**Proposal to use Git and the Github platform in the development of algorithms (Concepts) in the Detection of medical claims.**

**Author:** *ProDev Developr Team*

**Problems that exist in the development of algorithms:**

* Spaghetti code
* Comments within the same code (Give example).
* Code that previously worked is impossible for the developer to obtain again because it was overwritten.
* Compare different versions of the same code to determine the error.
* Due to the nature of the data that is constantly updated, at one point the data is obtained from a version of the data and then as time passes (months, days) no data is obtained.
  + This casts doubt on the effectiveness of the QA and developer's work.
* Code in different formats (small letters, large letters, chamfered notation, etc.) and styles that make it very difficult for other developers to understand.
* A huge learning curve.
* Algorithms (BigQuery script) without documentation (White Paper) which does not allow studying or knowing the logic of the algorithm.

**Solution proposal**

Use [git](https://www.atlassian.com/git/glossary) and [github](https://github.com/) as a system and repository and version control.

- Use github actions as a Continous Integration / Continuous Deployment (CI/CD) system for validation and checking of the source code of the algorithms.

- Proposed directory hierarchy and source code:

PORA

**README.md**

**<general\_docs>**

* ***sql\_code\_style.md***

**<schema\_name\_in\_BigQuery>**

* *<algorithm\_name\_1>*
  + ***algorithm\_name\_1.sql***
  + **<WP>**
    - *algorithm\_name\_idAlgoritm\_wp.docx*
  + **<TABLES>**
    - *finalPool\_v2.csv*
  + **<OUTPUTS>**
    - *result\_algorithm\_name.csv*

…..

* *<algorithm\_name\_n>*

**Glossary:**

**<general\_docs>:**

***sql\_code\_style.md***

**<schema\_name\_in\_BigQuery>:**

*<algorithm\_name\_1>*

***algorithm\_name\_1.sql***

**<WP>**

*algorithm\_name\_idAlgoritm\_wp.docx*

**<TABLES>**

**<OUTPUTS>**

**Approach to define the master branch of the repository in github:**

Each Jira issue is associated with a branch named ***#issue\_name*** of the algorithm that masters the main structure and the rest of the code.

* When developing an algorithm, whenever a commit is made to the text, the **#issue** associated with Jira must be colored at the end so that any progress is commented and it is easy to relate and access from Jira.
* To develop a new algorithm or fix a bug in one already created:
  + Create a new branch of the main code (master) whose name will be the name of the algorithm concatenated with the number of its associated issue.
  + Respect the code style rules for SQL (See **Annex 1**).
  + Follow the directory tree structure proposed above.
  + When the algorithm is completed, a pull request (PR) must be created against the master branch.
  + For a PR of an algorithm. If the QA process has been completed, then the PR is approved, but not before reviewing compliance with the code style and other requirements.
  + While the source code of an algorithm is in the PR process, a lead team developer or designated staff will review the code and check compliance with the code style and may make suggestions to the developer.
  + Only authorized or privileged personnel will mix the master (if decided, it could be the same developer due to the level of trust, although it is not suggested).

* Perform a push on the branch in which the source code of an algorithm is developed.
  + If you have test tables, you must create them in the test scheme.
  + Create or update the algorithm in the test scheme. (Automated with GitHub Actions).
  + Run the algorithm to validate that the code is syntactically correct. (Automated with GitHub Actions).
  + Get the output to create Sample\_File or Final File (Automated with GitHub Actions).
  + Save the output in csv files of type Sample\_File (Automated with GitHub Actions).

***Annex 1 Approach for SQL code style in Algorithms***

***## 1. General***

- 4 spaces for indenting, no tabs.

- No trailing whitespace.

- 80 characters per line. If a couple of characters fall off the end of the screen, it is okay.

- Always capitalize [SQL Commands](https://www.postgresql.org/docs/current/static/sql-commands.html) (e.g., `SELECT`, `AS`, `TO`, `IS`, `NULL`).

- Don't capitalize [functions](https://www.postgresql.org/docs/current/static/functions.html) (e.g., `count`, `max`).

- Don't use CamelCase capitalization.

- Variable names should be underscore separated (`SELECT sum(t.foo) AS foo\_tot FROM table1 t`).

- Comments should go near the top of your query, or at least near the closest `SELECT`.

- Set default value for bool datatype.

- Database `bool` column should be `NOT NULL DEFAULT false` always.

***## 2. Naming***

- Use singular nouns for table names since they describe the entity the table represents.

- Make table, column and constraint names lowercase.

- When having logical noun/verb language separations, component names will be separated by the underscore character (`\_`) to indicate the separation, e.g. `date\_of\_birth`.

- Index names should begin with the name of the table they depend on, e.g. `address\_id\_idx`.

- Add \_uq suffix to unique index, e.g. `address\_parcelkey\_uq\_idx`.

- Add pk\_ prefix to primary key, e.g. `pk\_address`.

- Foreign key should contain main and referenced table names with fk\_ prefix and `\_ref\_` abbreviation between table names, e.g `fk\_address\_ref\_building`.

- Use lowercase column types (`ALTER TABLE address ADD COLUMN xref\_id varchar(32) NOT NULL`).

***## 3. Query syntax***

Always rename aggregates and function-wrapped columns. Also always use `AS` to rename columns:

```sql

SELECT

    t.name, sum(t.amount) AS sum\_amount

FROM

    table1 t;

```

- Only one table should be in the `FROM`. Never use `FROM`-joins.

- WITH, SELECT, FROM, WHERE, GROUP BY, HAVING, etc clauses must be placed on the same indentation level.

- Explicitly use `INNER JOIN` not just JOIN, making multiple lines of `INNER JOIN`s easier to read.

- Always use short meaningful table aliases.

```sql

SELECT

    t.code AS building\_code,

    count(a.id) AS addresses\_count

FROM

    building t

        INNER JOIN address a ON (t.code = a.building\_code)

GROUP BY

    t.code;

```

Multiple `WHERE` clauses should go on different lines and begin with the SQL operator:

```sql

SELECT

    t.name

FROM

    table1 t

WHERE

    t.created\_by = 123

    AND t.description LIKE '%123%';

```

`DISTINCT ON (cols)` should be used instead of `DISTINCT`

```sql

SELECT

    DISTINCT ON (t.public\_name)

    t.public\_name

FROM

    admin\_account t

ORDER BY

    t.public\_name;

```

***### Additional Examples***

```sql

-- simple one line select

SELECT t.name FROM table1 t WHERE t.id = 1;

-- standard select

SELECT

    t1.id, t1.name, t2.name AS table2\_name

FROM

    table1 t1

        INNER JOIN table2 t2 ON (t1.id = t2.table1\_id)

WHERE

    t1.created\_by = 123

    AND t1.description LIKE '%123%'

    AND t1.created\_by IN (

        SELECT

            u.id

        FROM

            users u

        WHERE

            u.name LIKE '%John%'

    )

ORDER BY

    t1.name DESC;

-- VALUES list

SELECT

    t.id, t.name

FROM

    (

        VALUES

            (1, 'name 1'),

            (2, 'name 2')

    ) t (id, name);

-- CASE formatting

SELECT

    CASE

        WHEN p.description LIKE '%MAINTAIN%' THEN 'permit'

        WHEN pt.name LIKE '%FIRE ALARM%' THEN 'permit'

        WHEN ut.name LIKE '%FIRE ALARM%' THEN 'unit'

    END AS item\_type

FROM

    ...

-- single insert

INSERT INTO table1 (

    name, description

) VALUES (

    'name1', NULL

);

-- multiple inserts

INSERT INTO table2 (

    name, description, ...,

    created\_at

) VALUES

    ('name1', 'descr1', ..., LOCALTIMESTAMP),

    (

        'name2', 'descr2', ...,

        LOCALTIMESTAMP

    );

INSERT INTO table2 (name, description)

    SELECT

        t.name, NULL AS description

    FROM

        table3 t;

INSERT INTO table2 (

    name, description, ...,

    created\_at

)

    SELECT

        t.name, t.descr AS description, ...,

        LOCALTIMESTAMP AS created\_at

    FROM

        table3 t;

UPDATE table1

SET name = 'name2',

    description = 'description2'

WHERE id = 123;

DELETE FROM table1

WHERE id = 123;