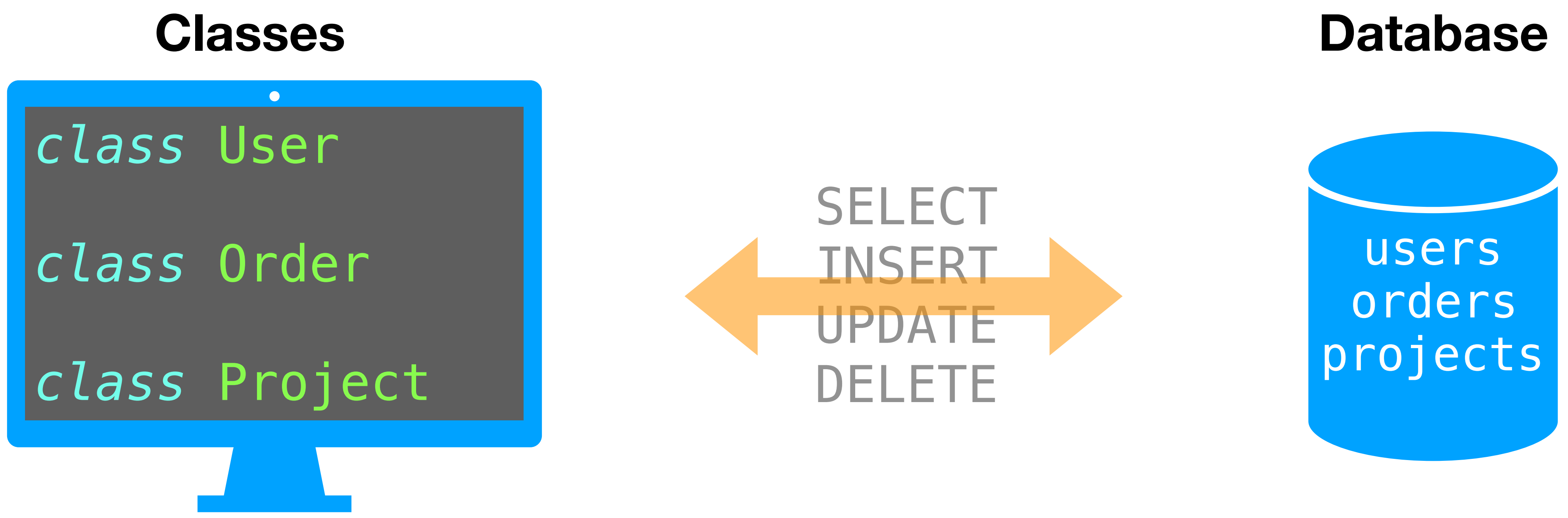
JARDIN

The easy and lightweight ORM for Python

Emmanuel Turlay
Instacart
emmanuel@turlay.net

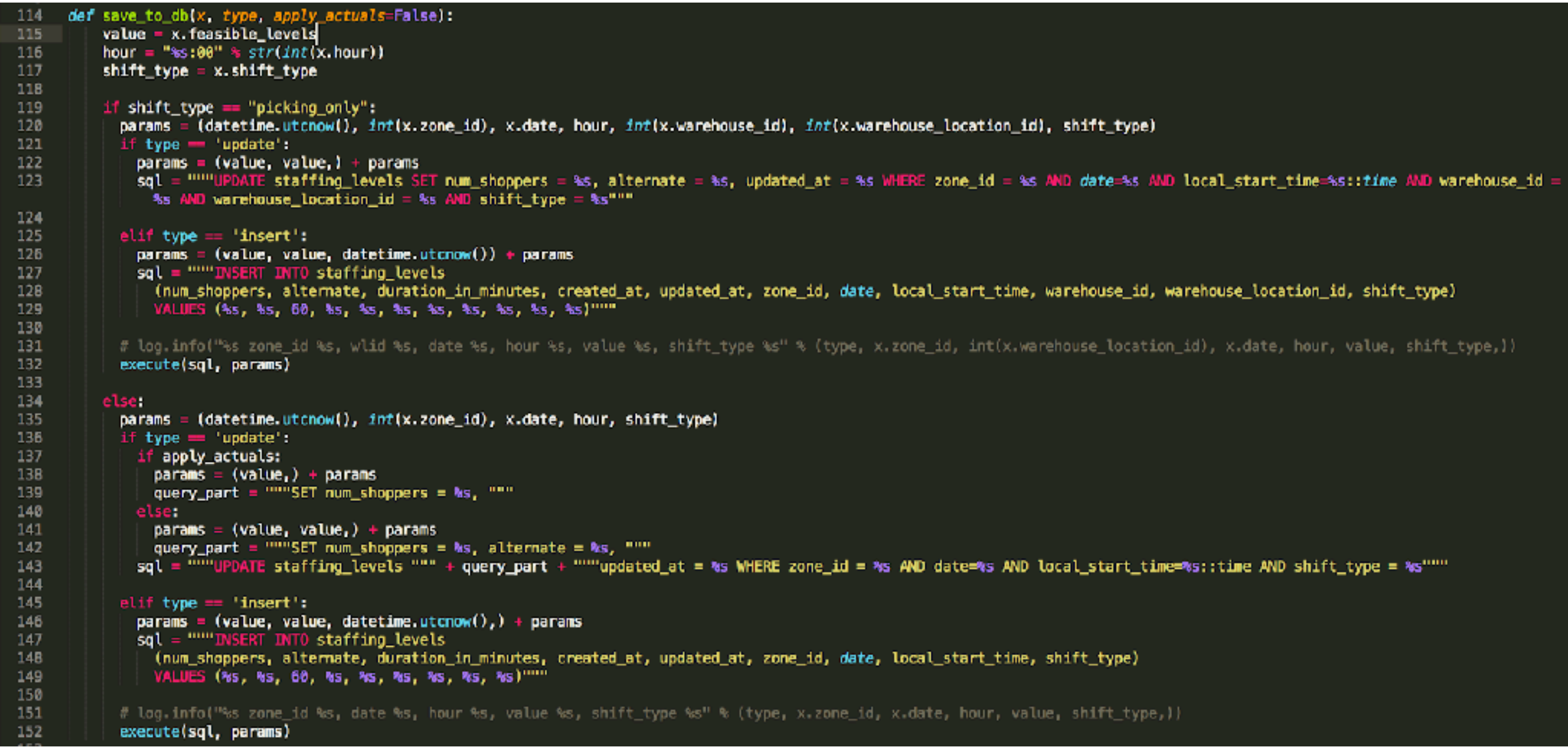
What’s an ORM?

Object Relational Mapping



SQLAlchemy, ActiveRecord, DataMapper…

Why an ORM?



```python
def save_to_db(x, type, apply_actuals=False):
    value = x.feasible_levels
    hour = "%s:00" % str(int(x.hour))
    shift_type = x.shift_type

    if shift_type == "picking_only":
      params = (datetime.utcnow(), int(x.zone_id), x.date, hour, int(x.warehouse_id), int(x.warehouse_location_id), shift_type)
      if type == 'update':
        params = (value, value,) + params
        sql = """UPDATE staffing_levels SET num_shoppers = %s, alternate = %s, updated_at = %s WHERE zone_id = %s AND date=%s AND local_start_time=%s::time AND warehouse_id =
          %s AND warehouse_location_id = %s AND shift_type = %s"""

      elif type == 'insert':
        params = (value, value, datetime.utcnow()) + params
        sql = """INSERT INTO staffing_levels
          (num_shoppers, alternate, duration_in_minutes, created_at, updated_at, zone_id, date, local_start_time, warehouse_id, warehouse_location_id, shift_type)
          VALUES (%s, %s, 60, %s, %s, %s, %s, %s, %s, %s, %s)"""

      # log.info("%s zone_id %s, wlid %s, date %s, hour %s, value %s, shift_type %s" % (type, x.zone_id, int(x.warehouse_location_id), x.date, hour, value, shift_type,))
      execute(sql, params)

    else:
      params = (datetime.utcnow(), int(x.zone_id), x.date, hour, shift_type)
      if type == 'update':
        if apply_actuals:
          params = (value,) + params
          query_part = """SET num_shoppers = %s, """
        else:
          params = (value, value,) + params
          query_part = """SET num_shoppers = %s, alternate = %s, """
        sql = """UPDATE staffing_levels """ + query_part + """updated_at = %s WHERE zone_id = %s AND date=%s AND local_start_time=%s::time AND shift_type = %s"""

      elif type == 'insert':
        params = (value, value, datetime.utcnow(),) + params
        sql = """INSERT INTO staffing_levels
          (num_shoppers, alternate, duration_in_minutes, created_at, updated_at, zone_id, date, local_start_time, shift_type)
          VALUES (%s, %s, 60, %s, %s, %s, %s, %s, %s)"""

      # log.info("%s zone_id %s, date %s, hour %s, value %s, shift_type %s" % (type, x.zone_id, x.date, hour, value, shift_type,))
      execute(sql, params)
```

Abstraction layer

Database connections

Interact with database objects

Error handling

Model-level business logic

JARDIN

jardin *(noun, french)* – garden, yard, grove

SQLite MySQL PostgreSQL amazon REDSHIFT snowflake

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Configure

```
# jardin_conf.py

import logging

DATABASES = {
    'db1_master': 'postgres://user:pass@host:port/database_w',
    'db1_replica': 'postgres://user:pass@host:port/database_r',
    'db2': {
        ‘scheme': 'mysql',
        'username': 'user',
        …
    }
}

LOG_LEVEL = logging.DEBUG

WATERMARK = 'MyCoolApp'
```

```
$ export JARDIN_CONF=/path/to/jardin_conf.py
```

Declare

```python
# db_models.py

import jardin

class Db1AbstractModel(jardin.Model):
    db_names = {
        'master': 'db1_master',
        'replica': 'db1_replica'
    }

class Db2AbstractModel(jardin.Model):
    db_names = {
        'master': 'db2',
        'replica': 'db2'
    }

class User(Db1AbstractModel): pass

class Order(Db2AbstractModel): pass
```

```
>>> from db_models import User

>>> User.insert(values={‘name’: ‘jardin’})
DEBUG:jardin:INSERT INTO users VALUES (…) /* MyCoolApp */
User(id=2, name=‘jardin’, created_at=‘…’, updated_at=‘…’, …)

>>> User.select(where={‘active’: False})
DEBUG:jardin:SELECT * FROM users WHERE … /* MyCoolApp */
     id   name         active
0    2    ‘jardin’     False
1    1    ‘sqlalchemy’ False

>>> User.update(values={‘active’: True}, where={‘name’: ‘jardin’})
DEBUG:jardin:UPDATE users SET … WHERE … /* MyCoolApp */

>>> user = User.find(1)
DEBUG:jardin:SELECT * FROM users WHERE … /* MyCoolApp */

>>> user.active = False

>>> user.save()
DEBUG:jardin:UPDATE users SET … WHERE … /* MyCoolApp */

>>> User.delete(where={‘active’: False})
DEBUG:jardin:DELETE FROM users WHERE … /* MyCoolApp */
```

```python
# path/to/file.py

from datetime import datetime, timedelta
from db_models import User, UserSettings
from jardin.comparators import gt


users = User.select(
    select={
          ‘user_id’: ‘id’,
          ‘username’: ‘name’,
          ‘newsletter’: ‘us.newsletter’
      },
     where={
         ‘active’: True,
         ‘created_at’: gt(datetime.utcnow() - timedelta(days=30)),
         ‘deleted_at’: None
     },
     inner_join=[UserSettings]
     order={‘created_at’: ‘DESC’},
     limit=10
)
```

```
DEBUG:jardin:(‘SELECT id AS user_id, name AS username, us.newsletter AS
newsletter FROM users u INNER JOIN user_settings us ON us.user_id =
u.id WHERE active IS TRUE AND created_at > %(created_at)s AND
deleted_at IS NULL ORDER BY created_at DESC LIMIT 10; /* MyCoolApp | /
path/to/file.py:8 */‘, OrderedDict((‘created_at’, ‘2018-04-09
19:00:00’)))
```

```
import jardin


dataframe = jardin.query(
    sql='SELECT * FROM orders WHERE abc = %(abc)s LIMIT 10;'
    params={'abc': 123},
    db='db1_replica'
)

dataframe = jardin.query(
    filename='/path/to/extract.sql'
    params={'abc': 123},
    db='db1_replica'
)
```

```python
import pandas as pd
from db_models import User

df = pd.DataFrame(…)

with User.transaction():
    User.delete(where={‘id’: df.id})
    User.insert(values=df)
```

Python 2.7+ and 3.5+

Pandas-dataframe integration

PostgreSQL, MySQL, SQLite, AWS Redshift and Snowflake

Multiple databases with master/replica split

Transactions

Connection drop recovery

Single-statement dataframe insertion

À la ActiveRecord query scopes

created_at, updated_at and soft-deletes out-of-the-box support

```
pip install jardin
```

instacart.github.io/jardin