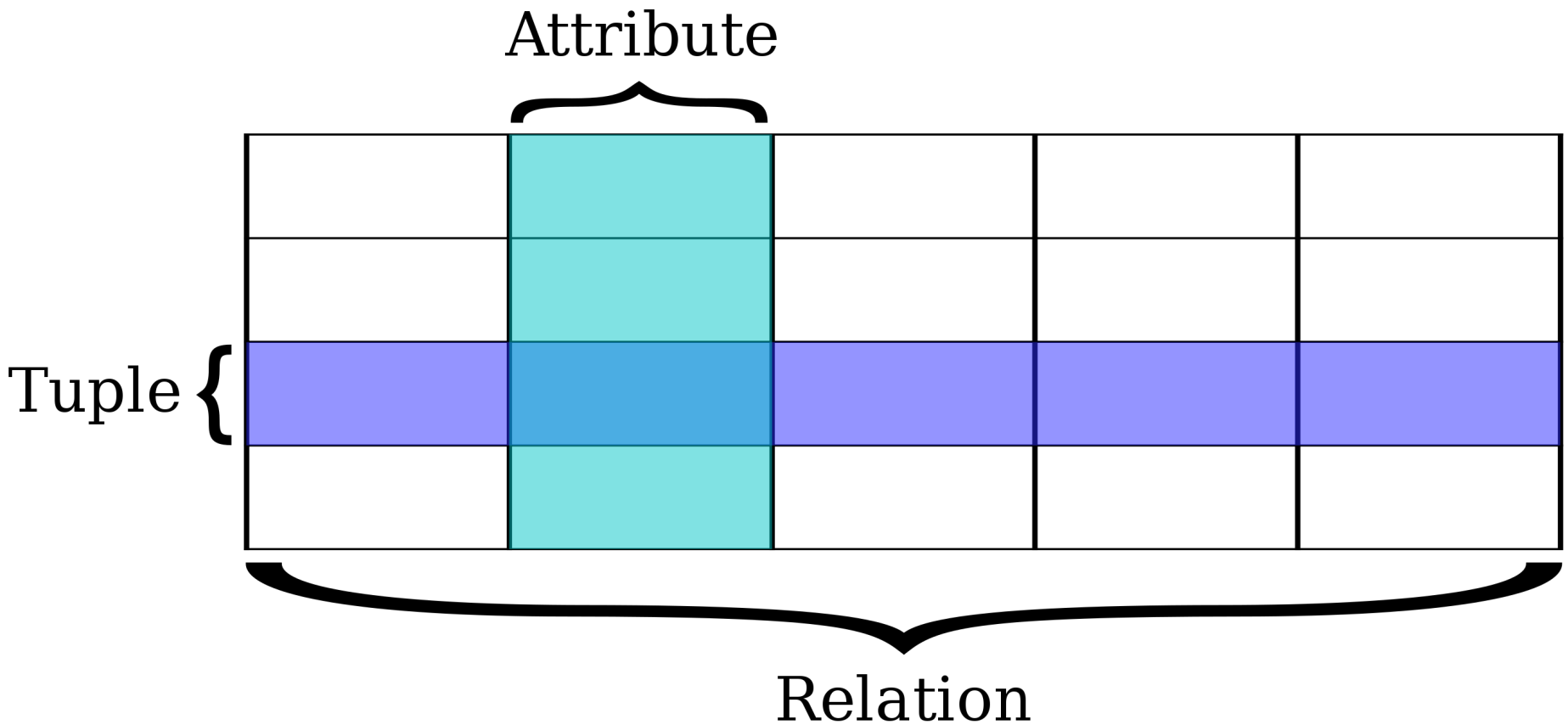
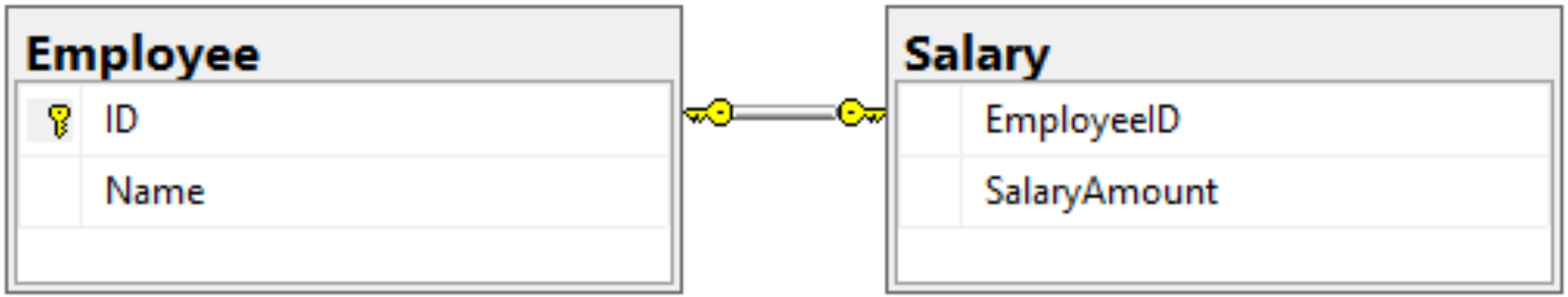
Реляционные базы данных. Работа с базами данных из Python

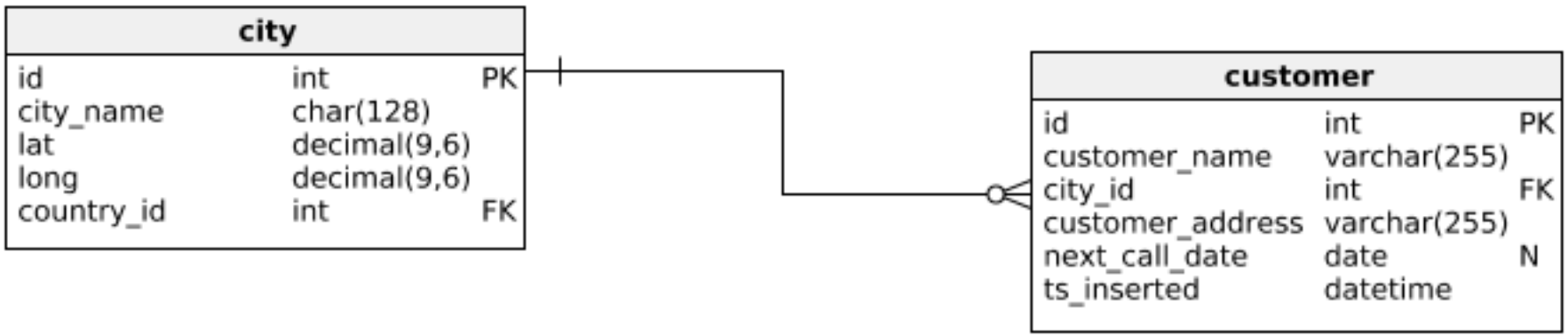
Таблицы



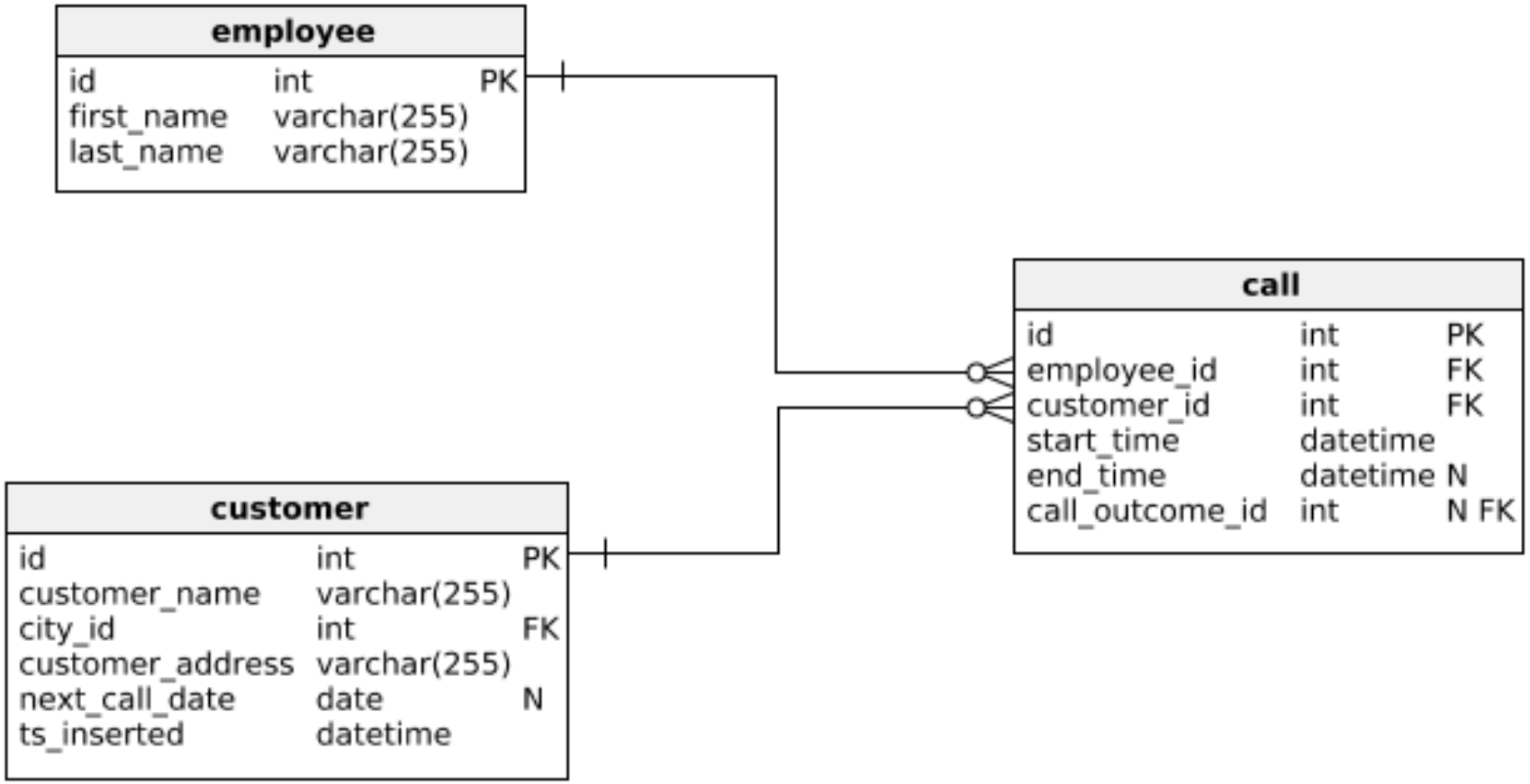
Отношение 1:1 (one-to-one)



Отношение 1:N (one-to-many)



Отношение N:N (many-to-many)



Primary key и foreign key

0. Заполняем базу данными

1. Выполните запрос к базе данных:

```
create table users
(
    id uuid primary key, -- уникальный id пользователя
    created timestamp -- время регистрации пользователя
);

create table orders
(
    id uuid primary key, -- уникальный id заказа
    user_id uuid references users(id), -- id пользователя, совершившего заказ, из таблицы users
    city varchar, -- город совершения заказа
    created timestamp -- время заказа
);
```

А ещё в таких СУБД есть

- Индексы
- Процедуры
- Триггеры

Python + SQL (postgresql)

- Есть много библиотек, они все похожи
- Для Python подойдёт psycopg2

Как общаться с БД?

```
import psycopg2
conn = psycopg2.connect(dbname='database',
user='db_user', password='mypassword',
host='localhost’)
cursor = conn.cursor()
```

А дальше выполняем запросы и получаем итерируемый объект с кортежами.

Не забываем закрывать соединения (`cursor.close()`, `conn.close()`).

Обращение к результатам запроса

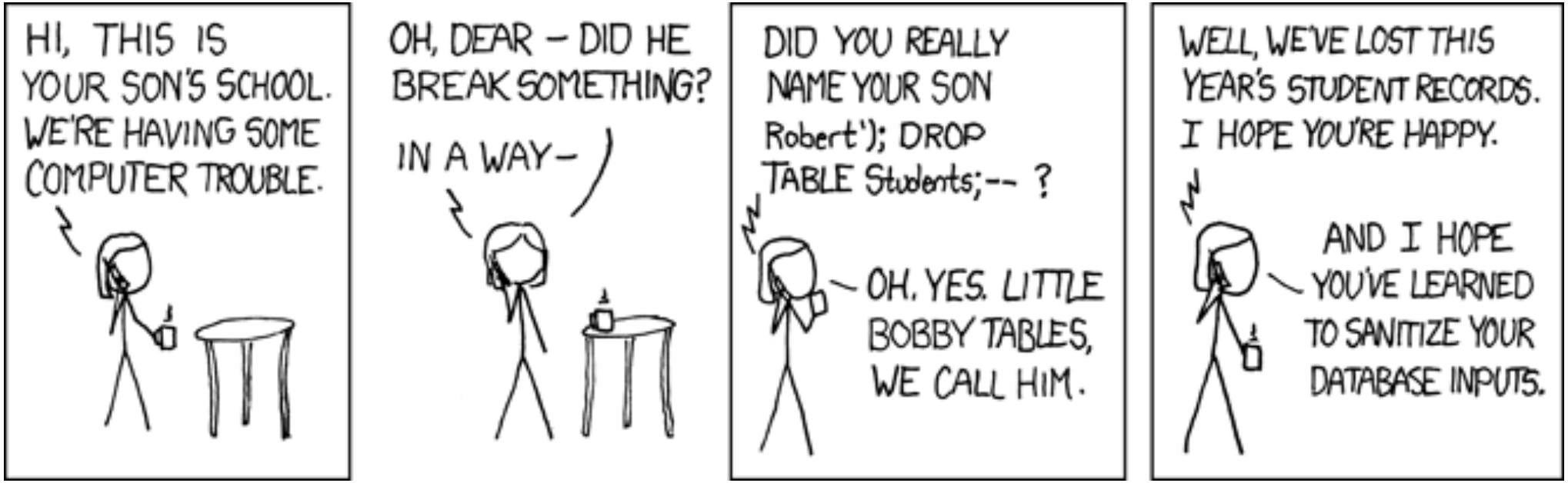
```
cursor.execute('SELECT * FROM orders LIMIT 10')

cursor.fetchall() # список всех строк
cursor.fetchone() # одна строка
cursor.fetchmany(size=5)
```

А можно просто `for row in cursor`!

Ещё про psycopg2

- Умеет возвращать словари вместо кортежей – см. DictCursor
- Умеет управлять транзакциями – см. conn.commit(), conn.rollback(), атрибут conn.autocommit
- Умеет подставлять параметры в строку запроса

SQL-инъекции



SQL-инъекции

”SELECT * FROM Students WHERE name = ’%s’” % student_name

ORM

- Прослойка между базой данных и представлением данных в виде Python-объектов
- Помогает ускорить разработку и уменьшить количество дефектов, в том числе связанных с безопасностью приложения
- SQLAlchemy – одна из самых популярных ORM-систем для Python

Модель данных

```
from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite:///:memory:', echo=True)

Base = declarative_base()
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column(String)

    def __init__(self, name, fullname, password):
        self.name = name
        self.fullname = fullname
        self.password = password
    def __repr__(self):
        return "<User('%s','%s', '%s')>" % (self.name, self.fullname, self.password)

# Создание таблицы
Base.metadata.create_all(engine)
```

Фабрика сессий

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine) # фабрика
session = Session() # сессия
```

Работа с объектами

vasiaUser = User("vasia", "Vasiliy Pypkin", "vasia2000")
session.add(vasiaUser) # пока что строка не сохранена в БД

ourUser = session.query(User).filter_by(name="vasia").first() # вернет созданного пользователя

session.commit()

session.rollback()

Запросы к данным

- session.query(MyClass).filter(MyClass.name == 'some name', MyClass.id > 5)
- Поддерживается практически всё, что есть в обычном SQL
- https://docs.sqlalchemy.org/en/13/orm/query.html