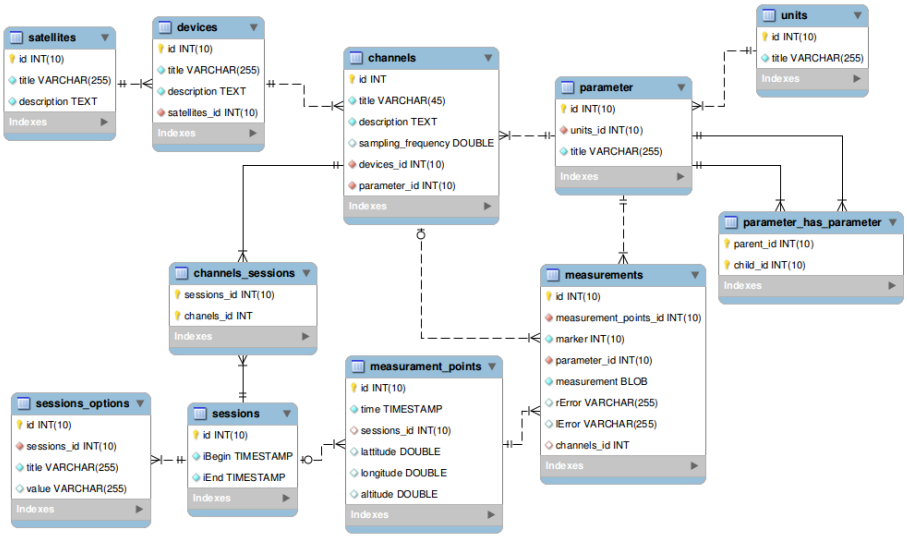
1 Data flow

1.1 Database



1.1.1 Basic information

Current project has a pretty complicated database structure, which is supposed by class of the information it designed for. Let's briefly overview its structure and comment some mostly important things which could impress you with its "first view" irrationality.

- *Satellites* – basic information about satellite projects. It's also works for measurement projects which have no satellites but still support us with helpful information, title – unique description.
- *Devices* – basic information about devices, which placed on a board of satellites. Each device has a few channels of measurements.
- *Channels* – describes a measurements channels of existed device. Each channel of device emit data of same type, characteristic and parameters. For example, if our channel measure a spectrum it means it always should be "HF" or "LF" but not both.
- *Sessions* – contains measuring sessions.
- *Session options* – should contain some specific information about session. It could be some supporting information which given by satellite project.

As you see there is many to many relation. This leads to:

1. one channel could be in a few sessions. And to be honest it has logical background, cause channel is a kind of stream of data which could be divided into time-intervals which represents by sessions.

2. one session could be shared by a few channels, because those channels could stream in a same time.

- **Толик, тут нужно написать про заявки и привести несколько примеров организации добавления этих заявок, зачем они нужны, и т.д. :)**
- **ну так напишіть =)**

1.1.2 User immutable tables

There a tables which could be edited only by administrative team and accessible for other users in *read-only* mode. There a list this tables:
satellites, devices, channels, parameters, units

1.1.3 User mutable tables

Tables which could be filled with user information via pulling `json`-file:
sessions, sessions_ options, measurement_ points, measurements

2 `JSON` protocol

Detailed information about `json` format you can read in wikipedia and go deeper into specification on official web-site . But mission of this document is to introduce our work format of `json`-files. It requires light understanding of `json`.

2.1 Data writing procedure

Any type of data represented in this database should be connected to specific satellite, device and parameter, but it could be released from session and channel. This specific appears because we also need to have some non-satellite data in this database like K_p, D_{sp}, A_p and $IGRF$ or second level data (processed data).

2.2 Adding sessions

2.3 Reconnect data with session

2.4 Adding data

Any data measurements should be connected with measurement points which describes by 4-dimensional point in time-space space.

2.5 Measurement point

Each measurement point is like an anchor in a space-time where you connect measurements as a values which was taken in that exact place and moment. That's why we suggest to use variables familiar to us, you might be interested in a several links about them: UT, latitude, longitude, altitude.

```
"measurement_points": [
    {
        "time": 234345563.6767, // UT
        "latitude": 45.45,      // Optional field for a while
        "longitude": 56.56,     // Optional field for a while
        "altitude": 3.44,       // Optional field for a while
    },
    {...}
]
```

Listing 1: Measurement point example

{...} – means that we can add a few more points in there to push some sequence.

2.6 Measurement

Then you have measurement points you can attach data measured there. But every data should be supported by some additional information:

marker – describes level of data processing, which starts from 0, where 0 is unprocessed "still" data

measurement – exact measured value we are craving

rError – right side error, length of right side of estimation interval

lError – left side error, length of left side of estimation interval

```
{
    "scope": {
        "parameters": "current",
    }
    "measurements": [
        {
            // <INT type> describes data level marker,
            // default is 0 - which means source %data
            "marker": 0,

            // <BLOB type> contains exect measurement
            "measurement": "smthn",

            // <BLOB type> contains right error
            // Optional field for a %while
            "rError": "r smthn",

            // <BLOB type> contains
            // Optional field for a %while
```

```
            "lError": "l smthn",
        },
        {...}
    ]
}
```

Listing 2: Measurement example

2.7 Session free flag

During measurements insertion session relation would be defined automatically, but here could be case when some measurement have no session which it belongs to. To work out this issue exists the way represented here.

`allowSessionFree` – allows you to add session free points, which could be reconnected later with adding session information or stay session free for ever.

```
"allowSessionFree": False // Default is False
```

Default value is `False` which means that points without session information couldn't be inserted into database and would throw an exception:

```
"sessionFreeException": {
    "message": "Something about how smart it is to try insert
        data without any session",
    "data": [
        {
            "measurement_point": {...}
            "measurement": {...}
        }
    ]
}
```

Listing 3: Example of exception

There present all information about points which no session, and data belongs to them.

2.8 Scope parameter

Could be place everywhere and changes key-parameters inside this scope. As key parameters could be the titles of objects from immutable tables (1.1.2). This thing should have next statement:

```
"immutable_table": "title_of_value"
```

There is:

`immutable_table` – name of immutable table which parameter you need to change in a scope

`title_of_value` – value of title field from this table

Let's take a look how it should be in a work:

```
"scope" : {
    "satellites": "Variant",
    "devices": "magnitometer",
    "channels": "Bz",
}
```

Listing 4: Scope parameter

2.9 Complete example

3 Brutal example

```
{
    "scope" : {
        "satellites": "Variant",
        "devices": "telemetria",
        "channels": "Bz",
    },
    "measurement_points": [
        {
            "time": 234345563.6767,
            "latitude": 45.45,
            "longitude": 56.56,
            "altitude": 3.44,
        },
        {
            "time": 45345563.6767,
            "latitude": 45.45,
            "longitude": 56.56,
            "altitude": 3.44,
        },
        {
            "time": 56745563.6767,
            "latitude": 45.45,
            "longitude": 56.56,
            "altitude": 3.44,
        },
    ],
    "measurements": [
        {
            "scope": {
                "parameters": "current",
            }
            "marker": 0, // <INT type> describes data level marker,
                 default is 0 - which means source data
            "measurement": "smthn", // <BLOB type> contains exect
                measurement
            "rError": "r smthn", // <BLOB type> contains right
                error
            "lError": "r smthn", // <BLOB type> contains left error
        },
        {...}
    ]
```

```
39 }
```

Listing 5: Brutal example