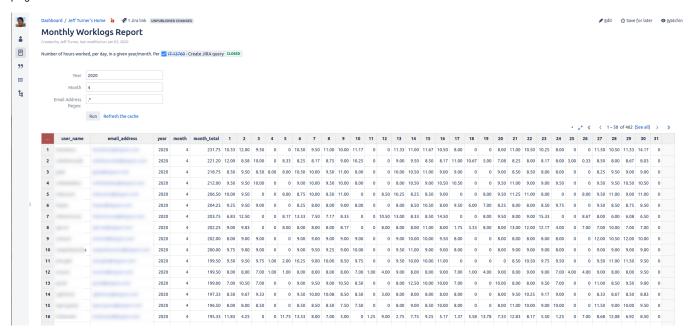
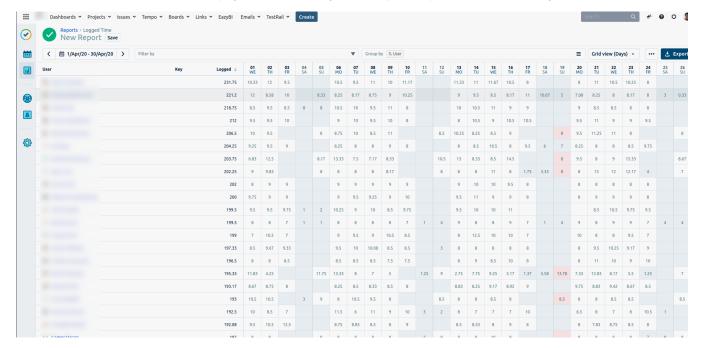
Creating interactive Jira reports in Confluence using free tools

This is a tutorial on how to use Confluence as a query / reporting engine, querying SQL data sources like the Jira database. For our example we query JIRA's database to build a Monthly Worklogs Report, showing hours worked per day for every user in a given month. We use the free Play SQL Base plugin.



Of course, Tempo Timesheets is the de-facto plugin for this sort of thing, and already has a report like what we're building:



Tempo's report is prettier and more powerful, allowing hours to grouped by any field (e.g. project, or tempo Account), even hierarchically. Tempo's one deficiency here, which motivated this reimplementation, is that it cannot show users which have not logged any work. Tempo's also honors Tempo's evil 'View All Worklogs' permission, which I consider an anti-feature.

But for the purposes of this tutorial, worklog information is just a nice example of *something* in the Jira database which you'd like to query in an interactive manner

Implementation

Choosing a Confluence SQL plugin

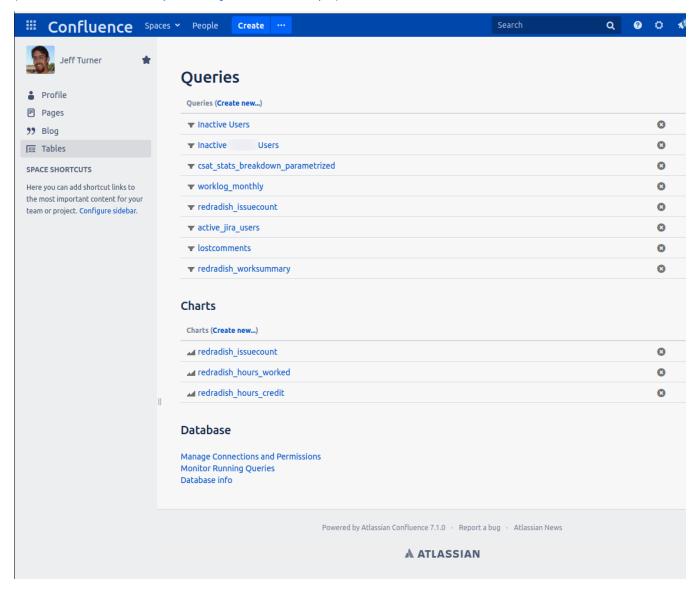
For this tutorial we are using the free Play SQL Base plugin. You could alternatively use PocketQuery or SQL for Confluence, which are in fact better plugins overall - in particular, they let you restrict who can run SQL queries, whereas Play SQL can't.

This tutorial uses Play SQL Base because it's what I had available. We will restrict SQL queries at the Postgres layer, which is a good thing to do anyway.

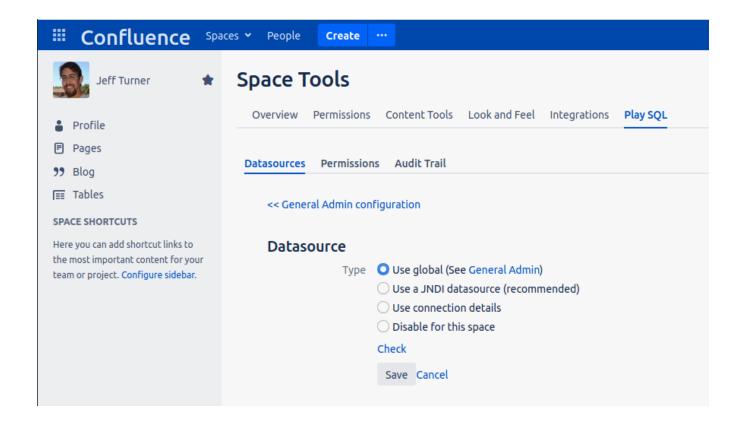
Configure Play SQL Base

In Confluence, type 'gg', 'Find new apps' and install the free Play SQL Base plugin.

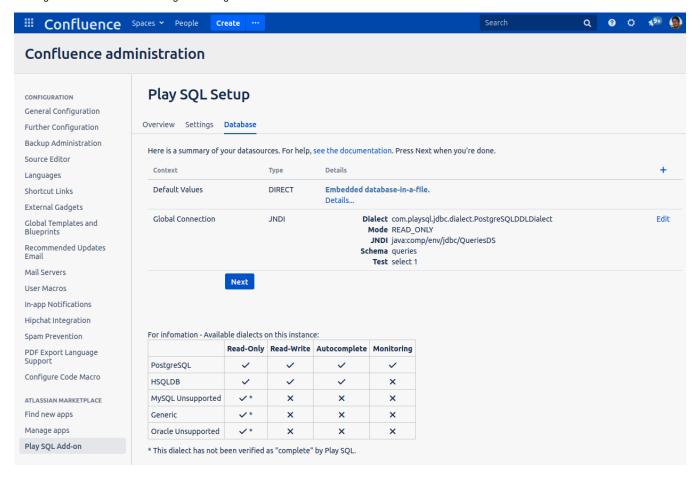
In Confluence spaces you will now see a new 'Tables' menu item. Here is the page from a live Confluence instance, with various queries already defined (there's one from the Automatically deactivating inactive Jira users report):



Click 'Manage Connections and Permissions' and set up the space's database connection. Here we just use the global datasource:



Clicking 'General Admin' shows the global config:



Creating a Postgres read-only account

At this point we're about to tell Play SQL how to connect to our database. For the sake of security, we want to connect as a user with **read-only** permission s, and with **visibility restricted** to just data necessary for our report.

The **read-only** requirement can be achieved with Postgres permissions. The **restricted visibility** requirement can be achieved by only allowing queries of predefined views, in a custom queries schema. The main Jira tables in the public schema will be inaccessible.

First, create a 'queries' schema, with a sample view containing a small amount of data:

```
root@jturner-desktop:~# su - postgres
postgres@jturner-desktop:~$ psql redradish_jira
Null display is "".
Line style is unicode.
Border style is 2.
psql (12.2 (Ubuntu 12.2-4))
Type "help" for help.
redradish_jira=# CREATE SCHEMA IF NOT EXISTS queries;
CREATE SCHEMA
redradish_jira=# CREATE OR REPLACE VIEW queries.sample AS select project.pkey || '-' || jiraissue.issuenum AS
key, summary from public.project JOIN public.jiraissue ON project.id=jiraissue.project LIMIT 5;
redradish jira=# select * from gueries.sample;
   key
                           summary
 SOC-3
          A second Response for good measure
          Ongoing Atlassian Product Support, 2014
 IC-34
          Invoice 93236 - 1/Jul/15 to 30/Sep/15
 JTODO-19 Tax Payment Q2 Due
 CLIC-2
        Move projects to OnDemand
(5 rows)
```

Next, create a <code>jira_queries_readonly</code> role that can only view the <code>queries</code> schema tables, and a <code>confluence_reports</code> user granted that role. These commands are cribbed shamelessly from https://blog.redash.io/postgres-readonly/, so read that to understand them properly. Run them when connected to the Jira database, <code>not</code> the default 'postgres' database.

```
CREATE ROLE jira_queries_readonly;

GRANT CONNECT ON DATABASE redradish_jira TO jira_queries_readonly;

GRANT USAGE ON SCHEMA queries TO jira_queries_readonly;

GRANT SELECT ON ALL TABLES IN SCHEMA queries TO jira_queries_readonly;

CREATE USER confluence_reports WITH PASSWORD 'confluence_reports';

GRANT jira_queries_readonly TO confluence_reports;
```

Verify that, when connecting as confluence_reports we can see our sample query but not generic Jira tables:

```
# PGUSER=confluence_reports PGPASSWORD=confluence_reports PGHOST=localhost PGDATABASE=redradish_jira psql -tAc
"select count(*) from queries.sample;"
5
# PGUSER=confluence_reports PGPASSWORD=confluence_reports PGHOST=localhost PGDATABASE=redradish_jira psql -tAc
"select count(*) from public.jiraissue;"
ERROR: permission denied for table jiraissue
```

Define a Datasource in Confluence

There are two ways to tell Play SQL (and other SQL plugins) how to connect to a database:

- A direct connection the plugin will contact the database directly, given a hostname, port, username and password
- A JNDI/Datasource connection the plugin will ask Confluence's middleware (the Tomcat application server) for a preconfigured database connection

Either way will work. I used a datasource, defined as the jdbc/QueriesDS section in my /opt/atlassian/confluence/conf/server.xml file:

```
<Engine name="Standalone" defaultHost="localhost" debug="0">
            <Host name="localhost" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="false"</pre>
startStopThreads="4">
                    <Context path="" docBase="../confluence" debug="0" reloadable="false" useHttpOnly="true">
                    <Resource name="jdbc/ConfluenceDS" auth="Container" type="javax.sql.DataSource"</pre>
                           username="confluence"
                           password="<REDACTED>"
                           driverClassName="org.postgresql.Driver"
                           url="jdbc:postgresql://localhost:5432/confluence"
                           maxTotal="20"
                           validationQuery="select 1"/>
                    <Resource name="jdbc/QueriesDS" auth="Container" type="javax.sql.DataSource"</pre>
                           username="confluence_reports"
                           password="confluence_reports"
                           driverClassName="org.postgresql.Driver"
                           url="jdbc:postgresql://localhost:5432/jira?currentSchema=queries"
                           maxTotal="20"
                           validationQuery="select 1"/>
                    <!-- Logging configuration for Confluence is specified in confluence/WEB-INF/classes/log4j.
properties -->
                      <!-- Uncomment this to DISABLE session serialization.
                    <Manager pathname=""/>
                    <Valve className="org.apache.catalina.valves.StuckThreadDetectionValve" threshold="60"/>
                </Context>
                <Context path="${confluence.context.path}/synchrony-proxy" docBase="../synchrony-proxy" debug="
0 "
                         reloadable="false" useHttpOnly="true">
                    <Valve className="org.apache.catalina.valves.StuckThreadDetectionValve" threshold="60"/>
                </Context>
            </Host>
         </Engine>
```

You will need to restart Confluence to pick up this change.

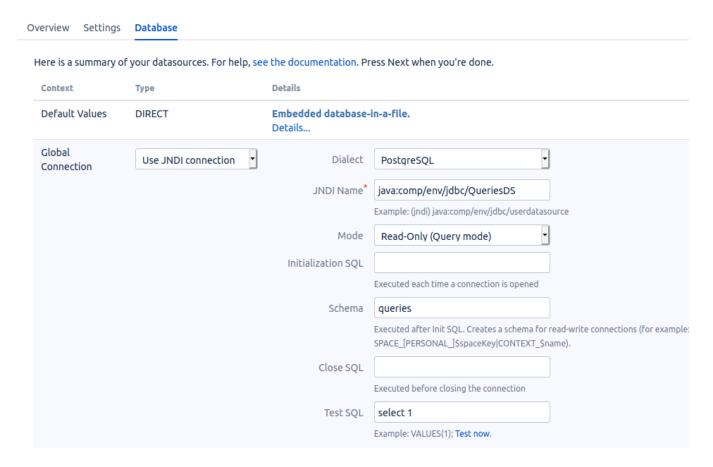
- It's more secure database credentials aren't stored as plaintext in the database or in innumerable backups.
- it lets you configure the 'QueriesDS' differently in production vs. sandbox. The database hostname for Jira might be different on the sandbox server. Rather than reconfigure PlaySQL every time you sync sandbox data, you configure 'QueriesDS' once correctly in the sandbox conf /server.xml.
- the app server can provide stats about database connection use via JMX or JavaMelody.
- It's just conceptually nicer (the inversion of control principle).

Configure PlaySQL with the Datasource

To recap, we've just been on a detour to create a read-only Postgres account, and edited Confluence's conf/server.xml file to define our QueriesDS datasource.

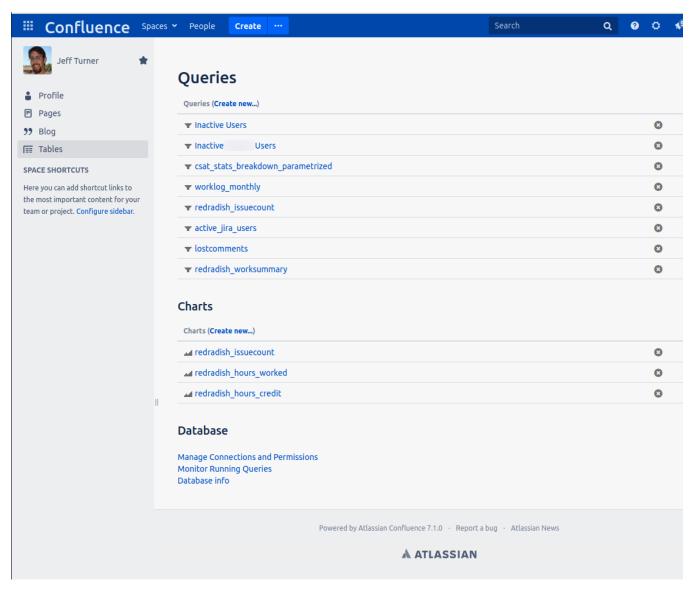
Now configure Play SQL to use the Datasource. Here I've configured QueriesDS as our default 'global connection':

Play SQL Setup



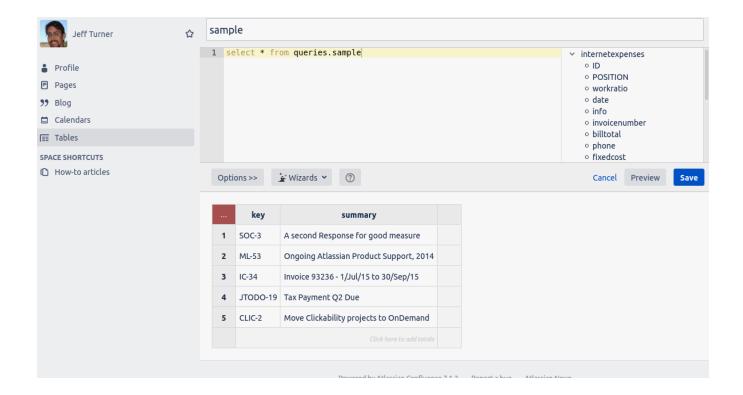
Create a test Play SQL Table

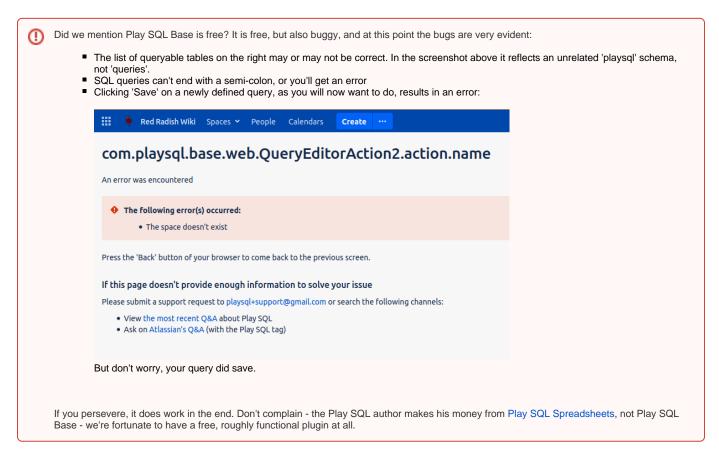
Now return to the 'Tables' tab in a space:



Under 'Queries' click 'Create new...'.

Now query your sample view and click 'Preview' to verify it works:





Create the timesheets database view

So far we've successfully queried queries.sample. We now create a $queries.worklog_monthly$ view containing our real timesheet data.

```
-- A giant table of worklog hours per day, for each day of the month, selectable by user, year and month
-- See https://www.redradishtech.com/display/KB/Creating+interactive+Jira+reports+in+Confluence+using+free+tools
create schema if not exists queries;
create or replace view queries.worklog_monthly AS
select * from (
        select user_name, email_address, year, month
        , round(sum(sum),2) AS month_total
        ,case sum("1") when 0 then 0 else round(sum("1"),2) end AS "1"
        ,case sum("2") when 0 then 0 else round(sum("2"),2) end AS "2"
        ,case sum("3") when 0 then 0 else round(sum("3"),2) end AS "3"
        ,case sum("4") when 0 then 0 else round(sum("4"),2) end AS "4"
        ,case sum("5") when 0 then 0 else round(sum("5"),2) end AS "5"
        ,case sum("6") when 0 then 0 else round(sum("6"),2) end AS "6"
        ,case sum("7") when 0 then 0 else round(sum("7"),2) end AS "7"
        ,case sum("8") when 0 then 0 else round(sum("8"),2) end AS "8"
        ,case sum("9") when 0 then 0 else round(sum("9"),2) end AS "9"
        ,case sum("10") when 0 then 0 else round(sum("10"),2) end AS "10"
        ,case sum("11") when 0 then 0 else round(sum("11"),2) end AS "11"
        ,case sum("12") when 0 then 0 else round(sum("12"),2) end AS "12"
        ,case sum("13") when 0 then 0 else round(sum("13"),2) end AS "13"
        ,case sum("14") when 0 then 0 else round(sum("14"),2) end AS "14"
        ,case sum("15") when 0 then 0 else round(sum("15"),2) end AS "15"
        ,case \mbox{sum("16")} when 0 then 0 else \mbox{round(sum("16"),2)} end AS "16"
        ,case sum("17") when 0 then 0 else round(sum("17"),2) end AS "17"
        ,case sum("18") when 0 then 0 else round(sum("18"),2) end AS "18"
        ,case sum("19") when 0 then 0 else round(sum("19"),2) end AS "19"
        ,case sum("20") when 0 then 0 else round(sum("20"),2) end AS "20"
        ,case sum("21") when 0 then 0 else round(sum("21"),2) end AS "21"
        ,case sum("22") when 0 then 0 else round(sum("22"),2) end AS "22"
        ,case sum("23") when 0 then 0 else round(sum("23"),2) end AS "23"
        ,case sum("24") when 0 then 0 else round(sum("24"),2) end AS "24"
        ,case sum("25") when 0 then 0 else round(sum("25"),2) end AS "25"
        ,case sum("26") when 0 then 0 else round(sum("26"),2) end AS "26"
        ,case sum("27") when 0 then 0 else round(sum("27"),2) end AS "27"
        ,case sum("28") when 0 then 0 else round(sum("28"),2) end AS "28"
        ,case sum("29") when 0 then 0 else round(sum("29"),2) end AS "29"
        ,case sum("30") when 0 then 0 else round(sum("30"),2) end AS "30"
        ,case sum("31") when 0 then 0 else round(sum("31"),2) end AS "31"
                select user_name, email_address, year, month, day, sum
                , case day when 1 then sum else 0 end AS "1" \,
                , case day when 2 then sum else 0 end AS "2"
                , case day when 3 then sum else 0 end AS "3"
                , case day when 4 then sum else 0 end AS "4"
                , case day when 5 then sum else 0 end AS "5"
                , case day when 6 then sum else 0 end AS "6"
                , case day when 7 then sum else 0 end AS "7"
                , case day when 8 then sum else 0 end AS "8"
                , case day when 9 then sum else 0 end AS "9"
                , case day when 10 then sum else 0 end AS "10"
                , case day when 11 then sum else 0 end AS "11"
                , case day when 12 then sum else 0 end AS "12"
                , case day when 13 then sum else 0 end AS "13"
                , case day when 14 then sum else 0 end AS "14" \,
                , case day when 15 then sum else 0 end AS "15"
                , case day when 16 then sum else 0 end AS "16"
                , case day when 17 then sum else 0 end AS "17"
                , case day when 18 then sum else 0 end AS "18"
                , case day when 19 then sum else 0 end AS "19"
                , case day when 20 then sum else 0 end AS "20"
                . case day when 21 then sum else 0 end AS "21"
                , case day when 22 then sum else 0 end AS "22"
                , case day when 23 then sum else 0 end AS "23" \,
                , case day when 24 then sum else 0 end AS "24"
                , case day when 25 then sum else 0 end AS "25"
                , case day when 26 then sum else 0 end AS "26"
                , case day when 27 then sum else 0 end AS "27"
                , case day when 28 then sum else 0 end AS "28"
```

```
, case day when 29 then sum else 0 end AS "29"
                , case day when 30 then sum else 0 end AS "30"
                , case day when 31 then sum else 0 end AS "31"
                from (
                        select
                                user_name
                                , email_address
                                , extract(year from dte) AS year
                                , extract(month from dte) AS month
                                , extract(day from dte) AS day
                                , sum(coalesce(timeworked,0))/60.0/60 AS sum
                        from
                                (select generate_series::date AS dte from generate_series('2019-01-01'::date,
now()::date, '1 day')) alldays
                                FULL OUTER JOIN cwd_user
                                ON (true)
                                INNER JOIN app_user
                                USING (lower_user_name)
                                LEFT JOIN
                                public.worklog
                                        worklog.author = app_user.user_key AND
                                        to_char(dte, 'YYYY-MM-DD') = to_char(worklog.startdate, 'YYYY-MM-DD')
                                WHERE cwd_user.active=1
                                -- and email_address ~ '(redradishtech\.com)$' -- Optionally filter to just
workloggable users here.
                        group by (user_name, email_address, year, month, day)
        ) z group by rollup((user_name, email_address), year, month)
) q
order by month_total desc
grant select on queries.worklog_monthly to jira_queries_readonly;
```

I suggest creating a directory in your Confluence app dir for SQL queries like this:

```
/opt/atlassian/jira # mkdir SQL_QUERIES
/opt/atlassian/jira # cd SQL_QUERIES/
/opt/atlassian/jira/SQL_QUERIES #
```

Then you can fetch the SQL directly using curl and run it to create the view in your database:

```
/opt/atlassian/jira/SQL_QUERIES # curl -sLOJ 'https://github.com/redradishtech/jira-interesting-sql-queries/raw
/master/worklog_monthly.sql'
/opt/atlassian/jira/SQL_QUERIES # sudo -u postgres psql redradish_jira -tAXq < worklog_monthly.sql
```

Verify that our confluence_reports user can read our new queries.worklog_monthly table:

```
# PGUSER=confluence_reports PGPASSWORD=confluence_reports PGHOST=localhost PGDATABASE=redradish_jira psql -tAc
"select count(*) from queries.worklog_monthly;"
121
```

Create a worklog_monthly Play SQL Table

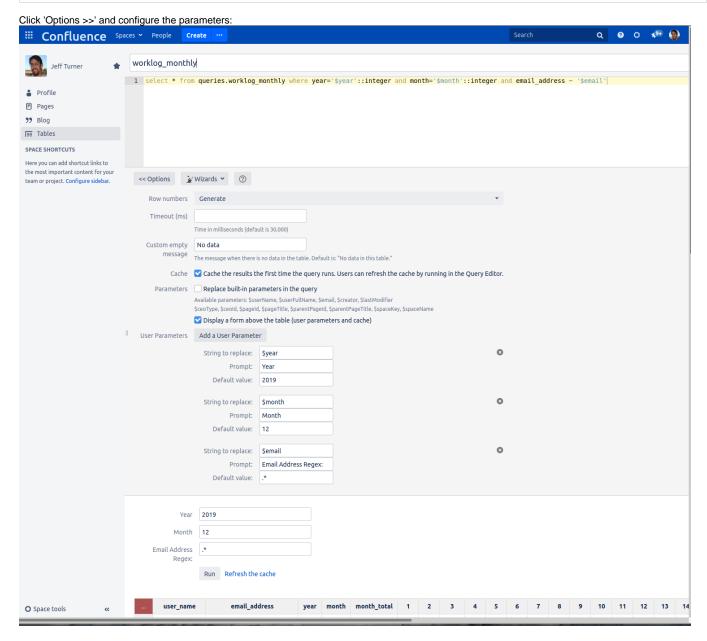
As we did earlier for queries.sample, now configure a Table in Play SQL for our queries.worklog_monthly view.

You should first enter the query:

```
select * from worklog_monthly
```

Preview it to make sure that works. If so, parametrize it:

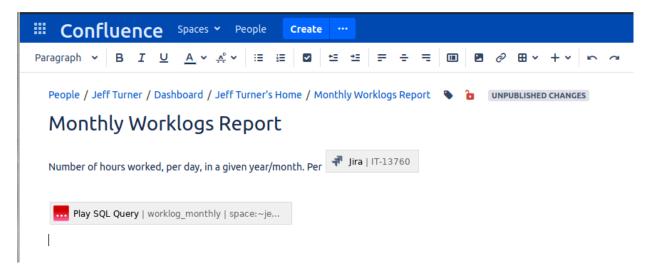
select * from queries.worklog_monthly where year='\$year'::integer and month='\$month'::integer and email_address
~ '\$email'



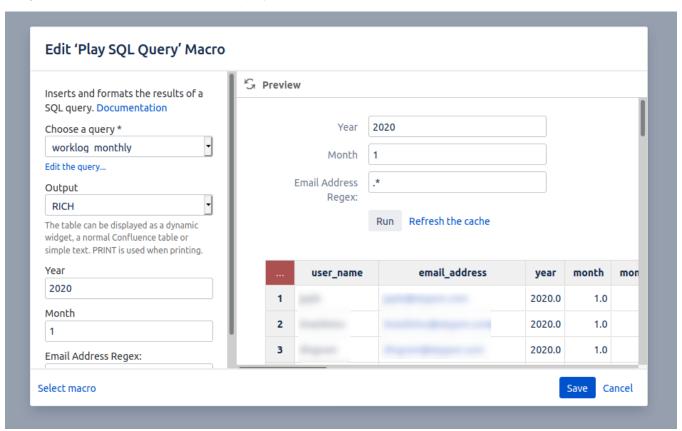
You may want to tick the 'Cache' checkbox if you have a lot of data to query.

Create a page containing the table

Our final step is to create a page in the Confluence space, containing a Play SQL Query macro:



Configure the macro to use the worklog_monthly query:



and there you have it: our final worklog report:

