InspectOMOP Documentation

Release 0.1.5

Jonathan Badger

Jun 01, 2020

Contents

Date: Jun 01, 2020 **Version**: 0.1.5

inspecotmop is a database agnostic lightweight python 3 package that assists with abstracting data from electronic health record (EHR) databases that follow the OMOP common data model (CDM). The source is available on GitHub. Feel free to contribute or fork!

- OHDSI: Observational Health Data Sciences and Informatics
- OMOP: Observation Medical Outcomes Partnership

CHAPTER 1

Background

Interoperability is an important goal in healthcare and medical informatics research. In an ideal world, one would be able to download the source code for a published project and only have to change a handful of lines of code to repeat an experiment, but this is far from the case. Repeating experiments in medical informatics, especially when utilizing EHR data requires spending an inordinate amount of time on ETL (Extraction, Transformation, and Loading). Why? Part of the reason is that health care data can be recorded using any number of medical vocabularies, ontologies, and data formats which prohibits direct communication and necessitates an intermediate step of data mapping and normalization. This can be achieved by adopting a common data model such as the OMOP CDM from the OHDSI group and serving the data using a relational database management systems (RDBMS), but creates a new interoperability problem. RDBMS typically use structured query language (SQL) as a fundamental mechanism for abstracting data and SQL itself is not universally portable across systems. This is the problem **inspectomop** addresses. **Inspectomop** utilizes SQLAlchemy to act as a universal translator of SQL dialects and makes sharing end-to-end informatics projects possible.

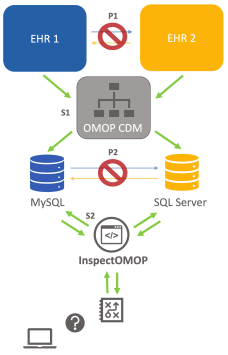

Fig. 1: **Interoperability problems and solutions: P1.** Data cannot be directly shared between EHRs. **S1.** Adopt the OMOP CDM **P2.** SQL queries are not universally portable across RDBMs. **S2.** InspectOMOP, an SQL dialect agnostic python package.

1.1 Table of Contents

1.1.1 Installation

From PyPI

inspectomop can be installed via pip from PyPI.

```
$ pip install inspectomop
```

From Source

1. Download the inspectomop project from GitHub.
2. From the root directory of inspectomop run:

```
python setup.py install
```

Dependencies

- python: version 3.0 or higher
- sqlalchemy: version 1.2.1 or higher
- pandas: version 0.22.0 or higher

1.1.2 Usage

There is a tiny SQLite database (1.4 MB) included with **inspectomop** to give first-time users a limited experimental playground and the ability to run code from the examples below.

Note: Inspectomop does **NOT** contain EHR data from real patients. The data are entirely synthetic and come from the SynPUF dataset released by Centers for Medicaid and Medicare Services (CMS).

Connecting to a database

Inspector objects are in charge of interfacing with the backend database, extracting the available OMOP CDM tables, and performing queries.

Inspectors require a single parameter, *connection_url*, for instantiation:

```
In [1]: import inspectomop as iomop

In [2]: connection_url = iomop.test.test_connection_url()

In [3]: inspector = iomop.Inspector(connection_url)
```

connection_url is a database URL defined by SQLAlchemy that describes how to connect to your database. A database URL has three main components: a dialect, driver, and URL. The dialect indicates what type of backend DB you wish to connect to. You can use any supported by SQLAlchemy (MySql, SQLite, Postgres, etc.) out-of-the-box or a dialect written by a third party. See the full list here. The driver indicates which python DBAPI library you wish to use to

run your queries. The SQLAlchemy dialects often contain a default DBAPI, so this may or may not be necessary depending on your configuration. Finally, the URL indicates where to look for the database and includes options for supplying a username and password.

```
'dialect+driver://username:password@host:port/database'
```

Note: See the SQLAlchemy docs on engine configuration for more details.

Here is an example URL for MySQL:

```
In [4]: mysql_url = 'mysql://johnny:appleseed@localhost/omop'
```

and one for SQLite:

```
In [5]: sql_url = 'sqlite:////abs/path/to/tiny_omop_test.sqlite3'
```

As you can see SQLite URLs are slightly different. They include an extra '/' and thus will have '///' for relative paths and '////' for absolute paths.

Inspecting a database

Accessing tables

The tables property of an *Inspector* contains a dictionary of associated OMOP tables that are accessible by table name.

```
In [6]: inspector.tables.keys()
Out[6]: dict_keys(['attribute_definition', 'care_site', 'cdm_source', 'cohort',
↪'cohort_attribute', 'cohort_definition', 'concept', 'concept_ancestor', 'concept_
↪class', 'concept_relationship', 'concept_synonym', 'condition_era', 'condition_
↪occurrence', 'cost', 'death', 'device_exposure', 'domain', 'dose_era', 'drug_era',
↪'drug_exposure', 'drug_strength', 'fact_relationship', 'location', 'measurement',
↪'note', 'note_nlp', 'observation', 'observation_period', 'payer_plan_period',
↪'person', 'procedure_occurrence', 'provider', 'relationship', 'source_to_concept_map
↪', 'specimen', 'visit_occurrence', 'vocabulary'])

In [7]: person = inspector.tables['person']
```

Accessing table columns

The columns in each table object are dot accessible and can be assigned to variables to construct query statements.

```
In [8]: from sqlalchemy import select

In [9]: person_id = person.person_id

In [10]: statement = select([person_id])

In [11]: print(statement)
SELECT main.person.person_id
FROM main.person
```

Complete table descriptions

You can also get a description of all columns within a table, the data types, etc.

```
In [12]: inspector.table_info('person')
Out[12]:
                          column      type  nullable  primary_key
0                      person_id   INTEGER     False         True
1             gender_concept_id   INTEGER     False        False
2                 year_of_birth   INTEGER     False        False
3                month_of_birth   INTEGER      True        False
4                  day_of_birth   INTEGER      True        False
5                birth_datetime  DATETIME      True        False
6               race_concept_id   INTEGER     False        False
7          ethnicity_concept_id   INTEGER     False        False
8                   location_id   INTEGER      True        False
9                   provider_id   INTEGER      True        False
10                 care_site_id   INTEGER      True        False
11          person_source_value      TEXT      True        False
12          gender_source_value      TEXT      True        False
13     gender_source_concept_id   INTEGER      True        False
14            race_source_value      TEXT      True        False
15       race_source_concept_id   INTEGER      True        False
16       ethnicity_source_value      TEXT      True        False
17  ethnicity_source_concept_id   INTEGER      True        False
```

Running built-in queries

A basic example

There are a variety of built in queries available in the *Queries* submodule. A typical query takes arguments for inputs (concept_ids, keywords, etc.), an *Inspector* to run the query against, and optionally a list of columns to subset from the default columns returned by the query.

```
# retrieve concepts for a list of concept_ids
In [13]: from inspectomop.queries.general import concepts_for_concept_ids

In [14]: concept_ids = [2, 3, 4, 7, 8, 10, 46287342, 46271022]

In [15]: return_columns = ['concept_name', 'concept_id']

In [16]: concepts_for_concept_ids(concept_ids, inspector, return_columns=return_
↪columns).fetchall()
Out[16]:
[(2, 'Gender'),
 (3, 'Race'),
 (4, 'Ethnicity'),
 (7, 'Metadata'),
 (8, 'Visit'),
 (10, 'Procedure'),
 (46271022, 'Chronic kidney disease'),
 (46287342, '2 ML Verapamil hydrochloride 2.5 MG/ML Injection')]
```

Note: You can get a list of columns a query returns by looking at the *return_columns* parameter in the docstring for

each query.

Specifying how results are returned

By default all queries return a *Results* object. Results objects behave like database cursors and have the expected methods such as .fetchone() and .fetchall() for fetching rows.

Note: Results objects ultimately point back to the underlying DBAPI used for interacting with the DB (pymssql for SQL Server, sqlite3 for SQLite, etc). More or less these should follow the pythong DB API spec for cursor objects. Most of this is handled by SQLAlchemy. *Results* is a subclass of `sqlalchemy.engine.ResultProxy` with additional methods for working with pandas.

Fetching examples

```
In [17]: results = concepts_for_concept_ids(concept_ids, inspector)

#get the return column names
In [18]: results.keys()
Out[18]:
['concept_id',
 'concept_name',
 'concept_code',
 'concept_class_id',
 'standard_concept',
 'vocabulary_id',
 'vocabulary_name']

#get one row
In [19]: results.fetchone()
Out[19]: (2, 'Gender', 'OMOP generated', 'Domain', '', 'Domain', 'OMOP Domain')

#get many rows
In [20]: two_results = results.fetchmany(2)

In [21]: len(two_results)
Out[21]: 2

#iterating over rows
In [22]: for row in results:
   ....:     print(row[:2])
   ....:
(7, 'Metadata')
(8, 'Visit')
(10, 'Procedure')
(46271022, 'Chronic kidney disease')
(46287342, '2 ML Verapamil hydrochloride 2.5 MG/ML Injection')
```

Results as pandas DataFrames

Results objects also have two handy methods, .as_pandas() and .as_pandas_chunks(), for returning results as pandas DataFrames.

```
#return the results as as a dataframe
In [23]: results = concepts_for_concept_ids(concept_ids, inspector).as_pandas()

In [24]: results[['concept_name','vocabulary_id']]
Out[24]:
                                        concept_name vocabulary_id
0                                             Gender        Domain
1                                               Race        Domain
2                                          Ethnicity        Domain
3                                           Metadata        Domain
4                                              Visit        Domain
5                                          Procedure        Domain
6                               Chronic kidney disease        SNOMED
7  2 ML Verapamil hydrochloride 2.5 MG/ML Injection        RxNorm

# return the results in chunks
In [25]: chunksize = 3

In [26]: results = concepts_for_concept_ids(concept_ids, inspector).as_pandas_
↪chunks(chunksize)

In [27]: for num, chunk in enumerate(results):
   ....:     print('chunk {}'.format(num + 1))
   ....:     print(chunk['concept_name'])
   ....:
chunk 1
0       Gender
1         Race
2    Ethnicity
Name: concept_name, dtype: object
chunk 2
0     Metadata
1        Visit
2    Procedure
Name: concept_name, dtype: object
chunk 3
0                              Chronic kidney disease
1    2 ML Verapamil hydrochloride 2.5 MG/ML Injection
Name: concept_name, dtype: object
```

Creating custom queries

From SQLAlchemy SQL Expressions

Statements built out of constructs from SQLAlchemy's *SQL Expression API* make queries backend-neutral paving the way for sharable code that can be used in a plug-and-play fashion. While there is no guarantee that *every* query will work with *every* backend, most of the basic selects, joins, etc should run without issue.

SQLAlchemy is extremely powerful, but like any software package, has a bit of a learning curve. It is highly recommended that users read the SQL Expression Language Tutorial and note the warning below.

Below are a few simple examples of using SQLAlchemy expression language constructs for running queries on the OMOP CDM.

Warning: Tables from Inspector.tables are actually mapped to ORM objects. These are *NOT* the same as *Table* objects from the SQLAlchemy Core API, although they can be used in nearly identical fashion in SQL Expressions with the following caveat about accessing table columns:

```
In [28]: from sqlalchemy import alias

In [29]: p = inspector.tables['person']

In [30]: p_alias = alias(inspector.tables['person'], 'p_alias')

# p is an automapped ORM object with dot accessible columns
In [31]: p
Out[31]: sqlalchemy.ext.automap.person

In [32]: p.person_id
Out[32]: <sqlalchemy.orm.attributes.InstrumentedAttribute at 0x7fe0f617f678>

# p_alias is an Alias object.
# Columns must be accessed using .c.column
In [33]: p_alias
Out[33]: <sqlalchemy.sql.selectable.Alias at 0x7fe0f61bbfd0; p_alias>

In [34]: p_alias.c.person_id
Out[34]: Column('person_id', INTEGER(), table=<p_alias>, primary_key=True,␣
↪nullable=False)

# and so this fails
In [35]: p_alias.person_id
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-35-a8a43d4647a5> in <module>
----> 1 p_alias.person_id

AttributeError: 'Alias' object has no attribute 'person_id'
```

Explanation: Using a portion of the SQLAlchemy ORM to infer table structure was a conscious design decision. Although it makes for a bit of confusion when constructing queries with SQL expressions users that work in an interactive development environment (iPython, Jupyter Notebooks, etc.) get the benefit of dot accessible column properties. In addition, automapping alleviates compatibility issues that would inevitably arise with hard-coded table structures on future versions of the OMOP CDM.

Select all of the conditions for person 1:

```
In [36]: from sqlalchemy import select, and_

In [37]: c = inspector.tables['concept']

In [38]: co = inspector.tables['condition_occurrence']

In [39]: person_id = 1

In [40]: statement = select([co.condition_start_date, co.condition_concept_id, c.
↪concept_name]).\
   ....:             where(and_(\
   ....:                 co.person_id == person_id,\
   ....:                 co.condition_concept_id == c.concept_id))
   ....:
```

(continues on next page)

(continued from previous page)

```
In [41]: print(statement)
SELECT main.condition_occurrence.condition_start_date, main.condition_occurrence.
↪condition_concept_id, main.concept.concept_name
FROM main.condition_occurrence, main.concept
WHERE main.condition_occurrence.person_id = :person_id_1 AND main.condition_
↪occurrence.condition_concept_id = main.concept.concept_id

In [42]: inspector.execute(statement).as_pandas()
Out[42]:
   condition_start_date  ...                                       concept_name
0            2010-03-12  ...                                       Osteoporosis
1            2009-07-25  ...                                           Backache
2            2009-07-25  ...                                      Low back pain
3            2010-08-17  ...                                        Neck sprain
4            2010-11-05  ...                Subchronic catatonic schizophrenia
5            2009-10-14  ...                                       Hypocalcemia
6            2010-03-12  ...                          Congestive heart failure
7            2010-11-05  ...                                       Schizophrenia
8            2010-03-12  ...              Antiallergenic drug adverse reaction
9            2010-04-01  ...                                    Bipolar disorder
10           2010-03-12  ...                          Pure hypercholesterolemia
11           2009-10-14  ...                                Postoperative pain
12           2010-04-01  ...  Bipolar I disorder, single manic episode, in f...
13           2009-07-25  ...                                Menopausal syndrome
14           2009-07-25  ...                               Thoracic radiculitis
15           2010-03-12  ...                                  Retention of urine

[16 rows x 3 columns]
```

Count the number of inpatient and outpatient visits for each person broken down by visit type and sorted by person_id:

```
In [43]: from sqlalchemy import join, func

In [44]: vo = inspector.tables['visit_occurrence']

In [45]: j = join(vo, c, vo.visit_concept_id == c.concept_id)

In [46]: j2 = join(j, p, vo.person_id == p.person_id)

In [47]: visit_types = ['Inpatient Visit','Outpatient Visit']

In [48]: statement = select([p.person_id, func.count(vo.visit_occurrence_id).label(
↪'num_visits'), c.concept_name.label('visit_type')]).\
   ....:             select_from(j2).\
   ....:             where(c.concept_name.in_(visit_types)).\
   ....:             group_by(p.person_id, c.concept_name).\
   ....:             order_by(p.person_id)
   ....:

In [49]: inspector.execute(statement).as_pandas()
Out[49]:
   person_id  num_visits        visit_type
0          1           1   Inpatient Visit
1          1           1  Outpatient Visit
2          2           4   Inpatient Visit
3          2           2  Outpatient Visit
```

(continues on next page)

(continued from previous page)

```
4           3           1  Outpatient Visit
5           5           4  Outpatient Visit
6           7          18  Outpatient Visit
7           8          11  Outpatient Visit
8           9           2  Outpatient Visit
```

From Strings

You *can* execute unaltered SQL strings directly, but remember to always used parametrized code for shared/production projects.

Warning: Only use strings for rapid prototyping and in-house projects! Executing strings directly breaks backend compatibility and can potentially lead to SQL injection attacks!

Example:

```
In [50]: inspector.execute('select person_id from person').as_pandas()
Out[50]:
   person_id
0          1
1          2
2          3
3          4
4          5
5          6
6          7
7          8
8          9
9         16
```

Sharing custom queries as functions

Custom queries that may prove useful to the OMOP CDM community can easily be shared by wrapping them in a function and following a standard recipe. View the source code on GitHub to get a better feel of how to construct queries and contribute (via pull request or posting your function in issues).

In general, consider the following:

- appropriately named query functions should begin with the data you intend to return and end with the data/parameters you expect as input. E.g. *concepts_for_concept_ids*
- the return value for a query should *always* be a `Results` object. This provides consistency and gives the end-user control over how to process the results.
- write a docstring following the numpydoc docstring guide to accompany your code.

Prototype:

```
def output_for_input(inputs, inspector, return_columns=None):
    """
    Short description.
```

(continues on next page)

(continued from previous page)

```
    Longer explanation.

    Parameters
    ----------
    inputs : type
        description of inputs
    inspector : inspectomop.inspector.Inspector
    return_columns : list of str, optional
        - optional subset of columns to return from the query
        - columns : ['col_name_1', 'col_name_2']

    Returns
    -------
    results : inspectomop.results.Results
        a cursor-like object with methods such as fetchone(), fetchmany() etc.

    Notes
    -----
    Optional

    """
    columns = [] # specify return columns

    if return_columns: # filter based on end-user selection
        columns = [col for col in columns if col.name in return_columns]

    statement = select([columns]).where(inputs == criteria)

    return inspector.execute(statement)
```

1.1.3 API Reference

Inspector

inspectomop.Inspector

Constructor

Inspector(connection_url)	Creates an Inspector object which can be used to run OMOP data queries

inspectomop.inspector.Inspector

class inspectomop.inspector.**Inspector**(*connection_url*)

Creates an Inspector object which can be used to run OMOP data queries

Parameters **connection_url** (*string*) – A connection url of form 'dialect+driver://username:password@host:port/database'. The driver can be any currently supported by sqlalchemy (sqlite, mysql, postgresql, etc.).

connection_url

Type str

Notes

SQLite DBs require an additional '/' as in 'sqlite:///foo.db' for a relative path and 'sqlite:////abs/path/to/foo.db' for an absolute path.

See http://docs.sqlalchemy.org/en/latest/core/engines.html for more information about connection URLs and supported dialects.

Examples

```
>>> import inspectomop as iomop
>>> connection_url = 'sqlite:///:memory:'
>>> iomop.Inspector(connection_url)
```

__init__(*connection_url*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(connection_url)	Initialize self.
attach_sqlite_db(db_file, schema_name)	For SQLite backends, attaches an additional sqlite database file.
execute(statement)	Executes an SQL query on the OMOP CDM.
table_info(table_name)	Return a Pandas DataFrame describing the fields and properties of a table.

Attributes

clinical_tables	A dictionary containing all of the `Clinical` OMOP CDM tables in the connected database.
connection_url	password@host:port/database' used to specify the dialect, location, etc.
derived_elements_tables	A dictionary containing all of the `Derived Elements` OMOP CDM tables in the connected database.
engine	A convenience hook to the underlying sqlalchemy engine.
health_economics_tables	A dictionary containing all of the `Health Economics` OMOP CDM tables in the connected database.
health_system_tables	A dictionary containing all of the `Health System` OMOP CDM tables in the connected database.
metadata_tables	A dictionary containing all of the `MetaData` OMOP CDM tables in the connected database.
tables	A dictionary containing all OMOP CDM tables in the connected database.
vocabularies_tables	A dictionary containing all of the `Vocabularies` OMOP CDM tables in the connected database.

Attributes

Inspector.connection_url	password@host:port/database' used to specify the dialect, location, etc.
Inspector.engine	A convenience hook to the underlying sqlalchemy engine.
Inspector.tables	A dictionary containing all OMOP CDM tables in the connected database.
Inspector.vocabularies_tables	A dictionary containing all of the `Vocabularies` OMOP CDM tables in the connected database.
Inspector.metadata_tables	A dictionary containing all of the `MetaData` OMOP CDM tables in the connected database.
Inspector.clinical_tables	A dictionary containing all of the `Clinical` OMOP CDM tables in the connected database.
Inspector.health_system_tables	A dictionary containing all of the `Health System` OMOP CDM tables in the connected database.
Inspector.health_economics_tables	A dictionary containing all of the `Health Economics` OMOP CDM tables in the connected database.
Inspector.derived_elements_tables	A dictionary containing all of the `Derived Elements` OMOP CDM tables in the connected database.

inspectomop.inspector.Inspector.connection_url

`Inspector.`**`connection_url`**

password@host:port/database' used to specify the dialect, location, etc. of the database.

Type A URL of the form 'dialect+driver

Type //username

inspectomop.inspector.Inspector.engine

`Inspector.`**`engine`**

A convenience hook to the underlying sqlalchemy engine.

Use Inspector.execute for submitting queries.

inspectomop.inspector.Inspector.tables

`Inspector.`**`tables`**

A dictionary containing all OMOP CDM tables in the connected database.

inspectomop.inspector.Inspector.vocabularies_tables

`Inspector.`**`vocabularies_tables`**

A dictionary containing all of the `Vocabularies` OMOP CDM tables in the connected database.

inspectomop.inspector.Inspector.metadata_tables

Inspector.**metadata_tables**
A dictionary containing all of the MetaData OMOP CDM tables in the connected database.

inspectomop.inspector.Inspector.clinical_tables

Inspector.**clinical_tables**
A dictionary containing all of the Clinical OMOP CDM tables in the connected database.

inspectomop.inspector.Inspector.health_system_tables

Inspector.**health_system_tables**
A dictionary containing all of the Health System OMOP CDM tables in the connected database.

inspectomop.inspector.Inspector.health_economics_tables

Inspector.**health_economics_tables**
A dictionary containing all of the Health Economics OMOP CDM tables in the connected database.

inspectomop.inspector.Inspector.derived_elements_tables

Inspector.**derived_elements_tables**
A dictionary containing all of the Derived Elements OMOP CDM tables in the connected database.

Methods

Inspector.attach_sqlite_db(db_file, schema_name)	For SQLite backends, attaches an additional sqlite database file.
Inspector.execute(statement)	Executes an SQL query on the OMOP CDM.
Inspector.table_info(table_name)	Return a Pandas DataFrame describing the fields and properties of a table.

inspectomop.inspector.Inspector.attach_sqlite_db

Inspector.**attach_sqlite_db**(*db_file*, *schema_name*)
For SQLite backends, attaches an additional sqlite database file. Uses 'ATTACH DATABSE db_file AS schema_name'

Parameters

- **db_file** (*String*) – A string giving a path to a database file. Ex. 'databases/my_db_to_attach.db'
- **schema_name** (*String*) – The name to associate with the attached schema

inspectomop.inspector.Inspector.execute

Inspector.**execute**(*statement*)

Executes an SQL query on the OMOP CDM.

Parameters **statement** (*sqlalchemy object or string*) –

sqlalchemy objects - statements can be created using sqlalchemy objects such as select, insert, etc. and the under
e.g. select([concept]).where(concept.concept_id==0)

strings - can be a string containing an SQL statemment such as e.g. 'SELECT concept_name from concept where concept_id = 0'

Returns **results** – The results object is a subclass of SQLAlchemy results_proxy with extra methods for retrieving the results as pandas DataFrames. Traditional methods conforming to the python DB connection spec work as well e.g. fetchone, fetchmany, fetchall

Return type inspectomop.Results

Notes

*** Use of raw SQL strings is not recommended as they bypass the dialect translation and security provided by using SQLAlchemy ***

See also:

`inpsectomop.Results()`, `inspectomop.queries()`

inspectomop.inspector.Inspector.table_info

Inspector.**table_info**(*table_name*)

Return a Pandas DataFrame describing the fields and properties of a table.

Parameters **table_name** (*String*) –

Returns **table_info** – columns are 'column', 'type', 'nullable', 'primary_key'

Return type Pandas.DataFrame

Results

inspectomop.Results

Constructor

Results(results_proxy)	A cursor-like object with methods such as *fetchone*, *fetchmany* etc.

inspectomop.results.Results

class inspectomop.results.**Results**(*results_proxy*)

A cursor-like object with methods such as *fetchone*, *fetchmany* etc. that can be used to retrieve rows of results from query execution.

A subclass of sqlalchemy.engine.result.ResultProxy that adds additional methods for retreving query results as Pandas DataFrames.

See also:

Results.as_pandas, *Results.as_pandas_chunks*

__init__ (*results_proxy*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(results_proxy)	Initialize self.
as_pandas()	Return all rows from a *results* object as a pandas DataFrame
as_pandas_chunks(chunksize)	Yields a pandas DataFrame with n_rows = chunksize
close()	Close this ResultProxy.
fetchall()	Fetch all rows, just like DB-API `cursor.fetchall()`.
fetchmany([size])	Fetch many rows, just like DB-API `cursor.fetchmany(size=cursor.arraysize)`.
fetchone()	Fetch one row, just like DB-API `cursor.fetchone()`.
first()	Fetch the first row and then close the result set unconditionally.
keys()	Return the list of string keys that would represented by each `RowProxy`.
last_inserted_params()	Return the collection of inserted parameters from this execution.
last_updated_params()	Return the collection of updated parameters from this execution.
lastrow_has_defaults()	Return `lastrow_has_defaults()` from the underlying `ExecutionContext`.
next()	Implement the Python next() protocol.
postfetch_cols()	Return `postfetch_cols()` from the underlying `ExecutionContext`.
prefetch_cols()	Return `prefetch_cols()` from the underlying `ExecutionContext`.
process_rows(rows)	
scalar()	Fetch the first column of the first row, and close the result set.
supports_sane_multi_rowcount()	Return `supports_sane_multi_rowcount` from the dialect.
supports_sane_rowcount()	Return `supports_sane_rowcount` from the dialect.

Attributes

closed	
inserted_primary_key	Return the primary key for the row just inserted.

Continued on next page

Table 8 – continued from previous page

is_insert	True if this _engine.ResultProxy is the result of a executing an expression language compiled _expression.insert() construct.
lastrowid	return the 'lastrowid' accessor on the DBAPI cursor.
out_parameters	
returned_defaults	Return the values of default columns that were fetched using the ValuesBase.return_defaults() feature.
returns_rows	True if this _engine.ResultProxy returns rows.
rowcount	Return the 'rowcount' for this result.

Warning: Although a public constructor exists, *Results* objects are meant to be instantiated indirectly from calls to *inspector.execute()*

Methods

Results.as_pandas()	Return all rows from a *results* object as a pandas DataFrame
Results.as_pandas_chunks(chunksize)	Yields a pandas DataFrame with n_rows = chunksize
Results.fetchone()	Fetch one row, just like DB-API cursor.fetchone().
Results.fetchmany([size])	Fetch many rows, just like DB-API cursor.fetchmany(size=cursor.arraysize).
Results.fetchall()	Fetch all rows, just like DB-API cursor.fetchall().

inspectomop.results.Results.as_pandas

Results.**as_pandas**()

Return all rows from a *results* object as a pandas DataFrame

Returns **results**

Return type Pandas.DataFrame

See also:

as_pandas_chunks()

inspectomop.results.Results.as_pandas_chunks

Results.**as_pandas_chunks**(*chunksize*)

Yields a pandas DataFrame with n_rows = chunksize

Parameters **chunksize** (*int*) – number of rows to return in each chunk

See also:

as_pandas()

inspectomop.results.Results.fetchone

`Results.`**`fetchone`**`()`

Fetch one row, just like DB-API `cursor.fetchone()`.

After all rows have been exhausted, the underlying DBAPI cursor resource is released, and the object may be safely discarded.

Calls to `_engine.ResultProxy.fetchone()` after all rows have been exhausted will return `None`. After the `_engine.ResultProxy.close()` method is called, the method will raise `ResourceClosedError`.

Returns a `RowProxy` object, or None if no rows remain

inspectomop.results.Results.fetchmany

`Results.`**`fetchmany`**`(`*size=None*`)`

Fetch many rows, just like DB-API `cursor.fetchmany(size=cursor.arraysize)`.

After all rows have been exhausted, the underlying DBAPI cursor resource is released, and the object may be safely discarded.

Calls to `_engine.ResultProxy.fetchmany()` after all rows have been exhausted will return an empty list. After the `_engine.ResultProxy.close()` method is called, the method will raise `ResourceClosedError`.

Returns a list of `RowProxy` objects

inspectomop.results.Results.fetchall

`Results.`**`fetchall`**`()`

Fetch all rows, just like DB-API `cursor.fetchall()`.

After all rows have been exhausted, the underlying DBAPI cursor resource is released, and the object may be safely discarded.

Subsequent calls to `_engine.ResultProxy.fetchall()` will return an empty list. After the `_engine.ResultProxy.close()` method is called, the method will raise `ResourceClosedError`.

Returns a list of `RowProxy` objects

Queries

inspectomop.queries

Care Site

inspectomop.queries.care_site

facility_counts_by_type(inspector[, ...])	Returns facility counts by type in the OMOP CDM i.e.
patient_counts_by_care_site_type(inspector)	Returns pateints counts by facility type.

inspectomop.queries.care_site.facility_counts_by_type

inspectomop.queries.care_site.**facility_counts_by_type**(*inspector*, *return_columns=None*)

Returns facility counts by type in the OMOP CDM i.e. # Inpatient Hospitals, Offices, etc.

Parameters

- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['place_of_service', 'place_of_service_concept_id', 'facility_count']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

Notes

Original SQL

CS01: Care site place of service counts:

```
SELECT
    cs.place_of_service_concept_id,
    count(1) places_of_service_count
FROM care_site cs
GROUP BY
    cs.place_of_service_concept_id
ORDER BY 1;
```

inspectomop.queries.care_site.patient_counts_by_care_site_type

inspectomop.queries.care_site.**patient_counts_by_care_site_type**(*inspector*, *return_columns=None*)

Returns pateints counts by facility type.

Parameters

- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['place_of_service', 'place_of_service_concept_id', 'patient_count']

Returns **results**

Return type *inspectomop.results.Results*

Notes

Original SQL

CS02: Patient count per care site place of service:

```
SELECT
    cs.place_of_service_concept_id,
    count(1) num_patients
FROM
    care_site cs,
    person p
WHERE
    p.care_site_id = cs.care_site_id
GROUP BY
    cs.place_of_service_concept_id
ORDER BY 1;
```

Condition

inspectomop.queries.condition

anatomical_site_by_keyword(keyword, inspector)	Retrieves anitomical site concepts given a keyword.
condition_concept_for_concept_id(concept_id, ...)	Retrieves the condition concept for a condition_concept_id.
condition_concepts_for_keyword(keyword, ...)	Retrieves standard concepts for a condition/keyword.
condition_concepts_for_source_codes(..., [, ...])	Retrieves standard condition concepts for source codes.
condition_concepts_occurring_at_anatomical_site_concept_id(...)	Retrieves condition concepts that occur at a given anatomical site.
conditions_caused_by_pathogen_or_causative_agent_concept_id(...)	Retreives all conditions caused by a pathogen or other causative agent concept_id.
disease_causing_agents_for_keyword(keyword, ...)	Retrieves disease causing agents by keyword.
source_codes_for_concept_ids(concept_ids, ...)	Retreives source condition concepts for OMOP concept_ids.
pathogen_concept_for_keyword(keyword, inspector)	Retrieves pathogen concepts based on a keyword with 'Organsim' as the concept_class_id.
place_of_service_counts_for_condition_concept_id(...)	Provides counts of conditions stratified by place_of_service (Office, Inpatient Hospital, etc.)

inspectomop.queries.condition.anatomical_site_by_keyword

inspectomop.queries.condition.**anatomical_site_by_keyword**(*keyword*, *inspector*, *return_columns=None*)

Retrieves anitomical site concepts given a keyword. Results of this query are useful for *condition_concepts_occurring_at_anatomical_site_concept_id*

Parameters

- **keyword** (*str*) – search string. ex 'Epiglottis'
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query

– columns : ['concept_id', 'concept_name', 'concept_code', 'concept_class_id', 'standard_concept', 'vocabulary_id', 'vocabulary_name']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

See also:

condition_concepts_occurring_at_anatomical_site_concept_id()

Notes

Original SQL

C10: Find an anatomical site by keyword:

```
SELECT
    C.concept_id Anatomical_site_ID,
    C.concept_name Anatomical_site_Name,
    C.concept_code Anatomical_site_Code,
    C.concept_class_id Anatomical_site_Class,
    C.standard_concept Anatomical_standard_concept,
    C.vocabulary_id Anatomical_site_Vocab_ID,
    V.vocabulary_name Anatomical_site_Vocab_Name
FROM
    concept C,
    vocabulary V
WHERE
    LOWER(C.concept_class_id) = 'body structure' AND
    LOWER(C.concept_name) like '%epiglottis%' AND
    C.vocabulary_id = V.vocabulary_id AND
    sysdate BETWEEN C.valid_start_date AND C.valid_end_date;
```

inspectomop.queries.condition.condition_concept_for_concept_id

inspectomop.queries.condition.**condition_concept_for_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Retrieves the condition concept for a condition_concept_id.

Parameters

- **concept_id** (*int*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['concept_id', 'concept_name', 'concept_code', 'concept_class_id', 'vocabulary_id', 'vocabulary_name']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

See also:

inspectomop.queries.general.concepts_for_concept_ids()

Notes

Original SQL

C01: Find condition by concept ID:

```
SELECT
    C.concept_id Condition_concept_id,
    C.concept_name Condition_concept_name,
    C.concept_code Condition_concept_code,
    C.concept_class_id Condition_concept_class,
    C.vocabulary_id Condition_concept_vocab_ID,
    V.vocabulary_name Condition_concept_vocab_name,
    CASE C.vocabulary_id
        WHEN 'SNOMED' THEN CASE lower(C.concept_class_id)
        WHEN 'clinical finding' THEN 'Yes' ELSE 'No' END
        WHEN 'MedDRA' THEN 'Yes'
        ELSE 'No'
    END Is_Disease_Concept_flag
FROM
    concept C,
    vocabulary V
WHERE
    C.concept_id = 192671 AND
    C.vocabulary_id = V.vocabulary_id AND
    sysdate BETWEEN valid_start_date AND valid_end_date;
```

inspectomop.queries.condition.condition_concepts_for_keyword

inspectomop.queries.condition.**condition_concepts_for_keyword**(*keyword*, *inspector*, *return_columns=None*)

Retrieves standard concepts for a condition/keyword.

Parameters

- **keyword** (*str*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘concept_id’, ‘concept_name’, ‘concept_code’, ‘concept_class_id’, ‘vocabulary_id’]

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

Notes

Original SQL

C02: Find a condition by keyword:

```
SELECT
    T.Entity_Concept_Id,
    T.Entity_Name,
    T.Entity_Code,
    T.Entity_Type,
    T.Entity_concept_class,
    T.Entity_vocabulary_id,
    T.Entity_vocabulary_name
FROM (
    SELECT
        C.concept_id Entity_Concept_Id,
        C.concept_name Entity_Name,
        C.CONCEPT_CODE Entity_Code,
        'Concept' Entity_Type,
        C.concept_class_id Entity_concept_class,
        C.vocabulary_id Entity_vocabulary_id,
        V.vocabulary_name Entity_vocabulary_name,
        NULL Entity_Mapping_Type,
        C.valid_start_date,
        C.valid_end_date
    FROM concept C
    JOIN vocabulary V ON C.vocabulary_id = V.vocabulary_id
    LEFT JOIN concept_synonym S ON C.concept_id = S.concept_id
        WHERE
            (C.vocabulary_id IN ('SNOMED', 'MedDRA') OR LOWER(C.concept_class_id)␣
↪= 'clinical finding' ) AND
            C.concept_class_id IS NOT NULL AND
            ( LOWER(C.concept_name) like '%myocardial infarction%' OR
            LOWER(S.concept_synonym_name) like '%myocardial infarction%' )
    ) T
WHERE sysdate BETWEEN valid_start_date AND valid_end_date
ORDER BY 6,2;
```

inspectomop.queries.condition.condition_concepts_for_source_codes

inspectomop.queries.condition.**condition_concepts_for_source_codes**(*source_codes*, *inspector*, *return_columns=None*)

Retrieves standard condition concepts for source codes. Ex ICD-9-CM –> SNOMED-CT

Parameters

- **source_codes** (*list of str*) – a list of source code strings. Ex ICD-9-CM [‘250.00’,‘250.01’]
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘source_code’, ‘source_concept_name’, ‘source_vocab_id’, ‘source_vocab_name’, ‘source_domain_id’,’target_concept_id’, ‘target_concept_name’, ‘target_concept_code’, ‘target_concept_class_id’,’target_vocab_id’, ‘target_vocab_name’]

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

Notes

Original SQL

C05: Translate a source code to condition concepts:

```
set search_path to full_201612_omop_v5;
SELECT DISTINCT
    c1.concept_code,
    c1.concept_name,
    c1.vocabulary_id source_vocabulary_id,
    VS.vocabulary_name source_vocabulary_description,
    C1.domain_id,
    C2.concept_id target_concept_id,
    C2.concept_name target_Concept_Name,
    C2.concept_code target_Concept_Code,
    C2.concept_class_id target_Concept_Class,
    C2.vocabulary_id target_Concept_Vocab_ID,
    VT.vocabulary_name target_Concept_Vocab_Name
FROM
    concept_relationship cr,
    concept c1,
    concept c2,
    vocabulary VS,
    vocabulary VT
WHERE
    cr.concept_id_1 = c1.concept_id AND
    cr.relationship_id = 'Maps to' AND
    cr.concept_id_2 = c2.concept_id AND
    c1.vocabulary_id = VS.vocabulary_id AND
    c1.domain_id = 'Condition' AND
    c2.vocabulary_id = VT.vocabulary_id AND
    c1.concept_code IN ('070.0') AND
    c2.vocabulary_id ='SNOMED' AND
    sysdate BETWEEN c1.valid_start_date AND c1.valid_end_date;
```

inspectomop.queries.condition.condition_concepts_occurring_at_anatomical_site_concept_id

inspectomop.queries.condition.**condition_concepts_occurring_at_anatomical_site_concept_id**(*co*
in
sp
to
re
tu

Retrieves condition concepts that occur at a given anatomical site. Input concept_id should be a concept of class 'Body Structure'

Parameters

- **concept_id** (*int*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query

– columns : ['cond_concept_id', 'cond_concept_name', 'cond_concept_code', 'cond_concept_class_id', 'cond_vocab_id', 'cond_vocab_name', 'anat_site_concept_id','anat_site_concept_name', 'anat_site_concept_code', 'anat_site_concept_class_id', 'anat_site_vocab_id', 'anat_site_vocab_name']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

See also:

anatomical_site_by_keyword()

Notes

Original SQL

C11: Find all SNOMED-CT condition concepts that are occurring at an anatomical site:

```
SELECT
    A.concept_Id Condition_ID,
    A.concept_Name Condition_name,
    A.concept_Code Condition_code,
    A.concept_Class_id Condition_class,
    A.vocabulary_id Condition_vocab_ID,
    VA.vocabulary_name Condition_vocab_name,
    D.concept_Id Anatomical_site_ID,
    D.concept_Name Anatomical_site_Name,
    D.concept_Code Anatomical_site_Code,
    D.concept_Class_id Anatomical_site_Class,
    D.vocabulary_id Anatomical_site_vocab_ID,
    VS.vocabulary_name Anatomical_site_vocab_name
FROM
    concept_relationship CR,
    concept A,
    concept D,
    vocabulary VA,
    vocabulary VS
WHERE
    CR.relationship_ID = 'Has finding site' AND
    CR.concept_id_1 = A.concept_id AND
    A.vocabulary_id = VA.vocabulary_id AND
    CR.concept_id_2 = D.concept_id AND
    D.concept_id = 4103720 --input AND
    D.vocabulary_id = VS.vocabulary_id AND
    sysdate BETWEEN CR.valid_start_date AND CR.valid_end_date;
```

inspectomop.queries.condition.conditions_caused_by_pathogen_or_causative_agent_concept_id

inspectomop.queries.condition.**conditions_caused_by_pathogen_or_causative_agent_concept_id**(

Retreives all conditions caused by a pathogen or other causative agent concept_id.

Parameters

- **concept_id** (*int*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['condition_concept_id', 'condition_name', 'condition_concept_code', 'condition_concept_class_id', 'condition_vocab_id', 'condition_vocab_name', 'causative_agent_concept_id', 'causative_agent_concept_name', 'causative_agent_concept_code', 'causative_agent_concept_class_id', 'causative_agent_vocab_id', 'causative_agent_vocab_name']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

See also:

disease_causing_agents_for_keyword(), *pathogen_concept_for_keyword()*

Notes

Original SQL

C09: Find all SNOMED-CT condition concepts that can be caused by a given pathogen or causative agent:

```
SELECT
    A.concept_Id Condition_ID,
    A.concept_Name Condition_name,
    A.concept_Code Condition_code,
    A.concept_Class_id Condition_class,
    A.vocabulary_id Condition_vocab_ID,
    VA.vocabulary_name Condition_vocab_name,
    D.concept_Id Causative_agent_ID,
    D.concept_Name Causative_agent_Name,
    D.concept_Code Causative_agent_Code,
    D.concept_Class_id Causative_agent_Class,
    D.vocabulary_id Causative_agent_vocab_ID,
    VS.vocabulary_name Causative_agent_vocab_name
FROM
    concept_relationship CR,
    concept A,
    concept D,
    vocabulary VA,
    vocabulary VS
WHERE
    CR.relationship_ID = 'Has causative agent' AND
    CR.concept_id_1 = A.concept_id AND
    A.vocabulary_id = VA.vocabulary_id AND
    CR.concept_id_2 = D.concept_id AND
    D.concept_id = 4248851 AND
    D.vocabulary_id = VS.vocabulary_id AND
    sysdate BETWEEN CR.valid_start_date AND CR.valid_end_date;
```

inspectomop.queries.condition.disease_causing_agents_for_keyword

inspectomop.queries.condition.**disease_causing_agents_for_keyword**(*keyword*, *inspector*, *return_columns=None*)

Retrieves disease causing agents by keyword. The concept_class_id can be any of: 'Pharmaceutical / biologic product', 'Physical object', 'Special concept', 'Event', 'Physical force', or 'Substance'.

Results of queries from *disease_causing_agents_for_keyword()* and *pathogen_concept_for_keyword()* can be used to search for conditions caused by the offending pathogens/agents using *conditions_caused_by_pathogen_or_causative_agent_concept_id()*

Parameters

- **keyword** (*str*) – search string. ex 'Radiation'
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['concept_id', 'concept_name', 'concept_code', 'concept_class_id', 'standard_concept', 'vocabulary_id', 'vocabulary_name']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

See also:

pathogen_concept_for_keyword(), *conditions_caused_by_pathogen_or_causative_agent_concept*

Notes

Original SQL

C08: Find a disease causing agent by keyword:

```
SELECT
    C.concept_id Agent_Concept_ID,
    C.concept_name Agent_Concept_Name,
    C.concept_code Agent_concept_code,
    C.concept_class_id Agent_concept_class,
    C.standard_concept Agent_Standard_Concept,
    C.vocabulary_id Agent_Concept_Vocab_ID,
    V.vocabulary_name Agent_Concept_Vocab_Name
FROM
    concept C,
    vocabulary V
WHERE
    LOWER(C.concept_class_id) in ('pharmaceutical / biologic product','physical␣
↪object','special concept','event', 'physical force','substance') AND
    LOWER(C.concept_name) like '%radiation%' AND
    C.vocabulary_id = V.vocabulary_id AND
sysdate BETWEEN C.valid_start_date AND C.valid_end_date;
```

inspectomop.queries.condition.source_codes_for_concept_ids

inspectomop.queries.condition.**source_codes_for_concept_ids**(*concept_ids*, *inspector*, *return_columns=None*)

Retreives source condition concepts for OMOP concept_ids. i.e SNOMED-CT –> ICD-9-CM, ICD-10-CM

Parameters

- **source_code** (*list of int*) – integer list of concept_ids to translate
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['concept_id','concept_code', 'concept_name', 'vocab_id', 'vocab_name', 'domain_id','source_concept_id', 'source_concept_name', 'source_concept_code', 'source_concept_class_id','source_vocab_id', 'source_vocab_name']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

Notes

Original SQL

C06: Translate a given condition to source codes:

```
set search_path to full_201612_omop_v5;
SELECT DISTINCT
    c1.concept_code,
    c1.concept_name,
    c1.vocabulary_id source_vocabulary_id,
    VS.vocabulary_name source_vocabulary_description,
    C1.domain_id,
    C2.concept_id target_concept_id,
    C2.concept_name target_Concept_Name,
    C2.concept_code target_Concept_Code,
    C2.concept_class_id target_Concept_Class,
    C2.vocabulary_id target_Concept_Vocab_ID,
    VT.vocabulary_name target_Concept_Vocab_Name
FROM
    concept_relationship cr,
    concept c1,
    concept c2,
    vocabulary VS,
    vocabulary VT
WHERE
    cr.concept_id_1 = c1.concept_id AND
    cr.relationship_id = 'Maps to' AND
    cr.concept_id_2 = c2.concept_id AND
    c1.vocabulary_id = VS.vocabulary_id AND
    c1.domain_id = 'Condition' AND
    c2.vocabulary_id = VT.vocabulary_id AND
    c1.concept_id = 312327 AND
    c1.vocabulary_id = 'SNOMED' AND
    sysdate BETWEEN c2.valid_start_date AND c2.valid_end_date;
```

inspectomop.queries.condition.pathogen_concept_for_keyword

inspectomop.queries.condition.**pathogen_concept_for_keyword**(*keyword*, *inspector*, *return_columns=None*)

Retrieves pathogen concepts based on a keyword with 'Organsim' as the concept_class_id.

Parameters

- **keyword** (*str*) – search string. ex 'Helicobacter Pylori'
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['concept_id', 'concept_name', 'concept_code', 'concept_class_id', 'standard_concept', 'vocabulary_id', 'vocabulary_name']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

See also:

disease_causing_agents_for_keyword(), *conditions_caused_by_pathogen_or_causative_agent_*

Notes

Original SQL

C07: Find a pathogen by keyword:

```
SELECT
    C.concept_id Pathogen_Concept_ID,
    C.concept_name Pathogen_Concept_Name,
    C.concept_code Pathogen_concept_code,
    C.concept_class_id Pathogen_concept_class,
    C.standard_concept Pathogen_Standard_Concept,
    C.vocabulary_id Pathogen_Concept_Vocab_ID,
    V.vocabulary_name Pathogen_Concept_Vocab_Name
FROM
    concept C,
    vocabulary V
WHERE
    LOWER(C.concept_class_id) = 'organism' AND
    LOWER(C.concept_name) like '%trypanosoma%' AND
    C.vocabulary_id = V.vocabulary_id AND
    sysdate BETWEEN C.valid_start_date AND C.valid_end_date;
```

inspectomop.queries.condition.place_of_service_counts_for_condition_concept_id

inspectomop.queries.condition.**place_of_service_counts_for_condition_concept_id**(*condition_concep*, *inspector*, *return_columns=N*

Provides counts of conditions stratified by place_of_service (Office, Inpatient Hospital, etc.)

Parameters

- **condition_concept_id** (*int*) – concept_id from the conditions table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['condition_concept_id', 'condition_concept_id,'place_of_service_concept_id', 'place_of_service', 'place_freq']

Returns **results** – a cursor-like object with methods such as fetchone(), fetchmany() etc.

Return type *inspectomop.results.Results*

Notes

SQL Modified from:

CO04: Count In what place of service where condition diagnoses:

```
SELECT
    c.concept_id AS condition_concept_id,
    c.concept_name AS condition_concept_id,
    cs.place_of_service_concept_id AS place_of_service_concept_id,
    c_place.concept_name AS place_of_service,
    count(cs.place_of_service_concept_id) AS place_freq
FROM
    main.concept AS c, main.concept AS c_place,
    (SELECT
        co.condition_concept_id AS condition_concept_id,
        co.visit_occurrence_id AS s1_visit_id
    FROM
        main.condition_occurrence AS co
    WHERE
        co.condition_concept_id = :condition_concept_id_1
        AND co.visit_occurrence_id IS NOT NULL)
JOIN
    main.visit_occurrence AS vo
ON
    s1_visit_id = vo.visit_occurrence_id
JOIN
    main.care_site AS cs
ON
    vo.care_site_id = cs.care_site_id
WHERE
    c_place.concept_id = cs.place_of_service_concept_id
    AND c.concept_id = :concept_id_1
GROUP BY c.concept_name
```

Drug

inspectomop.queries.drug

drug_classes_for_drug_concept_id(concept_id, ...)	Returns drug classes for drug or ingredient concept_ids.
drug_concepts_for_ingredient_concept_id(...)	Get all drugs that contain a given ingredient.
indications_for_drug_concept_id(concept_id, ...)	Find all indications for a drug given a concept_id.
ingredients_for_drug_concept_ids(..., ...)	Get ingredients for brand or generic drug concept_ids.
ingredient_concept_ids_for_ingredient_names(...)	Get concept_ids for a list of ingredients.

inspectomop.queries.drug.drug_classes_for_drug_concept_id

inspectomop.queries.drug.**drug_classes_for_drug_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Returns drug classes for drug or ingredient concept_ids.

Parameters

- **concept_id** (*int*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['concept_id','concept_name','concept_code','concept_class_id', 'vocabulary_id', 'vocabulary_name']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

D08: Find drug classes for a drug or ingredient:

```
SELECT
    c1.concept_id                Class_Concept_Id,
    c1.concept_name              Class_Name,
    c1.concept_code              Class_Code,
    c1.concept_class_id          Classification,
    c1.vocabulary_id             Class_vocabulary_id,
    v1.vocabulary_name           Class_vocabulary_name,
    ca.min_levels_of_separation  Levels_of_Separation
FROM
    concept_ancestor             ca,
    concept                      c1,
    vocabulary                   v1
WHERE
    ca.ancestor_concept_id = c1.concept_id
    AND    c1.vocabulary_id IN ('NDFRT', 'ETC', 'ATC', 'VA Class')
    AND    c1.concept_class_id IN ('ATC','VA Class','Mechanism of Action',
↪'Chemical Structure','ETC','Physiologic Effect')
    AND    c1.vocabulary_id = v1.vocabulary_id
```

(continues on next page)

(continued from previous page)

```
    AND    ca.descendant_concept_id = 1545999
    AND    sysdate BETWEEN c1.valid_start_date AND c1.valid_end_date;
```

inspectomop.queries.drug.drug_concepts_for_ingredient_concept_id

inspectomop.queries.drug.**drug_concepts_for_ingredient_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Get all drugs that contain a given ingredient.

Parameters

- **concept_id** (*int*) – concept_id corresponding to a drug ingredient
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['ingredient_concept_id', 'ingredient_name', 'ingredient_concept_code', 'ingredient_concept_class_id', 'drug_concept_id', 'drug_name','drug_concept_code', 'drug_concept_class_id']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

D04: Find drugs by ingredient:

```
SELECT
    A.concept_id Ingredient_concept_id,
    A.concept_Name Ingredient_name,
    A.concept_Code Ingredient_concept_code,
    A.concept_Class_id Ingredient_concept_class,
    D.concept_id Drug_concept_id,
    D.concept_Name Drug_name,
    D.concept_Code Drug_concept_code,
    D.concept_Class_id Drug_concept_class
FROM
    concept_ancestor CA,
    concept A,
    concept D
WHERE
    CA.ancestor_concept_id = A.concept_id
    AND CA.descendant_concept_id = D.concept_id
    AND sysdate BETWEEN A.valid_start_date AND A.valid_end_date
    AND sysdate BETWEEN D.valid_start_date AND D.valid_end_date
    AND CA.ancestor_concept_id = 966991;
```

inspectomop.queries.drug.indications_for_drug_concept_id

inspectomop.queries.drug.**indications_for_drug_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Find all indications for a drug given a concept_id. Returns matches from NDFRT, FDB, and corresponding SNOMED conditions.

*Note: The results set should be filtered by 'c_domain_id' == 'Condition'

Parameters

- **concept_id** (*int*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['c_concept_id', 'c_concept_name', 'c_domain_id', 'min_levels_of_separation', 'an_concept_id', 'an_concept_name', 'an_vocab', 'de_concept_id', de_concept_name', 'de_vocab']

Returns results

Return type *inspectomop.results.Results*

Notes

SQL (see 2nd example for actual implimentation)

D13: Find indications as condition concepts for a drug:

```
select
    a.min_levels_of_separation as a_min,
    an.concept_id as an_id,
    an.concept_name as an_name,
    an.vocabulary_id as an_vocab,
    an.domain_id as an_domain,
    an.concept_class_id as an_class,
    de.concept_id as de_id,
    de.concept_name as de_name,
    de.vocabulary_id as de_vocab,
    de.domain_id as de_domain,
    de.concept_class_id as de_class
from
    concept an
join
    concept_ancestor a on a.ancestor_concept_id=an.concept_id
join
    concept de on de.concept_id=a.descendant_concept_id
where
    an.concept_class_id in ('Ind / CI', 'Indication') -- One is for NDFRT, the␣
↪other for FDB Indications
    and de.vocabulary_id in ('RxNorm', 'RxNorm Extension') -- You don't need that␣
↪if you join directly with DRUG_EXPOSURE
    and lower(an.concept_name) like '%diabetes%'
```

To tie directly to SNOMED concepts, this query is used

select c.concept_id as c_id, c.concept_name as c_name, c.vocabulary_id as c_vocab, c.domain_id as c_domain, c.concept_class_id as c_class, – Condition de.concept_id as de_id, de.concept_name as de_name, de.vocabulary_id as de_vocab, de.domain_id as de_domain, de.concept_class_id as de_class – Drug

from concept an – Indications

join concept_ancestor a on a.ancestor_concept_id=an.concept_id – connect to

join concept de on de.concept_id=a.descendant_concept_id – . . . drug

join concept_relationship r on r.concept_id_1=an.concept_id – connect to

join concept c on c.concept_id=r.concept_id_2 and c.domain_id='Condition' – Snomed Conditions

where an.concept_class_id in ('Ind / CI', 'Indication') and de.vocabulary_id in ('RxNorm', 'RxNorm Extension') and lower(c.concept_name) like '%diabet%'

inspectomop.queries.drug.ingredients_for_drug_concept_ids

inspectomop.queries.drug.**ingredients_for_drug_concept_ids**(*concept_ids*, *inspector*, *return_columns=None*)

Get ingredients for brand or generic drug concept_ids.

Parameters

- **concept_id** (*list of int*) – concept_ids corresponding to brand or generic drug concept_ids
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['drug_concept_id', 'drug_name', 'drug_concept_code', 'drug_concept_class', 'ingredient_concept_id', 'ingredient_name', 'ingredient_concept_code', 'ingredient_concept_class']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

D03: Find ingredients of a drug::

SELECT D.Concept_Id drug_concept_id, D.Concept_Name drug_name, D.Concept_Code drug_concept_code, D.Concept_Class_id drug_concept_class, A.Concept_Id ingredient_concept_id, A.Concept_Name ingredient_name, A.Concept_Code ingredient_concept_code, A.Concept_Class_id ingredient_concept_class

FROM full_201706_omop_v5.concept_ancestor CA, full_201706_omop_v5.concept A, full_201706_omop_v5.concept D

WHERE CA.descendant_concept_id = D.concept_id AND CA.ancestor_concept_id = A.concept_id AND LOWER(A.concept_class_id) = 'ingredient' AND sysdate BETWEEN A.VALID_START_DATE AND A.VALID_END_DATE AND sysdate BETWEEN

D.VALID_START_DATE AND D.VALID_END_DATE AND CA.descendant_concept_id IN (939355, 19102189, 19033566)

inspectomop.queries.drug.ingredient_concept_ids_for_ingredient_names

inspectomop.queries.drug.**ingredient_concept_ids_for_ingredient_names**(*ingredient_names*, *inspector*, *return_columns=None*)

Get concept_ids for a list of ingredients.

Parameters

- **ingredient_names** (*list of str*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘ingredient_name’, ‘concept_id’]

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL:

```
SELECT
    concept_id,
    concept_name
FROM
    concept,
WHERE
    vocabulary_id = 'RxNorm'
    AND concept_class_id = 'Ingredient'
    AND lower(concept_name) IN ('ingredient_name_1')
```

General

inspectomop.queries.general

ancestors_for_concept_id(concept_id, inspector)	Find all ancestor concepts for a concept_id.
children_for_concept_id(concept_id, inspector)	Find all child concepts for a concept_id.
concepts_for_concept_ids(concept_ids, inspector)	Returns concept information for a list of concept_ids
descendants_for_concept_id(concept_id, inspector)	Find all descendant concepts for a concept_id.
parents_for_concept_id(concept_id, inspector)	Find all parent concepts for a concept_id.

Continued on next page

Table 13 – continued from previous page

related_concepts_for_concept_id(concept_id, ...)	Find all concepts related to a concept_id.
siblings_for_concept_id(concept_id, inspector)	Find all sibling concepts for a concept_id i.e.(concepts that share common parents).
synonyms_for_concept_ids(concept_ids, inspector)	Returns concept information for a list of concept_ids
standard_vocab_for_source_code(source_code, ...)	Convert source code to all mapped standard vocabulary concepts.

inspectomop.queries.general.ancestors_for_concept_id

inspectomop.queries.general.**ancestors_for_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Find all ancestor concepts for a concept_id.

Parameters

- **concept_id** (*int*) – concept_id of interest from the concept table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['ancestor_concept_id', 'ancestor_concept_name', 'ancestor_concept_code', 'ancestor_concept_class_id', 'vocabulary_id', 'min_levels_of_separation', 'max_levels_of_separation']

Returns **results**

Return type *inspectomop.results.Results*

Notes

Original SQL

G08: Find ancestors for a given concept:

```
SELECT
    C.concept_id as ancestor_concept_id,
    C.concept_name as ancestor_concept_name,
    C.concept_code as ancestor_concept_code,
    C.concept_class_id as ancestor_concept_class_id,
    C.vocabulary_id,
    VA.vocabulary_name,
    A.min_levels_of_separation,
    A.max_levels_of_separation
FROM
    concept_ancestor A,
    concept C,
    vocabulary VA
WHERE
    A.ancestor_concept_id = C.concept_id AND
    C.vocabulary_id = VA.vocabulary_id AND A.ancestor_concept_id<>A.descendant_
↪concept_id AND A.descendant_concept_id = 192671 AND
```

(continues on next page)

(continued from previous page)

```
    sysdate BETWEEN valid_start_date AND valid_end_date
ORDER BY 5,7;
```

inspectomop.queries.general.children_for_concept_id

inspectomop.queries.general.**children_for_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Find all child concepts for a concept_id.

Parameters

- **concept_id** (*int*) – concept_id of interest from the concept table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['child_concept_id','child_concept_name', 'child_concept_code', 'child_concept_class_id', 'child_concept_vocabulary_id', 'child_concept_vocab_name']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

G11: Find children for a given concept:

```
SELECT
    D.concept_id Child_concept_id,
    D.concept_name Child_concept_name,
    D.concept_code Child_concept_code,
    D.concept_class_id Child_concept_class_id,
    D.vocabulary_id Child_concept_vocab_ID,
    VS.vocabulary_name Child_concept_vocab_name
FROM
    concept_ancestor CA,
    concept D,
    vocabulary VS
WHERE
    CA.ancestor_concept_id = 192671 AND
    CA.min_levels_of_separation = 1 AND
    CA.descendant_concept_id = D.concept_id AND
    D.vocabulary_id = VS.vocabulary_id AND
    sysdate BETWEEN D.valid_start_date AND D.valid_end_date;
```

inspectomop.queries.general.concepts_for_concept_ids

inspectomop.queries.general.**concepts_for_concept_ids**(*concept_ids*, *inspector*, *return_columns=None*)

Returns concept information for a list of concept_ids

Parameters

- **concept_ids** (*list of int*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['concept_id', 'concept_name', 'concept_code', 'concept_class_id', 'standard_concept', 'vocabulary_id', 'vocabulary_name']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

G01: Find concept by concept ID:

```
SELECT
    C.concept_id,
    C.concept_name,
    C.concept_code,
    C.concept_class_id,
    C.standard_concept,
    C.vocabulary_id,
    V.vocabulary_name
FROM
    concept C,
    vocabulary V
WHERE
    C.concept_id = 192671 AND
    C.vocabulary_id = V.vocabulary_id AND
    sysdate BETWEEN valid_start_date AND valid_end_date;
```

inspectomop.queries.general.descendants_for_concept_id

inspectomop.queries.general.**descendants_for_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Find all descendant concepts for a concept_id.

Parameters

- **concept_id** (*int*) – concept_id of interest from the concept table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['descendant_concept_id', 'descendant_concept_name', 'descendant_concept_code', 'descendant_concept_class_id', 'vocabulary_id', 'min_levels_of_separation', 'max_levels_of_separation']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

G09: Find descendants for a given concept:

```
SELECT
    C.concept_id as descendant_concept_id,
    C.concept_name as descendant_concept_name,
    C.concept_code as descendant_concept_code,
    C.concept_class_id as descendant_concept_class_id,
    C.vocabulary_id,
    VA.vocabulary_name,
    A.min_levels_of_separation,
    A.max_levels_of_separation
FROM
    concept_ancestor A,
    concept C,
    vocabulary VA
WHERE
    A.ancestor_concept_id = C.concept_id AND
    C.vocabulary_id = VA.vocabulary_id AND A.ancestor_concept_id<>A.descendant_
↪concept_id AND A.descendant_concept_id = 192671 AND
    sysdate BETWEEN valid_start_date AND valid_end_date
ORDER BY 5,7;
```

inspectomop.queries.general.parents_for_concept_id

inspectomop.queries.general.**parents_for_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Find all parent concepts for a concept_id. (Ancestors whose level of separation is 1)

Parameters

- **concept_id** (*int*) – concept_id of interest from the concept table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘parent_concept_id’, ‘parent_concept_name’, ‘parent_concept_code’, ‘parent_concept_class_id’, ‘parent_concept_vocabulary_id’, ‘parent_concept_vocab_name’]

Returns **results**

Return type *inspectomop.results.Results*

Notes

Original SQL

G10: Find parents for a given concept:

```
SELECT
    A.concept_id Parent_concept_id,
    A.concept_name Parent_concept_name,
    A.concept_code Parent_concept_code,
    A.concept_class_id Parent_concept_class_id,
    A.vocabulary_id Parent_concept_vocab_ID,
    VA.vocabulary_name Parent_concept_vocab_name
FROM
    concept_ancestor CA,
    concept A,
    concept D,
    vocabulary VA
WHERE
    CA.descendant_concept_id = 192671 AND
    CA.min_levels_of_separation = 1 AND
    CA.ancestor_concept_id = A.concept_id AND
    A.vocabulary_id = VA.vocabulary_id AND
    CA.descendant_concept_id = D.concept_id AND
    sysdate BETWEEN A.valid_start_date AND A.valid_end_date;
```

inspectomop.queries.general.related_concepts_for_concept_id

inspectomop.queries.general.**related_concepts_for_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Find all concepts related to a concept_id.

Parameters

- **concept_id** (*int*) – concept_id of interest from the concept table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘relationship_polarity’,’relationship_id’, ‘relationship_name’, ‘concept_id’, ‘concept_name’, ‘concept_code’, ‘concept_class_id’, ‘vocabulary_id’, ‘vocabulary_name’]

Returns **results**

Return type *inspectomop.results.Results*

Notes

Original SQL

G07: Find concepts that have a relationship with a given concept:

```
SELECT
    'Relates to' relationship_polarity,
    CR.relationship_ID,
    RT.relationship_name,
    D.concept_Id concept_id,
    D.concept_Name concept_name,
```

(continues on next page)

(continued from previous page)

```
    D.concept_Code concept_code,
    D.concept_class_id concept_class_id,
    D.vocabulary_id concept_vocab_ID,
    VS.vocabulary_name concept_vocab_name
FROM
    concept_relationship CR,
    concept A,
    concept D,
    vocabulary VA,
    vocabulary VS,
    relationship RT
WHERE
    CR.concept_id_1 = A.concept_id AND
    A.vocabulary_id = VA.vocabulary_id AND
    CR.concept_id_2 = D.concept_id AND
    D.vocabulary_id = VS.vocabulary_id AND
    CR.relationship_id = RT.relationship_ID AND
    A.concept_id = 192671 AND
    sysdate BETWEEN CR.valid_start_date AND CR.valid_end_date
UNION ALL
SELECT
    'Is related by' relationship_polarity,
    CR.relationship_ID,
    RT.relationship_name,
    A.concept_Id concept_id,
    A.concept_name concept_name,
    A.concept_code concept_code,
    A.concept_class_id concept_class_id,
    A.vocabulary_id concept_vocab_ID,
    VA.Vocabulary_Name concept_vocab_name
FROM
    concept_relationship CR,
    concept A,
    concept D,
    vocabulary VA,
    vocabulary VS,
    relationship RT
WHERE
    CR.concept_id_1 = A.concept_id AND
    A.vocabulary_id = VA.vocabulary_id AND
    CR.concept_id_2 = D.concept_id AND
    D.vocabulary_id = VS.vocabulary_id AND
    CR.relationship_id = RT.relationship_ID AND
    D.concept_id = 192671 AND
    sysdate BETWEEN CR.valid_start_date AND CR.valid_end_date;
```

inspectomop.queries.general.siblings_for_concept_id

inspectomop.queries.general.**siblings_for_concept_id**(*concept_id*, *inspector*, *return_columns=None*)

Find all sibling concepts for a concept_id i.e.(concepts that share common parents). This may or may not result in concepts that have a close clinical relationship, especially if the query concept_id is already high up in the hierarchy or has multiple parents that diverge in their meaning.

Parameters

- **concept_id** (*int*) – concept_id of interest from the concept table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - **columns** [['sibling_concept_id', 'sibling_concept_name','sibling_concept_code','sibling_concept_class_id',] 'sibling_concept_vocabulary_id,'parent_concept_id','parent_concept_name']

Returns results

Return type *inspectomop.results.Results*

Notes

SQL:

```
SELECT
    s.concept_id AS sibling_concept_id,
    s.concept_name AS sibling_concept_name,
    a.concept_id AS parent_concept_id,
    a.concept_name AS parent_concept_name
FROM
    main.concept AS s,
    main.concept AS a,
    main.concept_ancestor AS ca,
    main.vocabulary AS va,
    main.concept AS d,
    main.concept_ancestor AS ca2
WHERE
    ca.descendant_concept_id = concept_id AND
    ca.min_levels_of_separation = 1 AND
    ca.ancestor_concept_id = a.concept_id AND
    a.vocabulary_id = va.vocabulary_id AND
    ca.descendant_concept_id = d.concept_id AND
    ca2.ancestor_concept_id = ca.ancestor_concept_id AND
    s.concept_id = ca2.descendant_concept_id
```

inspectomop.queries.general.synonyms_for_concept_ids

inspectomop.queries.general.**synonyms_for_concept_ids**(*concept_ids*, *inspector*, *return_columns=None*)

Returns concept information for a list of concept_ids

Parameters

- **concept_ids** (*list of int*) –
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['concept_id', 'concept_synonym_name']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

G04: Find synonyms for a given concept ID:

```
SELECT
    C.concept_id,
    S.concept_synonym_name
FROM
    concept C,
    concept_synonym S,
    vocabulary V
WHERE
    C.concept_id = 192671 AND
    C.concept_id = S.concept_id AND
    C.vocabulary_id = V.vocabulary_id AND
    sysdate BETWEEN C.valid_start_date AND C.valid_end_date;
```

inspectomop.queries.general.standard_vocab_for_source_code

inspectomop.queries.general.**standard_vocab_for_source_code**(*source_code*, *source_vocab_id*, *inspector*, *return_columns=None*)

Convert source code to all mapped standard vocabulary concepts.

Parameters

- **source_codes** (*str*) – alphanumeric source_code to query on e.g ICD-9 '250.00'
- **source_vocab_id** (*str*) –
  - vocabulary_id from the vocabulary table e.g 'ICD9CM'
  - see https://github.com/OHDSI/CommonDataModel/wiki/VOCABULARY for a full list
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['domain_id', 'concept_id', 'concept_name', 'concept_code', 'concept_class_id', 'vocabulary_id', 'target_concept_domain']

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

G05: Translate a code from a source to a standard vocabulary:

```
SELECT DISTINCT
    c1.domain_id,
    c2.concept_id          as Concept_Id,
```

(continues on next page)

(continued from previous page)

```
    c2.concept_name       as Concept_Name,
    c2.concept_code       as Concept_Code,
    c2.concept_class_id   as Concept_Class,
    c2.vocabulary_id      as Concept_Vocabulary_ID,
    c2.domain_id          as Target_concept_Domain
FROM
    concept_relationship cr
JOIN
    concept c1 ON c1.concept_id = cr.concept_id_1
JOIN
    concept c2 ON c2.concept_id = cr.concept_id_2
WHERE
    cr.relationship_id = 'Maps to' AND
    c1.concept_code IN ('070.0') AND
    c1.vocabulary_id = 'ICD9CM' AND
    sysdate BETWEEN cr.valid_start_date AND cr.valid_end_date;
```

Observation

inspectomop.queries.observation

observation_concepts_for_keyword(keyword, ...)	Search for LOINC and UCUM concepts by keyword.

inspectomop.queries.observation.observation_concepts_for_keyword

inspectomop.queries.observation.**observation_concepts_for_keyword**(*keyword*, *inspector*, *return_columns=None*)

Search for LOINC and UCUM concepts by keyword.

Parameters

- **keyword** (*str*) – e.x. 'LDL'
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - **columns** [['concept_id','concept_name','concept_code', 'concept_class_id', 'vocabulary_id',] 'vocabulary_name']

Returns **results**

Return type *inspectomop.results.Results*

Notes

Original SQL

O1: Find a Observation from a keyword:

```
SELECT
    T.Entity_Concept_Id,
    T.Entity_Name,
    T.Entity_Code,
    T.Entity_Type,
    T.Entity_concept_class_id,
    T.Entity_vocabulary_id,
    T.Entity_vocabulary_name
FROM   (
    SELECT
        C.concept_id       Entity_Concept_Id,
        C.concept_name     Entity_Name,
        C.concept_code     Entity_Code,
        'Concept'          Entity_Type,
        C.concept_class_id    Entity_concept_class_id,
        C.vocabulary_id    Entity_vocabulary_id,
        V.vocabulary_name  Entity_vocabulary_name,
        C.valid_start_date,
        C.valid_end_date
    FROM
        concept         C,
        vocabulary      V
    WHERE
        C.vocabulary_id IN ('LOINC', 'UCUM') AND
        C.concept_class_id IS NOT NULL AND
        C.standard_concept = 'S' AND
        C.vocabulary_id = V.vocabulary_id
       ) T
WHERE
    REGEXP_INSTR(LOWER(REPLACE(REPLACE(T.Entity_Name, ' ', ''), '-', '')),
        LOWER(REPLACE(REPLACE('LDL' , ' ', ''), '-', ''))) > 0 AND
    sysdate BETWEEN T.valid_start_date AND T.valid_end_date
```

Payer Plan

inspectomop.queries.payer_plan

counts_by_years_of_coverage(inspector)	Returns counts of payer coverage based on continuous coverage (payer_plan_period_start_date - payer_plan_period_end_date)365.25.
patient_distribution_by_plan_type(inspector)	Returns counts of payer coverage by plan type.

inspectomop.queries.payer_plan.counts_by_years_of_coverage

inspectomop.queries.payer_plan.**counts_by_years_of_coverage**(*inspector*)

Returns counts of payer coverage based on continuous coverage (payer_plan_period_start_date - payer_plan_period_end_date)365.25. Note this method may count patients with more than one insurance plan multiple times. Ex pt with Medicare Parts A, B, and D.

Parameters **inspector** (inspectomop.inspector.Inspector) –

Returns **df**

Return type pandas.DataFrame

Notes

Original SQL

PP01: Continuous years with patient counts:

```
SELECT
    floor((p.payer_plan_period_end_date - p.payer_plan_period_start_date)/365) AS␣
↪year_int,
    count(1) AS num_patients
FROM
    payer_plan_period p
GROUP BY
    floor((p.payer_plan_period_end_date - p.payer_plan_period_start_date)/365)
ORDER BY 1;
```

inspectomop.queries.payer_plan.patient_distribution_by_plan_type

inspectomop.queries.payer_plan.**patient_distribution_by_plan_type**(*inspector*)

Returns counts of payer coverage by plan type.

Parameters **inspector** (inspectomop.inspector.Inspector) –

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

PP02: Patient distribution by plan type:

```
SELECT
  t.plan_source_value,
  t.pat_cnt AS num_patients,
  100.00\*t.pat_cnt/ (sum(t.pat_cnt) over()) perc_of_total_count
FROM (
    SELECT
        p.plan_source_value,
        count(1) AS pat_cnt
    FROM
        payer_plan_period p
    GROUP BY
        p.plan_source_value
    ) t
ORDER BY
    t.plan_source_value;
```

Person

inspectomop.queries.person

patient_counts_by_gender(inspector[, ...])	Returns patient counts grouped by gender for the database or alternativily, for a supplied list of person_ids.
patient_counts_by_year_of_birth(inspector[, ...])	Returns patient counts grouped by year of birth for the database or alternativily, for a supplied list of person_ids.
patient_counts_by_residence_state(inspector[, ...])	Returns patient counts grouped by state for the database or alternativily, for a supplied list of person_ids.
patient_counts_by_zip_code(inspector[, ...])	Returns patient counts grouped by zip code for the database or alternativily, for a supplied list of person_ids.
patient_counts_by_year_of_birth_and_gender(...)	Returns patient counts stratified by year of birth and gender for the database or alternativily, for a supplied list of person_ids.

inspectomop.queries.person.patient_counts_by_gender

inspectomop.queries.person.**patient_counts_by_gender**(*inspector*, *person_ids=None*, *return_columns=None*)

Returns patient counts grouped by gender for the database or alternativily, for a supplied list of person_ids.

Parameters

- **person_ids** (*list of int, optional*) – list of person_ids [int]. If None (default), get the gender distribution for all individuals in the person table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : ['gender_concept_id','gender','count']

Returns **results**

Return type *inspectomop.results.Results*

Notes

Original SQL

PE03: Number of patients grouped by gender:

```
SELECT
    person.GENDER_CONCEPT_ID,
    concept.CONCEPT_NAME AS gender_name,
    COUNT(person.person_ID) AS num_persons_count
FROM
    person
INNER JOIN
    concept ON person.GENDER_CONCEPT_ID = concept.CONCEPT_ID
GROUP BY
    person.GENDER_CONCEPT_ID, concept.CONCEPT_NAME;
```

inspectomop.queries.person.patient_counts_by_year_of_birth

inspectomop.queries.person.**patient_counts_by_year_of_birth**(*inspector*, *person_ids=None*, *return_columns=None*)

Returns patient counts grouped by year of birth for the database or alternativily, for a supplied list of person_ids.

Parameters

- **person_ids** (*list of int, optional*) – list of person_ids [int]. If None (default), get the gender distribution for all individuals in the person table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘year_of_birth’, ‘count’]

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

PE06: Number of patients grouped by year of birth:

```
SELECT
    year_of_birth,
    COUNT(person_id) AS Num_Persons_count
FROM
    person
GROUP BY
    year_of_birth
ORDER BY
    year_of_birth;
```

inspectomop.queries.person.patient_counts_by_residence_state

inspectomop.queries.person.**patient_counts_by_residence_state**(*inspector*, *person_ids=None*, *return_columns=None*)

Returns patient counts grouped by state for the database or alternativily, for a supplied list of person_ids.

Parameters

- **person_ids** (*list of int, optional*) – list of person_ids [int]. If None (default), get the gender distribution for all individuals in the person table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘state’, ‘count’]

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

PE07: Number of patients grouped by residence state location:

```
SELECT
    NVL( state, 'XX' ) AS state_abbr,
    count(\*) as Num_Persons_count
FROM
    person
LEFT OUTER JOIN
    location USING( location_id )
GROUP BY
    NVL( state, 'XX' )
ORDER BY 1;
```

inspectomop.queries.person.patient_counts_by_zip_code

inspectomop.queries.person.**patient_counts_by_zip_code**(*inspector*, *person_ids=None*, *return_columns=None*)

Returns patient counts grouped by zip code for the database or alternativily, for a supplied list of person_ids.

Parameters

- **person_ids** (*list of int, optional*) – list of person_ids [int]. If None (default), get the gender distribution for all individuals in the person table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘state’,’zip_code’, ‘count’]

Returns results

Return type *inspectomop.results.Results*

Notes

Original SQL

PE08: Number of patients grouped by zip code of residence:

```
SELECT
    state,
    NVL( zip, '9999999' ) AS zip,
    count(\*) Num_Persons_count
FROM
    person
LEFT OUTER JOIN
    location
USING( location_id )
```

(continues on next page)

(continued from previous page)

```
GROUP BY
    state,
    NVL( zip, '9999999' )
ORDER BY 1, 2;
```

inspectomop.queries.person.patient_counts_by_year_of_birth_and_gender

inspectomop.queries.person.**patient_counts_by_year_of_birth_and_gender**(*inspector*, *person_ids=None*, *return_columns=None*)

Returns patient counts stratified by year of birth and gender for the database or alternativily, for a supplied list of person_ids.

Parameters

- **person_ids** (*list of int, optional*) – list of person_ids [int]. If None (default), get the gender distribution for all individuals in the person table
- **inspector** (inspectomop.inspector.Inspector) –
- **return_columns** (*list of str, optional*) –
  - optional subset of columns to return from the query
  - columns : [‘gender_concept_id’,’gender’,’year_of_birth’, ‘count’]

Returns **results**

Return type *inspectomop.results.Results*

Notes

Original SQL

PE09: Number of patients by gender, stratified by year of birth:

```
SELECT
    gender_concept_id,
    c.concept_name AS gender_name,
    year_of_birth,
    COUNT(p.person_id) AS num_persons
FROM
    person p
INNER JOIN
    concept c ON p.gender_concept_id = c.concept_id
GROUP BY
    gender_concept_id,
    c.concept_name,
    year_of_birth
ORDER BY
    concept_name,
    year_of_birth;
```

1.2 Indices and tables

- genindex
- modindex
- search

1.3 Acknowledgements

- This package builds off of amazing open source packages from the python community. Many thanks to the developers and maintainers of pandas, SQLAlchemy, and numpy.
- A big thank you to the OHDSI group for their continued efforts on improving the OMOP CDM and developing outstanding tools to advance the field of medical informatics.
- Most of the queries included in **inspectomop** were derived from the OMOP-Queries repository on GitHub.
- Test data were taken from a 1k sample of patients from the SynPUF dataset converted by to the OMOP CDM and provided by LTS Computing LLC

Index

Symbols

A

C

D

E

F

H

I

M

O

P

R

S

T

V