SQL-like Query / Control for the Internet of Things

Here's a sketch of some ideas for an IoT control language, similar to SQL, built on the IOTDB semantic definitions

Conventions

id

`id` is always the thing-id of a Thing

Bands

Band documentation here. There are multiple sets of data associated with any one Thing.

- meta: the metadata
- ostate: the "output state" - what we want a Thing to do, what we want it to transition to
- istate: the "input state" - the actual state of a Thing
- state: when reading, the istate; when writing, the ostate. This is usually, except for certain edge cases, the right thing to do.

Facets

These come from here: https://iotdb.org/pub/iot-facets

Units

These come from here: https://iotdb.org/pub/iot-units

Operators

Assume Pythonic Truth: i.e. false, 0, [], {} and NULL are all False; everything else is True.

This takes a very "semantic web" view toward lists. Some items can have multiple values, sometimes they are a single value. We have to deal with this gracefully.

- LIST(y): if y is a list, y; otherwise [y]
- x IN y: true if x is an element of LIST(y)
- x & y: the intersection of LIST(x) and LIST(y)
- x | y: the union of LIST(x) and LIST(y)

Lists and Dictionaries

JSONish. Not sure what this means yet.

Examples

Turn on everything

```
SET
    state.on = true
```

That's probably a little extreme for most people!

Mappings:

- `state.on` → OSTATE(iot-attribute:on)

Turn on "Desktop Lamp"

```
SET
    state.on = true
WHERE
    meta.name = "Desktop Lamp"
```

Mappings:

- `state.on` → OSTATE(iot-attribute:on)
- `meta.name` → META(schema:name)

The IOTQL knows that certain words gets mapped into different namespaces.

Set all lights in the basement to half-bright

```
SET
    state.brightness = 50%
WHERE
    meta.zone & "Basement"
AND
    meta.facet & facets.lighting
```

Originally we had "=" instead of "&", but it's not really the operator we want to do. Theoretically there's a list on both sides. The "&" operator is to test for intersection of lists. Items that are not lists are cast to lists.

Mappings:

- `state.brightness` → OSTATE(iot-attribute:brightness)
- `meta.zone` → META(iot:zone)
- `,meta.facet` → META(iot:facet)
- `facets.lighting` → iot-facet:lighting
- % → a value between 0 and 100, equivalent to UNITS(#,iot-unit:math.fraction.percent)

Get the temperature

```
SELECT
    state.sensor.temperature
```

as a variant

```
SELECT
    state.sensor.temperature AS temperature
```

Mappings:

- `sensor.temperature` → ISTATE(iot-attribute:sensor.temperature)

Note - what do we do with Things that don't have the attribute `sensor.temperature`? If a row is all NULL values, the row should be discarded from the results.

Get the temperature in Celsius, only from HVAC equipment

```
SELECT
    UNITS(state.sensor.temperature,units.temperature.metric.celsius)
WHERE
    meta.facet & facets.climate
```

Mappings:

- `state.sensor.temperature` → ISTATE(iot-attribute:sensor.temperature)
- `units.temperature.metric.celsius` → iot-attribute:temperature.metric.celsius
- `facets.climate` → iot-facet:climate
- `meta.facet` → iot:facet

Set the temperature in the basement to 68F

```
SET
    state.temperature = UNITS(68,units.temperature.imperial.fahrenheit)
WHERE
    meta.zone & "Basement"
AND
    meta.facet & [ climate.heating, climate.cooling ]
```

- `sensor.temperature` → ISTATE(iot-attribute:sensor.temperature)
- `zone` → META(iot:zone)
- `facet` → META(iot:facet)
- `climate.heating` → iot-facet:climate.heating
- `climate.cooling` → iot-facet:climate.cooling

Note JSON list structure!

Get the name of everything

```
SELECT
    id, meta.name
```

Change the name of something

```
SET
```

```
    meta.name = "Desktop Lamp"
WHERE
    meta.name = "Downstairs Lamp"
```

Get everything that is on

```
SELECT
    id, meta.name
WHERE
    state.on = true
```

Note that since we accept Pythonic type trues, so we could also do

```
SELECT
    id, meta.name
WHERE
    state.on
```

Create a scene with multiple actions

```
CREATE SCENE
    goodnight
BEGIN
    SET
        state.temperature = UNITS(18,units.temperature.metric.celsius)
    AND
        meta.facet & facets.climate.heating
    ;
    SET
        state.on = false
    WHERE
        meta.zone & [ "Basement", "Main Floor" ]
    AND
        meta.facet & lighting
END
```

Create a scene with one action and an argument

```
CREATE SCENE
    lights(value)
```

```
SET
    state.on = :value
WHERE
    meta.facet & lighting
```

Run a scene

```
DO goodnight
```

Create a trigger

```
CREATE TRIGGER
    front_door_light
SELECT
    state.open = true
WHERE
    meta.name = "Front Door Contact Switch"
BEGIN
    SET
        state.on = true
    WHERE
        meta.name = "Front Door Light"
        ;
    DELAY
        MINUTES(10)
    SET
        state.on = false
    WHERE
        meta.name = "Front Door Light"
        ;
END
```

The main issue with this one is if the lights are already on, or somewhere turns them on, they'll turn off because of this rule. There almost has to be some sort of "incrementing" system.