "1 lux to 65535 lux",
            "precision": "±(50+5% reading)",
            "operating_voltage": "4.5 ~ 20",
            "operating_condition": "-10℃~+50℃; 0~95%RH (non-condensing)",
            "delay_time": "Default 8S + -30% (can be customized)",
            "vendor": "Adafruit"
      }
}
```

JSON schema sample for our pre-defined events:

```
{
      "patt_id": "9b9t2t5f",
      "patt_name": "gas_snsr_ada_inc",
      "patt_type": "Gas Sensor",
      "patt_note": "This sensor is for xyz purpose",
      "patterns":
            {
                  "by_senors":
            [
                {
                    "snsr_id": "56sd65363",
                    "snsr_def_th": "55.45455956",
                    "snsr_rec_th": "89.56556565"
                },
                {
                    "snsr_id": "9sd8ee5w2",
                    "snsr__defth": "95.14245544",
                    "snsr_rec_th": "89.56556565"
                }
            ]
            },
    "patt_time_at": "1641062400",
    "patt_time_till": "1641062700"
}
```

Note: If we need to get more data about the sensor, camera, and more about events we can get it from these indexes using the IDs which are used as tags in our influxDB.

Security: In influxDB, we have token/password-based authentication for security, but to keep these index data secure we can limit permissions for the user using the UNIX system permission concept also we can encrypt these data using Encryption methods like AES. Encryption, Integrity, Security etc are very important for us and I'll write and explain more about it later in the other document.

References:

- https://docs.influxdata.com/influxdb/v2.4/reference/key-concepts/data-elements/
- https://docs.influxdata.com/influxdb/v2.4/reference/key-concepts/data-schema/
- https://www.influxdata.com/blog/data-layout-and-schema-design-best-practices-for-influxdb/
- https://docs.influxdata.com/influxdb/cloud/write-data/best-practices/schema-design/