idpp.db.util.IdPPdb

Object providing an interface for interacting with the IdPP database.

Utility

create_db

idpp.db.util.create_db(*f: str, overwrite: bool = False*)→ None

creates a sqlite database for storing molecular properties

raises a RuntimeError if the database already exists unless overwrite is True

Parameters:

f : `str`

filename/path of the database

overwrite : `bool`, **default=False**

if the database file already exists and this flag is True, then overwrite existing database and do not raise the RuntimeError

Overview

class **idpp.db.util.IdPPdb**(*db_path: str, read_only: bool = False, enforce_idpp_ver: bool = True, combine_ms2: bool = False*)

Object for interacting with IdPP database

Attributes:

db_path : `str`

path to IdPP database file

read_only : `bool`

database is read-only

enforce_idpp_ver : `bool`

raise a RuntimeError if version of this package does not match idpp_ver in the database

cur : `sqlite3.Cursor`

cursor for making queries directly from the underlying database

uncommitted_changes : `bool`

flag indicating whether there are uncommitted changes to the database

version_info : `dict(str:str)`

version information from the database (key: component, value: version)

change_log : `list(dict(str:str))`

database change log (list of dicts with 'tstamp', 'author' and 'notes' entries)

last_qry : `str`

stores the last query that was run, useful reference for the SQL queries that are being run behind the scenes and helpful for debugging

check_insert_hits : `int`

count how many hits (i.e. an existing entry ID was returned) there were in this session from all insert method calls that use self._check_insert

check_insert_misses : `int`

count how many misses (i.e. a new entry was added) there were in this session from all insert method calls that use self._check_insert

last_check_insert_was_hit : `bool`

flag indicating if the last check_insert was a hit

ledger_size : `dict(str:int)`

dict mapping the number of elements currently being stored (value) in the ledger for each table (key)

ledger_mem : `int`

estimated current memory footprint of the ledger in MB, this is a decent approximation of the total memory footprint of an IdPPdb instance since the ledger is by far the largest component (assuming the interface was not initialized as read-only)

combine_ms2 : `bool`

indicates whether MS2 spectra are combined on entry

Methods

`close`([ignore_uncommitted_changes])	close connection to the database, release the ledge
`commit`()	commit changes to the databse (write to file)
`fetch_adduct_data`(n_rows[, ionization])	fetch data from Adducts table (joined to Compound
`fetch_adduct_data_extended`(n_rows[, ...])	(extended) fetch data from Adducts table (joined to
`fetch_adduct_id_by_cmpd_id`(cmpd_id[, ...])	Fetch all adduct_ids associated with an input cmpd
`fetch_ccs_by_adduct_id`(adduct_id[, ...])	fetch data from CCSs table corresponding to a spec
`fetch_ccs_data`(n_rows[, ionization, ...])	fetch data from CCSs table (joined to Adducts, Com
`fetch_cmpd_data`(n_rows)	fetch data from Compounds table (joined to Formu
`fetch_form_data`(n_rows)	fetch data from Formulas table yields one batch of
`fetch_inchi_data`(n_rows)	fetch data from InChIs table yields one batch of rov
`fetch_ms2_data`(n_rows[, ionization, ...])	fetch data from MS2Spectra, MS2Fragments, MS2S
`fetch_ms2_ids_by_adduct_id`(adduct_id[, ...])	fetch IDs from MS2Spectra table corresponding to
`fetch_rt_by_adduct_id`(adduct_id[, ...])	fetch data from RTs table corresponding to a specif
`fetch_rt_by_cmpd_id`(cmpd_id[, select_sources])	fetch data from RTs table corresponding to a specif
`fetch_rt_data`(n_rows[, ionization, ...])	fetch data from RTs table (joined to Adducts, Comp
`fetch_smi_data`(n_rows)	fetch data from Smiles table yields one batch of rov

`fetch_src_data` (n_rows)	fetch data from Sources table yields one batch of r
`insert_adduct` (adduct, cmpd_id, mz, z)	insert an entry into the Compounds table, return th
`insert_analysis_result` (dataset_id, counts, ...)	Insert the results from one trial of probability analy
`insert_ccs` (ccs, adduct_id, src_id)	insert an entry into the CCSs table, return the rowi
`insert_change_log_entry` (author, notes)	insert an entry into the change log with automatica
`insert_class_definition` (class_name[, notes])	insert an entry into the ClassDefs table, defining a
`insert_class_label` (cls_id, cmpd_id)	insert a class label (specified by cls_id) for a compou
`insert_cmpd` (name[, form_id, smi_id, inchi_id])	insert an entry into the Compounds table, return th
`insert_dataset` (description, query)	insert an entry into the Datasets table
`insert_ext_id` (cmpd_id, src_id, ext_id)	insert an entry into the ExternalIDs table
`insert_form` (form)	insert an entry into the Formulas table, return the r
`insert_inchi` (inchi_key[, inchi])	insert an entry into the InChIs table, return the row
`insert_ms2` (ms2_mz, ms2_i, adduct_id, src_id)	insert an entry into the MS2Spectra table, return th
`insert_rt` (rt, adduct_id, src_id)	insert an entry into the RTs table, return the rowid
`insert_smi` (smi)	insert an entry into the Smiles table, return the row
`insert_src` (src_name, src_ref[, src_notes])	insert an entry into the Sources table, return the ro
`update_version_info` ()	update version information stored in the database
`vacuum` ()	run the VACUUM command on the database, to sh

General Methods

`IdPPdb.__init__`

idpp.db.util.IdPPdb.__init__(*self, db_path: str, read_only: bool = False, enforce_idpp_ver: bool = True, combine_ms2: bool = False*) → None

Create an instance of IdPPdb inteface object

Parameters:

db_path : `str`

path to IdPP database file

read_only : `bool`, **default=False**

only allow fetch methods and do not maintain a rowid ledger, is faster to use if not modifying the database

enforce_idpp_ver : `bool`, **default=True**

raise a RuntimeError if version of this package does not match idpp_ver in the database

combine_ms2 : `bool`, **default=True**

combine MS2 spectra on inserting into the database

`IdPPdb.commit`

idpp.db.util.IdPPdb.commit(*self*) → None

commit changes to the databse (write to file)

`IdPPdb.close`

---

idpp.db.util.IdPPdb.close(*self, ignore_uncommitted_changes: bool = False*) → None

close connection to the database, release the ledger memory (if not read-only)

Parameters:

ignore_uncommitted_changes : `bool`, **default=False**

A RuntimeError is raised if the uncommitted changes flag is set (i.e., trying to close the database without committing the changes that have been made), this option ignores that check and no error is raised

Version Information and Change Log

`IdPPdb.update_version_info`

---

idpp.db.util.IdPPdb.update_version_info(*self*) → None

update version information stored in the database to be consistent with the current version of this idpp package (and its expected minimum Python version)

`IdPPdb.insert_change_log_entry`

---

idpp.db.util.IdPPdb.insert_change_log_entry(*self, author: str, notes: str*) → None

insert an entry into the change log with automatically generated timestamp, also updates the db_ver column of the VersionInfo table

Parameters:

author : `str`

who is responsible for the changes

notes : `str`

description of the changes

Sources

`IdPPdb.insert_src`

---

idpp.db.util.IdPPdb.insert_src(*self, src_name: str, src_ref: str, src_notes: str | None = None*) → int

insert an entry into the Sources table, return the rowid (checks if it already exists first and returns existing rowid if so)

Parameters:

src_name : `str`

name for the source

src_ref : `str`

link/reference info for the source

src_notes : `str`, **optional**

optional notes/metadata for the source, in JSON format

Returns:

src_id : `int`

identifier for the source that was just added

`IdPPdb.fetch_src_data`

idpp.db.util.IdPPdb.fetch_src_data(*self*, *n_rows: int*) → Iterator[List[Tuple[Any]]]

fetch data from Sources table yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

Yields:

src_id : `int`

source identifier

src_name : `str`

source name

src_ref : `str`

source reference info

src_notes : `str`

source notes (in JSON format)

Formulas

`IdPPdb.insert_form`

idpp.db.util.IdPPdb.insert_form(*self*, *form: str*) → int

insert an entry into the Formulas table, return the rowid (checks if it already exists first and returns existing rowid if so)

Always tries to convert to mzapy.isotopes.OrderedMolecularFormula first, if that fails return -1 placeholder

Parameters:

form : `str`

molecular formula string

Returns:

form_id : `int`

identifier for the formula that was just added (or already present)

`IdPPdb.fetch_form_data`

idpp.db.util.IdPPdb.fetch_form_data(*self*, *n_rows: int*) → Iterator[List[Tuple[Any]]]

fetch data from Formulas table yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

Yields:

form_id : `int`

formula identifier

form : `str`

molecular formula

Smiles

`IdPPdb.insert_smi`

idpp.db.util.IdPPdb.insert_smi(*self*, *smi: str*) → int

insert an entry into the Smiles table, return the rowid (checks if it already exists first and returns existing rowid if so)

Parameters:

smi : `str`

SMILES structure

Returns:

smi_id : `int`

identifier for the SMILES structure that was just added (or already present)

`IdPPdb.fetch_smi_data`

idpp.db.util.IdPPdb.fetch_smi_data(*self*, *n_rows*)

fetch data from Smiles table yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

Yields:

smi_id : `int`

SMILES identifier

smi : `str`

SMILES structure

InChIs

`IdPPdb.insert_inchi`

idpp.db.util.IdPPdb.insert_inchi(*self*, *inchi_key: str*, *inchi: str | None = None*) → int

insert an entry into the InChIs table, return the rowid (checks if it already exists first and returns existing rowid if so)

Parameters:

inchi_key : `str`

InChI key

inchi : `str`, **optional**

InChI structure string

Returns:

inchi_id : `int`

identifier for the compound that was just added (or already present)

`IdPPdb.fetch_inchi_data`

idpp.db.util.IdPPdb.fetch_inchi_data(*self*, *n_rows: int*) → Iterator[List[Tuple[Any]]]

fetch data from InChIs table yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

Yields:

inchi_id : `int`

InChI identifier

inchi_key : `str`

InChI key

inchi : `str`

full InChI structure

Compounds

`IdPPdb.insert_cmpd`

idpp.db.util.IdPPdb.insert_cmpd(*self*, *name: str*, *form_id: int = -1*, *smi_id: int = -1*, *inchi_id: int = -1*) → int

insert an entry into the Compounds table, return the rowid (checks if it already exists first and returns existing rowid if so) required form_id, smi_id, and inchi_id default to placeholder values (-1)

Parameters:

name : `str`

compound name

form_id : `int`, **default=-1**

Formulas row identifier

smi_id : `int`, **default=-1**

Smiles row identifier

inchi_id : `int`, **default=-1**

InChIs row identifier

Returns:

cmpd_id : `int`

identifier for the compound that was just added (or already present)

`IdPPdb.fetch_cmpd_data`

idpp.db.util.IdPPdb.fetch_cmpd_data(*self*, *n_rows: int*) → Iterator[List[Tuple[Any]]]

fetch data from Compounds table (joined to Formulas, Smiles, and InChIs table) yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

Yields:

cmpd_id : `int`

compound identifier

cmpd_name : `str`

compound name

form_id : `int`

Formulas identifier

form : `str`

molecular formula

smi_id : `int`

Smiles identifier

smi : `str`

SMILES structure

inchi_id : `int`

InChIs identifier

inchi_key : `str`

InChI key

inchi : `str`

full InChI structure

Adducts

`IdPPdb.insert_adduct`

`idpp.db.util.IdPPdb.insert_adduct`(*self, adduct: str, cmpd_id: int, mz: float, z: int*)→ int

insert an entry into the Compounds table, return the rowid (checks if it already exists first and returns existing rowid if so)

Parameters:

adduct : `str`

adduct type

cmpd_id : `int`

Compounds row identifier

z : `int`

adduct charge

mz : `float`

adduct m/z

Returns:

adduct_id : `int`

identifier for the adduct that was just added (or already present)

`IdPPdb.fetch_adduct_data`

---

idpp.db.util.IdPPdb.fetch_adduct_data(*self*, *n_rows: int*, *ionization: str = 'both'*)

fetch data from Adducts table (joined to Compounds and Formulas table) yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

ionization : `str`, **default='both'**

only yield adducts with a specified ionization state: +/pos/POS/etc. for positive, -/NEG/neg/etc. for negative, or both (the default) for both

Yields:

adduct_id : `int`

adduct identifier

adduct : `str`

adduct

adduct_z : `int`

adduct charge

adduct_mz : `real`

adduct m/z

cmpd_id : `int`

compound identifier

cmpd_name : `str`

compound name

`IdPPdb.fetch_adduct_data_extended`

---

idpp.db.util.IdPPdb.fetch_adduct_data_extended(*self*, *n_rows: int*, *ionization: str = 'both'*, *require_smi: bool = False*, *restrict_adducts: List[str] | None = None*)→ Iterator[List[Tuple[Any]]]

(extended) fetch data from Adducts table (joined to Compounds and Formulas table, and additionally include all of the compound data: formula, smiles, inchi) yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

ionization : `str`, **default='both'**

only yield adducts with a specified ionization state: +/pos/POS/etc. for positive, -/NEG/neg/etc. for negative, or both (the default) for both

require_smi : `bool`, **default=False**

add a constraint to the query to only return rows that have SMILES structures (for the compound -> cmpd_smi)

restrict_adducts : `list(str)`, **optional**

if included, restrict the results to only include the specified adducts

Yields:

adduct_id : `int`

adduct identifier

adduct : `str`

adduct

adduct_z : `int`

adduct charge

adduct_mz : `float`

adduct m/z

cmpd_id : `int`

compound identifier

cmpd_name : `str`

compound name

cmpd_form_id : `int`

(compound) Formulas identifier

cmpd_form : `str`

(compound) molecular formula

cmpd_smi_id : `int`

(compound) Smiles identifier

cmpd_smi : `str`

(compound) SMILES structure

cmpd_inchi_id : `int`

(compound) InChIs id

cmpd_inchi_key : `str`

(compound) InChI key

cmpd_inchi : `str`

(compound) InChI structure

CCSs

`IdPPdb.insert_ccs`

`idpp.db.util.IdPPdb.insert_ccs`(*self*, *ccs: float*, *adduct_id: int*, *src_id: int*)→ int

insert an entry into the CCSs table, return the rowid (DOES NOT check if already exists before adding, always adds new entry)

Parameters:

ccs : `float`

CCS value

adduct_id : `int`

Adducts row identifier

src_id : `int`

Sources row identifier

Returns:

ccs_id : `int`

identifier for the CCS value that was just added

`IdPPdb.fetch_ccs_data`

`idpp.db.util.IdPPdb.fetch_ccs_data`(*self, n_rows: int, ionization: str = 'both', select_sources: List[str | int] | None = None*)→ Iterator[List[Tuple[Any]]]

fetch data from CCSs table (joined to Adducts, Compounds, and Sources tables) yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

ionization : `str`, **default='both'**

only yield adducts with a specified ionization state: +/pos/POS/etc. for positive, -/NEG/neg/etc. for negative, or both (the default) for both

select_sources : `list(str or int)`, **optional**

specify a list of specific sources (by source name, *str*, or sorce identifier, *int*) to include, if *None* include all sources

Yields:

ccs_id : `int`

CCS identifier

ccs : `float`

CCS value

adduct_id : `int`

adduct identifier

adduct : `str`

adduct

adduct_z : `int`

adduct charge

adduct_mz : `real`

adduct m/z

cmpd_id : `int`

compound identifier

cmpd_name : `str`

compound name

src_id : `int`

source identifier

src_name : `str`

source name

RTs

`IdPPdb.insert_rt`

---

idpp.db.util.IdPPdb.insert_rt(*self*, *rt: float*, *adduct_id: int*, *src_id: int*)→ int

insert an entry into the RTs table, return the rowid (DOES NOT check if already exists before adding, always adds new entry)

Parameters:

rt : `float`

RT value

adduct_id : `int`

Adducts row identifier

src_id : `int`

Sources row identifier

Returns:

rt_id : `int`

identifier for the RT value that was just added

`IdPPdb.fetch_rt_data`

---

idpp.db.util.IdPPdb.fetch_rt_data(*self*, *n_rows: int*, *ionization: str = 'both'*, *select_sources: List[str | int] | None = None*)→ Iterator[List[Tuple[Any]]]

fetch data from RTs table (joined to Adducts, Compounds, and Sources tables) yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

ionization : `str`, **default='both'**

only yield adducts with a specified ionization state: +/pos/POS/etc. for positive, -/NEG/neg/etc. for negative, or both (the default) for both

select_sources : `list(str/int)`, **optional**

specify a list of specific sources (by source name, *str*, or sorce identifier, *int*) to include, if *None* include all sources

Yields:

rt_id : `int`

RT identifier

rt : `float`

RT value

adduct_id : `int`

adduct identifier

adduct : `str`

adduct

adduct_z : `int`

adduct charge

adduct_mz : `real`

adduct m/z

cmpd_id : `int`

compound identifier

cmpd_name : `str`

compound name

src_id : `int`

source identifier

src_name : `str`

source name

MS2Spectra

`IdPPdb.insert_ms2`

---

idpp.db.util.IdPPdb.insert_ms2(*self, ms2_mz: ndarray[Any, dtype[_ScalarType_co]], ms2_i: ndarray[Any, dtype[_ScalarType_co]], adduct_id: int, src_id: int, ms2_ce: int | None = None, max_n_fragments: int = 256*)→ int

insert an entry into the MS2Spectra table, return the rowid

If there is already a spectrum for the same adduct_id and self.combine_ms2 is set, then combine the input spectrum with that one and update, otherwise add a new entry

Parameters:

ms2_mz : `numpy.ndarray`

ms2_i : `numpy.ndarray`

m/z and intensity components of MS/MS spectrum (as numpy arrays, of floats)

adduct_id : `int`

Adducts row identifier

src_id : `int`

Sources row identifier

ms2_ce : `int`, **optional**

specify collision energy (voltage as int) for the spectrum

max_n_fragments : `int`, **default=256**

only retain the top N fragments in each spectrum

Returns:

ms2_id : `int`

identifier for the MS/MS spectrum that was just added

`IdPPdb.fetch_ms2_data`

---

idpp.db.util.IdPPdb.fetch_ms2_data(*self, n_rows: int, ionization: str = 'both', select_sources: List[str | int] | None = None*)→ Iterator[List[Tuple[Any]]]

fetch data from MS2Spectra, MS2Fragments, MS2Sources tables (joined to Adducts and Compounds tables) yields one batch of rows at a time as list of tuples

Parameters:

n_rows : `int`

number of rows to yield in each batch

ionization : `str`**, default='both'**

only yield adducts with a specified ionization state: +/pos/POS/etc. for positive, -/NEG/neg/etc. for negative, or both (the default) for both

select_sources : `list(str/int)`**, optional**

specify a list of specific sources (by source name, *str*, or sorce identifier, *int*) to include, if *None* include all sources

Yields:

ms2_id : `int`

MS2 spectrum identifier

ms2_mz : `numpy.ndarray(float)`

m/z component of MS2 spectrum as an array

ms2_i : `numpy.ndarray(float)`

intensity component of MS2 spectrum as an array

ms2_ce : `str`

collision energy of the combined spectra

adduct_id : `int`

adduct identifier

adduct : `str`

adduct

adduct_z : `int`

adduct charge

adduct_mz : `real`

adduct m/z

cmpd_id : `int`

compound identifier

cmpd_name : `str`

compound name

src_ids : `list(int)`

source identifiers

Miscellaneous

`IdPPdb.insert_ext_id`

idpp.db.util.IdPPdb.insert_ext_id(*self, cmpd_id: int, src_id: int, ext_id: str*)→ None

insert an entry into the ExternalIDs table

does not return a rowID since this table just associates external identifiers with existing compounds/sources

Parameters:

cmpd_id : `int`

compound identifier

src_id : `int`

source identifier

ext_id : `str`

external identifier