Django SQL Explorer

Release 3.2.1

Chris Clark

Feb 02, 2024

CONTENTS:

CHAPTER

ONE

SQL EXPLORER

Documentation

SQL Explorer aims to make the flow of data between people fast, simple, and confusion-free. It is a Django-based application that you can add to an existing Django site, or use as a standalone business intelligence tool.

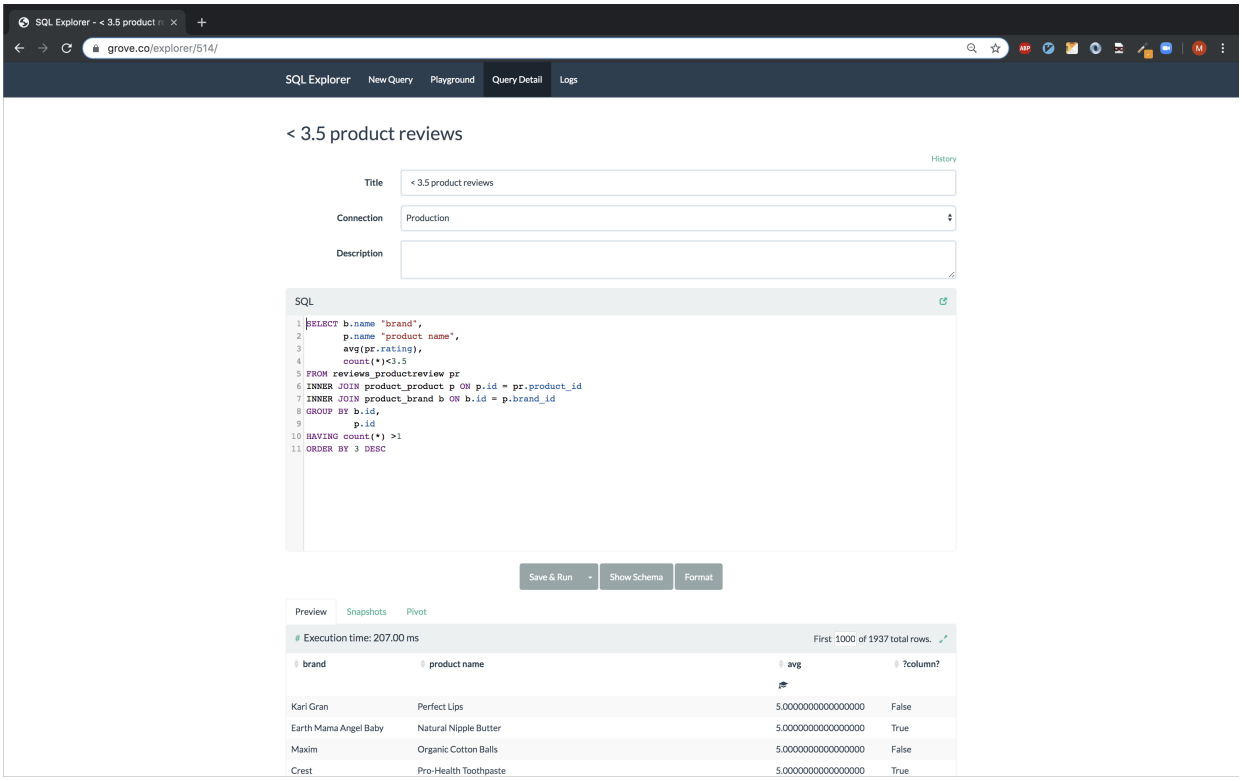
Quickly write and share SQL queries in a simple, usable SQL editor, preview the results in the browser, share links, download CSV, JSON, or Excel files (and even expose queries as API endpoints, if desired), and keep the information flowing!

Comes with support for multiple connections, to many different SQL database types, a schema explorer, query history (e.g. lightweight version control), a basic security model, in-browser pivot tables, and more.

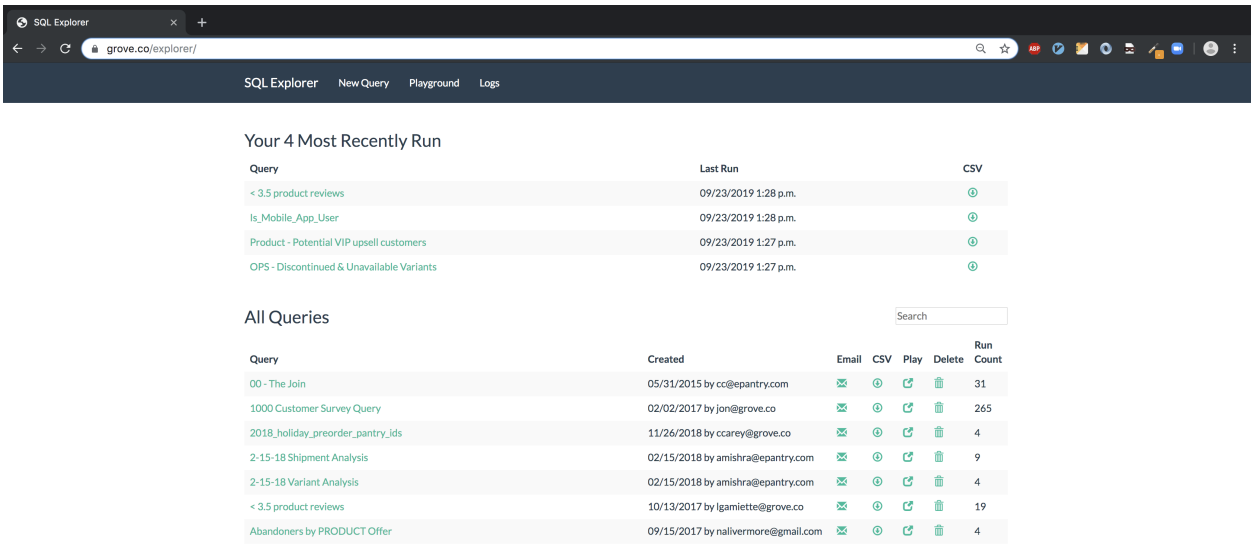
SQL Explorer values simplicity, intuitive use, unobtrusiveness, stability, and the principle of least surprise.

Sql Explorer is MIT licensed, and pull requests are welcome.

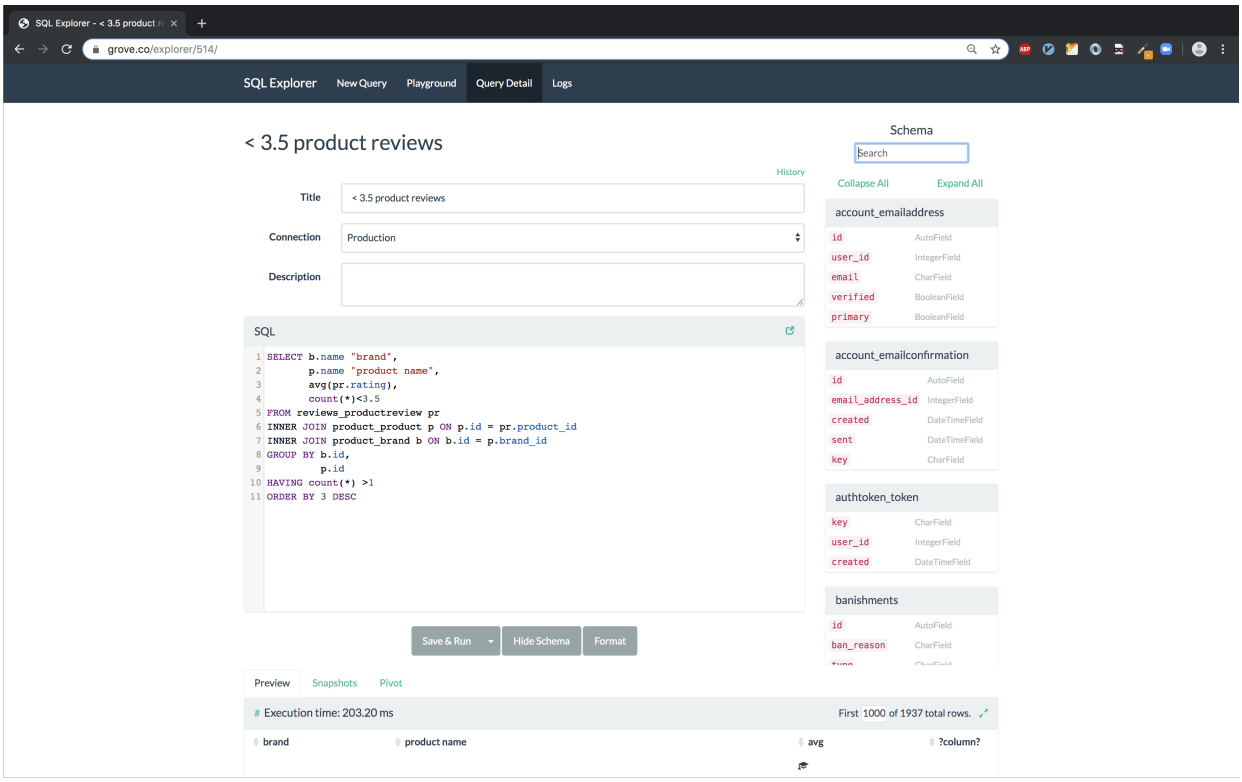
A view of a query



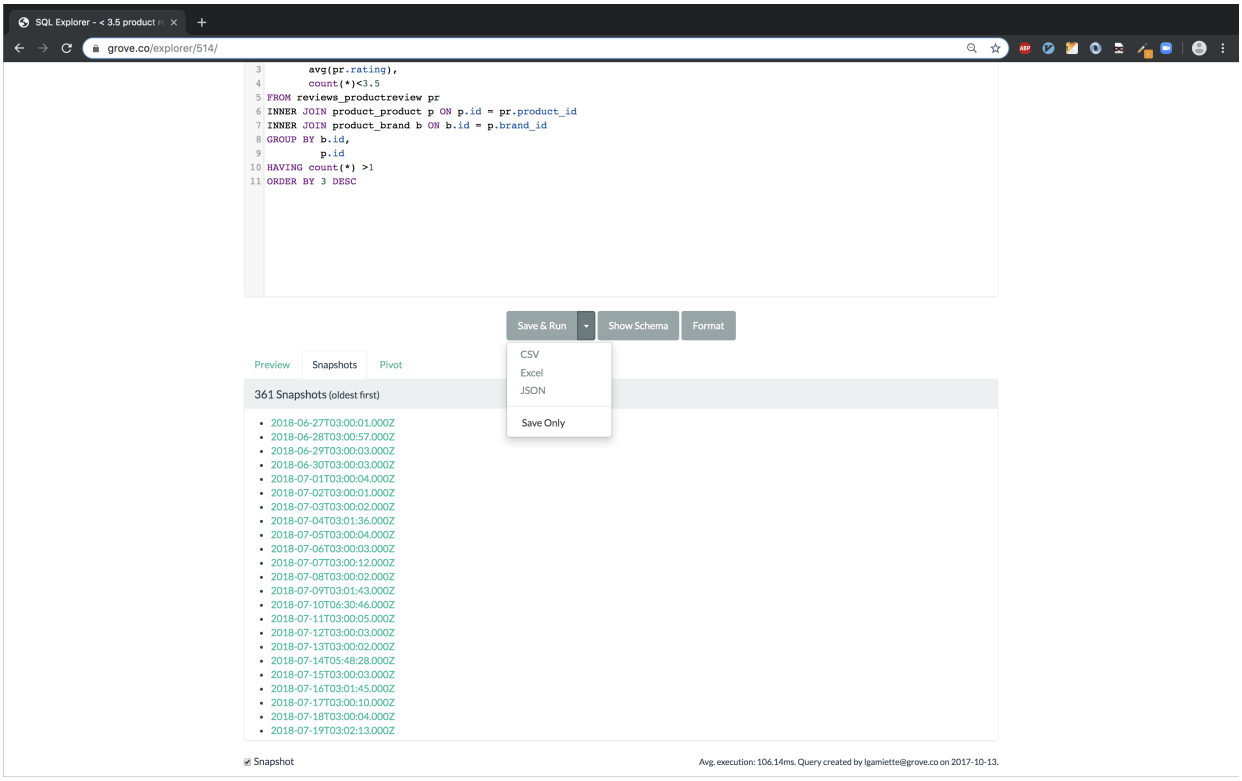
Viewing all queries



Quick access to DB schema info



Snapshot query results to S3 & download as csv



1.1 Development

Included is a test_project that you can use to kick the tires. Just create a new virtualenv, cd into `test_project` and run `start.sh` (or walk through the steps yourself) to get a test instance of the app up and running.

You can now navigate to 127.0.0.1:8000/ and begin exploring!

1.1.1 Features

Security

- Let's not kid ourselves - this tool is all about giving people access to running SQL in production. So if that makes you nervous (**and it should**) - you've been warned. Explorer makes an effort to not allow terrible things to happen, but be careful! It's recommended you setup read-only roles for each of your database connections and only use these particular connections for your queries through the `EXPLORER_CONNECTIONS` setting.
- Explorer supports two different permission checks for users of the tool. Users passing the `EXPLORER_PERMISSION_CHANGE` test can create, edit, delete, and execute queries. Users who do not pass this test but pass the `EXPLORER_PERMISSION_VIEW` test can only execute queries. Other users cannot access any part of Explorer. Both permission groups are set to is_staff by default and can be overridden in your settings file.
- Enforces a SQL blacklist so destructive queries don't get executed (delete, drop, alter, update etc). This is not a substitute for using a readonly connection – but is better than nothing for certain use cases where a readonly connection may not be available.

Easy to get started

- Built on Django's ORM, so works with Postgresql, Mysql, and Sqlite. And, between you and me, it works fine on RedShift as well.
- Small number of dependencies.
- Just want to get in and write some ad-hoc queries? Go nuts with the Playground area.

Snapshots

- Tick the 'snapshot' box on a query, and Explorer will upload a .csv snapshot of the query results to S3. Configure the snapshot frequency via a celery cron task, e.g. for daily at 1am:

```
'explorer.tasks.snapshot_queries': {
    'task': 'explorer.tasks.snapshot_queries',
    'schedule': crontab(hour=1, minute=0)
}
```

- Requires celery, obviously. Also uses boto3. All of these deps are optional and can be installed with `pip install "django-sql-explorer[snapshots]"`
- The checkbox for opting a query into a snapshot is ALL THE WAY on the bottom of the query view (underneath the results table).
- You must also have the setting `EXPLORER_TASKS_ENABLED` enabled.

Email query results

- Click the email icon in the query listing view, enter an email address, and the query results (zipped .csv) will be sent to you asynchronously. Very handy for long-running queries.

Parameterized Queries

- Use $$foo$$ in your queries and Explorer will build a UI to fill out parameters. When viewing a query like `SELECT * FROM table WHERE id=$$id$$`, Explorer will generate UI for the `id` parameter.
- Parameters are stashed in the URL, so you can share links to parameterized queries with colleagues
- Use `$$paramName:defaultValue$$` to provide default values for the parameters.
- Use `$$paramName|label$$` to add a label (e.g. "User ID") to the parameter.
- You can combine both a default and label to your parameter but you must start with the label: `$$paramName|label:defaultValue$$`.

Schema Helper

- `/explorer/schema/<connection-alias>` renders a list of your table and column names + types that you can refer to while writing queries. Apps can be excluded from this list so users aren't bogged down with tons of irrelevant tables. See settings documentation below for details.
- Autocomplete for table and column names in the Codemirror SQL editor
- This is available quickly as a sidebar helper while composing queries (see screenshot)
- Quick search for the tables you are looking for. Just start typing!
- Explorer uses Django DB introspection to generate the schema. This can sometimes be slow, as it issues a separate query for each table it introspects. Therefore, once generated, Explorer caches the schema information. There is also the option to generate the schema information asynchronously, via Celery. To enable this, make sure Celery is installed and configured, and set `EXPLORER_ENABLE_TASKS` and `EXPLORER_ASYNC_SCHEMA` to `True`.

Template Columns

- Let's say you have a query like `SELECT id, email FROM user` and you'd like to quickly drill through to the profile page for each user in the result. You can create a `template` column to do just that.
- Just set up a template column in your settings file:

```
EXPLORER_TRANSFORMS = [
    ('user', '<a href="https://yoursite.com/profile/{0}/">{0}</a>')
]
```

- And change your query to `SELECT id AS "user", email FROM user`. Explorer will match the `user` column alias to the transform and merge each cell in that column into the template string. *Cool!*
- Note you **must** set `EXPLORER_UNSAFE_RENDERING` to `True` if you want to see rendered HTML (vs string literals) in the output. This will globally un-escape query results in the preview pane. E.g. any queries that return HTML will render as HTML in the preview pane. This could have cross-site scripting implications if you don't trust the data source you are querying.

Pivot Table

- Go to the Pivot tab on query results to use the in-browser pivot functionality (provided by Pivottable JS).
- Hit the link icon on the top right to get a URL to recreate the exact pivot setup to share with colleagues.
- Download the pivot view as a CSV.

Displaying query results as charts

If the results table adheres to a certain format, the results can be displayed as a pie chart or a line chart.

To enable this feature, set `EXPLORER_CHARTS_ENABLED` setting to `True` and install the plotting libraries `matplotlib` and `seaborn` with

```
pip install "django-sql-explorer[charts]"
```

This will add the "Pie chart" and the "Line chart" tabs alongside the "Preview" and the "Pivot" tabs in the query results view.

The tabs show the respective charts if the query result table adheres to a format which the chart widget can read. Otherwise a message explaining the required format together with an example query is displayed.

Query Logs

- Explorer will save a snapshot of every query you execute so you can recover lost ad-hoc queries, and see what you've been querying.
- This also serves as cheap-and-dirty versioning of Queries, and provides the 'run count' property and average duration in milliseconds, by aggregating the logs.
- You can also quickly share playground queries by copying the link to the playground's query log record – look on the top right of the sql editor for the link icon.
- If Explorer gets a lot of use, the logs can get beefy. explorer.tasks contains the 'truncate_querylogs' task that will remove log entries older than <days> (30 days and older in the example below).

```
'explorer.tasks.truncate_querylogs': {
    'task': 'explorer.tasks.truncate_querylogs',
    'schedule': crontab(hour=1, minute=0),
    'kwargs': {'days': 30}
}
```

Multiple Connections

- Have data in more than one database? No problemo. Just set up multiple Django database connections, register them with Explorer, and you can write, save, and view queries against all of your different data sources. Compatible with any database support by Django. Note that the target database does *not* have to contain any Django schema, or be related to Django in any way. See connections.py for more documentation on multi-connection setup.

Power tips

- On the query listing page, focus gets set to a search box so you can just navigate to `/explorer` and start typing the name of your query to find it.
- Quick search also works after hitting "Show Schema" on a query view.
- Command+Enter and Ctrl+Enter will execute a query when typing in the SQL editor area.
- Hit the "Format" button to format and clean up your SQL (this is non-validating – just formatting).
- Use the Query Logs feature to share one-time queries that aren't worth creating a persistent query for. Just run your SQL in the playground, then navigate to `/logs` and share the link (e.g. `/explorer/play/?querylog_id=2428`)
- Click the 'history' link towards the top-right of a saved query to filter the logs down to changes to just that query.
- If you need to download a query as something other than csv but don't want to globally change delimiters via `settings.EXPLORER_CSV_DELIMETER`, you can use `/query/download?delim=|` to get a pipe (or whatever) delimited file. For a tab-delimited file, use `delim=tab`. Note that the file extension will remain .csv
- If a query is taking a long time to run (perhaps timing out) and you want to get in there to optimize it, go to `/query/123/?show=0`. You'll see the normal query detail page, but the query won't execute.
- Set env vars for `EXPLORER_TOKEN_AUTH_ENABLED=TRUE` and `EXPLORER_TOKEN=<SOME TOKEN>` and you have an instant data API. Just:

```
curl --header "X-API-TOKEN: <TOKEN>" https://www.your-site.com/explorer/<QUERY_ID>/
↪stream?format=csv
```

You can also pass the token with a query parameter like this:

```
curl https://www.your-site.com/explorer/<QUERY_ID>/stream?format=csv&token=<TOKEN>
```

1.1.2 Install

- Requires Python 3.10 or higher.
- Requires Django 3.2 or higher.

Set up a Django project with the following:

```
$ pip install django
$ django-admin startproject project
```

More information in the django tutorial.

Install with pip from pypi:

```
$ pip install django-sql-explorer
```

If you would also like to support downloading Excel files install with the dependency using:

```
$ pip install django-sql-explorer[xls]
```

Add to your `INSTALLED_APPS`, located in the `settings.py` file in your project folder:

```
INSTALLED_APPS = (
    ...,
    'explorer',
    ...
)
```

Configure your settings to something like:

```
EXPLORER_CONNECTIONS = { 'Default': 'readonly' }
EXPLORER_DEFAULT_CONNECTION = 'readonly'
```

The first setting lists the connections you want to allow Explorer to use. The keys of the connections dictionary are friendly names to show Explorer users, and the values are the actual database aliases used in `settings.DATABASES`. It is highly recommended to setup read-only roles in your database, add them in your project's `DATABASES` setting and use these read-only connections in the `EXPLORER_CONNECTIONS`.

Add the following to your urls.py (all Explorer URLs are restricted via the `EXPLORER_PERMISSION_VIEW` and `EXPLORER_PERMISSION_CHANGE` settings. See Settings section below for further documentation.):

```
from django.urls import path, include

urlpatterns = [
    ...
    path('explorer/', include('explorer.urls')),
```

(continues on next page)

(continued from previous page)

```
    ...
]
```

If you want to quickly use django-sql-explorer with the existing default connection **and know what you are doing** (or you are on development), you can use the following settings:

```
EXPLORER_CONNECTIONS = { 'Default': 'default' }
EXPLORER_DEFAULT_CONNECTION = 'default'
```

Run migrate to create the tables:

`python manage.py migrate`

Create a superuser:

`python manage.py createsuperuser`

And run the server:

`python manage.py runserver`

You can now browse to http://127.0.0.1:8000/explorer/ and get exploring!

The default behavior when viewing a parameterized query is to autorun the associated SQL with the default parameter values. This may perform poorly and you may want a chance for your users to review the parameters before running. If so you may add the following setting which will allow the user to view the query and adjust any parameters before hitting "Save & Run"

```
EXPLORER_AUTORUN_QUERY_WITH_PARAMS = False
```

There are a handful of features (snapshots, emailing queries) that rely on Celery and the dependencies in optional-requirements.txt. If you have Celery installed, set `EXPLORER_TASKS_ENABLED=True` in your settings.py to enable these features.

Installing From Source

If you are installing SQL Explorer from source (by cloning the repository), you may want to first look at simply running test_project/start.sh.

If you want to install it into an existing project, you can do so by following the instructions above, and additionally building the front-end dependencies.

After cloning, simply run:

```
nvm install
nvm use
npm install
npm run build
```

The front-end assets will be built and placed in the /static/ folder and collected properly by your Django installation during the *collect static* phase. Copy the /explorer directory into site-packages and you're ready to go.

And frankly, as long as you have a reasonably modern version of Node and NPM installed, you can probably skip the nvm steps.

Because the front-end assets must be built, installing SQL Explorer via pip from github is not supported. The package will be installed, but the front-end assets will be missing and will not be able to be built, as the necessary configuration files are not included when github builds the wheel for pip.

1.1.3 Settings

Here are all of the available settings with their default values.

SQL Blacklist

Disallowed words in SQL queries to prevent destructive actions.

```
EXPLORER_SQL_BLACKLIST = (
     # DML
     'COMMIT',
     'DELETE',
     'INSERT',
     'MERGE',
     'REPLACE',
     'ROLLBACK',
     'SET',
     'START',
     'UPDATE',
     'UPSERT',

     # DDL
     'ALTER',
     'CREATE',
     'DROP',
     'RENAME',
     'TRUNCATE',

     # DCL
     'GRANT',
     'REVOKE',
 )
```

Default rows

The number of rows to show by default in the preview pane.

```
EXPLORER_DEFAULT_ROWS = 1000
```

Include table prefixes

If not `None`, show schema only for tables starting with these prefixes. "Wins" if in conflict with `EXCLUDE`

```
EXPLORER_SCHEMA_INCLUDE_TABLE_PREFIXES = None  # shows all tables
```

Exclude table prefixes

Don't show schema for tables starting with these prefixes, in the schema helper.

```
EXPLORER_SCHEMA_EXCLUDE_TABLE_PREFIXES = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.admin'
)
```

Include views

Include database views

```
EXPLORER_SCHEMA_INCLUDE_VIEWS = False
```

ASYNC schema

Generate DB schema asynchronously. Requires Celery and `EXPLORER_TASKS_ENABLED`

```
EXPLORER_ASYNC_SCHEMA = False
```

Default connection

The name of the Django database connection to use. Ideally set this to a connection with read only permissions

```
EXPLORER_DEFAULT_CONNECTION = None  # Must be set for the app to work, as this is␣
↪required
```

Database connections

A dictionary of `{'Friendly Name': 'django_db_alias'}`.

```
EXPLORER_CONNECTIONS = {}  # At a minimum, should be set to something like { 'Default':
↪'readonly' } or similar. See connections.py for more documentation.
```

Permission view

Callback to check if the user is allowed to view and execute stored queries

```
EXPLORER_PERMISSION_VIEW = lambda r: r.user.is_staff
```

Permission change

Callback to check if the user is allowed to add/change/delete queries

```
EXPLORER_PERMISSION_CHANGE = lambda r: r.user.is_staff
```

Transforms

List of tuples, see *Template Columns* more info.

```
EXPLORER_TRANSFORMS = []
```

Recent query count

The number of recent queries to show at the top of the query listing.

```
EXPLORER_RECENT_QUERY_COUNT = 10
```

User query views

A dict granting view permissions on specific queries of the form

```
EXPLORER_GET_USER_QUERY_VIEWS = {userId:[queryId, ...], ...}
```

Default Value:

```
EXPLORER_GET_USER_QUERY_VIEWS = {}
```

Token Authentication

Bool indicating whether token-authenticated requests should be enabled. See *Power tips*.

```
EXPLORER_TOKEN_AUTH_ENABLED = False
```

Token

Access token for query results.

```
EXPLORER_TOKEN = "CHANGEME"
```

Celery tasks

Turn on if you want to use the `snapshot_queries` celery task, or email report functionality in `tasks.py`

```
EXPLORER_TASKS_ENABLED = False
```

S3 access key

S3 Access Key for snapshot upload

```
EXPLORER_S3_ACCESS_KEY = None
```

S3 secret key

S3 Secret Key for snapshot upload

```
EXPLORER_S3_SECRET_KEY = None
```

S3 bucket

S3 Bucket for snapshot upload

```
EXPLORER_S3_BUCKET = None
```

S3 region

S3 region. Defaults to us-east-1 if not specified.

```
EXPLORER_S3_REGION = 'us-east-1'
```

S3 endpoint url

S3 endpoint url. Normally not necessary to set. Useful to set if you are using a non-AWS S3 service or you are using a private AWS endpoint.

```
EXPLORER_S3_ENDPOINT_URL = 'https://accesspoint.vpce-abc123-abcdefgh.s3.us-east-1.vpce.
↪amazonaws.com'
```

S3 destination path

S3 destination path. Defaults to empty string. Useful to set destination folder relative to S3 bucket. Along with settings `EXPLORER_S3_ENDPOINT_URL` and `EXPLORER_S3_BUCKET` you can specify full destination path for async query results.

```
EXPLORER_S3_DESTINATION = 'explorer/query'

# if
EXPLORER_S3_ENDPOINT_URL = 'https://amazonaws.com'
EXPLORER_S3_BUCKET = 'test-bucket'
# then files will be saved to
# https://amazonaws.com/test-bucket/explorer/query/filename1.csv
# where `filename1.csv` is generated filename
```

S3 link expiration

S3 link expiration time. Defaults to 3600 seconds (1hr) if not specified. Links are generated as presigned urls for security

```
EXPLORER_S3_LINK_EXPIRATION = 3600
```

S3 signature version

The signature version when signing requests. As of `boto3` version 1.13.21 the default signature version used for generating presigned urls is still `v2`. To be able to access your s3 objects in all regions through presigned urls, explicitly set this to `s3v4`.

```
EXPLORER_S3_SIGNATURE_VERSION = 's3v4'
```

From email

The default 'from' address when using async report email functionality

```
EXPLORER_FROM_EMAIL = "django-sql-explorer@example.com"
```

Data exporters

The export buttons to use. Default includes Excel, so xlsxwriter from `requirements/optional.txt` is needed

```
EXPLORER_DATA_EXPORTERS = [
    ('csv', 'explorer.exporters.CSVExporter'),
    ('excel', 'explorer.exporters.ExcelExporter'),
    ('json', 'explorer.exporters.JSONExporter')
]
```

Unsafe rendering

Disable auto escaping for rendering values from the database. Be wary of XSS attacks if querying unknown data.

```
EXPLORER_UNSAFE_RENDERING = False
```

No permission view

Path to a view used when the user does not have permission. By default, a basic login view is provided but a dotted path to a python view can be used

```
EXPLORER_NO_PERMISSION_VIEW = 'explorer.views.auth.safe_login_view_wrapper'
```

1.1.4 Dependencies

An effort has been made to keep the number of dependencies to a minimum.

Python

Name	Version	License
sqlparse	0.4.0	BSD

- sqlparse is used for SQL formatting

Python - Optional Dependencies

Name	Version	License
celery	>=3.1,<4	BSD
django-celery	>=3.3.1	BSD
Factory Boy	>=3.1.0	MIT
xlsxwriter	>=1.3.6	BSD
boto	>=2.49	MIT

- Factory Boy is required for tests
- celery is required for the 'email' feature, and for snapshots
- boto is required for snapshots
- xlsxwriter is required for Excel export (csv still works fine without it)

JavaScript & CSS

Please see package.json for the full list of JavaScript dependencies.

Vite builds the JS and CSS bundles for SQL Explorer. The bundle for the SQL editor is fairly large at ~400kb, due primarily to CodeMirror. There is opportunity to reduce this by removing jQuery, which we hope to do in a future release.

The built front-end files are distributed in the PyPi release (and will be found by collectstatic). Instructions for building the front-end files are in install.rst.

Tests

Factory Boy is needed if you'd like to run the tests, which can you do easily:

```
python manage.py test
```

and with coverage:

```
coverage run --source='.' manage.py test
```

then:

```
coverage report
```

...97%! Huzzah!

Running Locally

Included is a test_project that you can use to kick the tires. Just create a new virtualenv, cd into `test_project` and run `start.sh` (or walk through the steps yourself) to get a test instance of the app up and running.

You can now navigate to 127.0.0.1:8000/ and begin exploring!

1.1.5 Change Log

This document records all notable changes to django-sql-explorer. This project adheres to Semantic Versioning.

`4.0b2`_ (2024-02-01)

- `#565`_: Front-end modernization. Code completion via CodeMirror 6. Bootstrap5. Vite-based build
- `#566`_: Django 5 support & tests
- `#537`_: S3 signature version support
- `#562`_: Visually show whether the last run was successful
- `#571`_: Replace isort and flake8 with Ruff (linting)

`3.2.1`_ (2023-07-13)

- #539: Test for SET PASSWORD
- #544: Fix *User* primary key reference

3.2.0 (2023-05-17)

- #533: CSRF token httponly support + s3 destination for async results

3.1.1 (2023-02-27)

- #529: Added `makemigrations --check` pre-commit hook
- #528: Add missing migration

3.1.0 (2023-02-25)

- #520: Favorite queries
- #519: Add labels to params like `$$paramName|label:defaultValue$$`
- #517: Pivot export
- #524: ci: pre-commit autoupdate
- #523: ci: ran pre-commit on all files for ci bot integration
- #522: ci: coverage update
- #521: ci: Adding django 4.2 to the test suite

3.0.1 (2022-12-16)

- #515: Fix for running without optional packages

3.0 (2022-12-15)

- Add support for Django >3.2 and drop support for <3.2
- Add support for Python 3.9, 3.10 and 3.11 and drop support for <3.8
- #496: Document breakage of "Format" button due to `CSRF_COOKIE_HTTPONLY` (#492)
- #497: Avoid execution of parameterised queries when viewing query
- #498: Change sql blacklist functionality from regex to sqlparse
- #500: Form display in popup now requires sanitize: false flag
- #501: Updated celery support
- #504: Added pre-commit hooks
- #505: Feature/more s3 providers
- #506: Check sql blacklist on execution as well as save
- #508: Conditionally import optional packages

2.5.0 (2022-10-09)

- #494: Fixes Security hole in blacklist for MySQL (#490)
- #488: docs: Fix a few typos
- #481: feat: Add pie and line chart tabs to query result preview
- #478: feat: Improved templates to make easier to customize (Fix #477)

2.4.2 (2022-08-30)

- #484: Added `DEFAULT_AUTO_FIELD` (Fix #483)
- #475: Add `SET` to blacklisted keywords

2.4.1 (2022-03-10)

- #471: Fix extra white space in description and SQL fields.

2.4.0 (2022-02-10)

- #470: Upgrade JS/CSS versions.

2.3.0 (2021-07-24)

- #450: Added Russian translations.
- #449: Translates expression for duration

2.2.0 (2021-06-14)

- Updated docs theme to furo
- #445: Added `EXPLORER_NO_PERMISSION_VIEW` setting to allow override of the "no permission" view (Fix #440)
- #444: Updated structure of the settings docs (Fix #443)

2.1.3 (2021-05-14)

- #442: `GET` params passed to the fullscreen view (Fix #433)
- #441: Include BOM in CSV export (Fix #430)

2.1.2 (2021-01-19)

- #431: Fix for hidden SQL panel on a new query

2.1.1 (2021-01-19)

Mistake in release

2.1.0 (2021-01-13)

- **BREAKING CHANGE**: `request` object now passed to `EXPLORER_PERMISSION_CHANGE` and `EXPLORER_PERMISSION_VIEW` (#417 to fix #396)

Major Changes

- #413: Static assets now served directly from the application, not CDN. (#418 also)
- #414: Better blacklist checking - Fix #371 and #412
- #415: Fix for MySQL following change for Oracle in #337

Minor Changes

- #370: Get the CSRF cookie name from django instead of a hardcoded value
- #410 and #416: Sphinx docs
- #420: Formatting change in templates
- #424: Collapsable SQL panel
- #425: Ensure a *Query* object contains SQL

2.0.0 (2020-10-09)

- **BREAKING CHANGE**: #403: Dropping support for EOL Python 2.7 and 3.5

Major Changes

- #404: Add support for Django 3.1 and drop support for (EOL) <2.2
- #408: Refactored the application, updating the URLs to use path and the views into a module

Minor Changes

- #334: Django 2.1 support
- #337: Fix Oracle query failure caused by *TextField* in a group by clause
- #345: Added (some) Chinese translation
- #366: Changes to Travis django versions
- #372: Run queries as atomic requests
- #382: Django 2.2 support
- #383: Typo in the README
- #385: Removed deprecated *render_to_response* usage
- #386: Bump minimum django version to 2.2
- #387: Django 3 support

- #390: README formatting changes
- #393: Added option to install *XlsxWriter* as an extra package
- #397: Bump patch version of django 2.2
- #406: Show some love to the README
- Fix #341: PYC files excluded from build

1.1.3 (2019-09-23)

- #347: URL-friendly parameter encoding
- #354: Updating dependency reference for Python 3 compatibility
- #357: Include database views in list of tables
- #359: Fix unicode issue when generating migration with py2 or py3
- #363: Do not use "message" attribute on exception
- #368: Update EXPLORER_SCHEMA_EXCLUDE_TABLE_PREFIXES

Minor Changes

- release checklist included in repo
- readme updated with new screenshots
- python dependencies/optional-dependencies updated to latest (six, xlsxwriter, factory-boy, sqlparse)

1.1.2 (2018-08-14)

- Fix #269
- Fix bug when deleting query
- Fix bug when invalid characters present in Excel worksheet name

Major Changes

- Django 2.0 compatibility
- Improved interface to database connection management

Minor Changes

- Documentation updates
- Load images over same protocol as originating page

1.1.1 (2017-03-21)

- Fix #288 (incorrect import)

1.1.0 (2017-03-19)

- **BREAKING CHANGE**: `EXPLORER_DATA_EXPORTERS` setting is now a list of tuples instead of a dictionary. This only affects you if you have customized this setting. This was to preserve ordering of the export buttons in the UI.
- **BREAKING CHANGE**: Values from the database are now escaped by default. Disable this behavior (enabling potential XSS attacks) with the `EXPLORER_UNSAFE_RENDERING setting`.

Major Changes

- Django 1.10 and 2.0 compatibility
- Theming & visual updates
- PDF export
- Query-param based authentication (#254)
- Schema built via SQL querying rather than Django app/model introspection. Paves the way for the tool to be pointed at any DB, not just Django DBs

Minor Changes

- Switched from TinyS3 to Boto (will switch to Boto3 in next release)
- Optionally show row numbers in results preview pane
- Full-screen view (icon on top-right of preview pane)
- Moved 'open in playground' to icon on top-right on SQL editor
- Save-only option (does not execute query)
- Show the time that the query was rendered (useful if you've had a tab open a while)

1.0.0 (2016-06-16)

- **BREAKING CHANGE**: Dropped support for Python 2.6. See `.travis.yml` for test matrix.
- **BREAKING CHANGE**: The 'export' methods have all changed. Those these weren't originally designed to be external APIs, folks have written consuming code that directly called export code.

  If you had code that looked like:

  ```
  explorer.utils.csv_report(query)
  ```

  You will now need to do something like:

  ```
  explorer.exporters.get_exporter_class('csv')(query).get_file_output()
  ```

- There is a new export system! v1 is shipping with support for CSV, JSON, and Excel (xlsx). The availablility of these can be configured via the EXPLORER_DATA_EXPORTERS setting. * *Note* that for Excel export to work, you will need to install `xlsxwriter` from `optional-requirements.txt`.
- Introduced Query History link. Find it towards the top right of a saved query.
- Front end performance improvements and library upgrades.
- Allow non-admins with permission to log into explorer.
- Added a proper test_project for an easier entry-point for contributors, or folks who want to kick the tires.
- Loads of little bugfixes.

0.9.2 (2016-02-02)

- Fixed readme issue (.1) and `setup.py` issue (.2)

0.9.1 (2016-02-01)

Major changes

- Dropped support for Django 1.6, added support for Django 1.9. See .travis.yml for test matrix.
- Dropped charted.js & visualization because it didn't work well.
- Client-side pivot tables with pivot.js. This is ridiculously cool!

Minor (but awesome!) changes

- Cmd-/ to comment/uncomment a block of SQL
- Quick 'shortcut' links to the corresponding querylog to more quickly share results. Look at the top-right of the editor. Also works for playground!
- Prompt for unsaved changes before navigating away
- Support for default parameter values via $$paramName:defaultValue$$
- Optional Celery task for truncating query logs as entries build up
- Display historical average query runtime
- Increased default number of rows from 100 to 1000
- Increased SQL editor size (5 additional visible lines)
- CSS cleanup and streamlining (making better use of foundation)
- Various bugfixes (blacklist not enforced on playground being the big one)
- Upgraded front-end libraries
- Hide Celery-based features if tasks not enabled.

0.8.0 (2015-10-21)

- Snapshots! Dump the csv results of a query to S3 on a regular schedule. More details in readme.rst under 'features'.
- Async queries + email! If you have a query that takes a long time to run, execute it in the background and Explorer will send you an email with the results when they are ready. More details in readme.rst
- Run counts! Explorer inspects the query log to see how many times a query has been executed.
- Column Statistics! Click the … on top of numeric columns in the results pane to see min, max, avg, sum, count, and missing values.
- Python 3! * Django 1.9!
- Delimiters! Export with delimiters other than commas.
- Listings respect permissions! If you've given permission to queries to non-admins, they will see only those queries on the listing page.

0.7.0 (2015-02-18)

- Added search functionality to schema view and explorer view (using list.js).
- Python 2.6 compatibility.
- Basic charts via charted (from Medium via charted.co).
- SQL formatting function.
- Token authentication to retrieve csv version of queries.
- Fixed south_migrations packaging issue.
- Refactored front-end and pulled CSS and JS into dedicated files.

0.6.0 (2014-11-05)

- Introduced Django 1.7 migrations. See readme.rst for info on how to run South migrations if you are not on Django 1.7 yet.
- Upgraded front-end libraries to latest versions.
- Added ability to grant selected users view permissions on selected queries via the `EXPLORER_USER_QUERY_VIEWS` parameter
- Example usage: `EXPLORER_USER_QUERY_VIEWS = {1: [3,4], 2:[3]}`
- This would grant user with PK 1 read-only access to query with PK=3 and PK=4 and user 2 access to query 3.
- Bugfixes
- Navigating to an explorer URL without the trailing slash now redirects to the intended page (e.g. `/logs` -> `/logs/`)
- Downloading a .csv and subsequently re-executing a query via a keyboard shortcut (cmd+enter) would re-submit the form and re-download the .csv. It now correctly just refreshes the query.
- Django 1.7 compatibility fix

0.5.1 (2014-09-02)

Bugfixes

- Created_by_user not getting saved correctly
- Content-disposition .csv issue
- Issue with queries ending in `...like '%...` clauses
- Change the way customer user model is referenced
- Pseudo-folders for queries. Use “Foo * Ba1”, “Foo * Bar2” for query names and the UI will build a little “Foo” pseudofolder for you in the query list.

0.5.0 (2014-06-06)

- Query logs! Accessible via `explorer/logs/`. You can look at previously executed queries (so you don't, for instance, lose that playground query you were working, or have to worry about mucking up a recorded query). It's quite usable now, and could be used for versioning and reverts in the future. It can be accessed at `explorer/logs/`
- Actually captures the creator of the query via a ForeignKey relation, instead of just using a Char field.
- Re-introduced type information in the schema helpers.
- Proper relative URL handling after downloading a query as CSV.
- Users with view permissions can use query parameters. There is potential for SQL injection here. I think about the permissions as being about preventing users from borking up queries, not preventing them from viewing data. You've been warned.
- Refactored params handling for extra safety in multi-threaded environments.

0.4.1 (2014-02-24)

- Renaming template blocks to prevent conflicts

0.4 (2014-02-14 *Happy Valentine's Day!*)

- Templatized columns for easy linking
- Additional security config options for splitting create vs. view permissions
- Show many-to-many relation tables in schema helper

0.3 (2014-01-25)

- Query execution time shown in query preview
- Schema helper available as a sidebar in the query views
- Better defaults for sql blacklist
- Minor UI bug fixes

0.2 (2014-01-05)

- Support for parameters
- UI Tweaks
- Test coverage

0.1.1 (2013-12-31)

Bug Fixes

- Proper SQL blacklist checks
- Downloading CSV from playground

0.1 (2013-12-29)

Initial Release