

# Базы Данных: не только SQL

Орлов Илья  
Нижний Новгород  
2019г.



# Агенда

- Способы ухода от SQL
- Недалеко ушли: работа с БД через ORM
- Примеры ORM в Python: sqlalchemy
- Ограничения реляционной модели
- Теорема CAP: согласованность или доступность
- Уходим дальше: NoSQL-решения
- Принципиальные различия SQL и NoSQL
- SQL vs NoSQL: как сделать выбор



# Реляционная БД

Сотрудники					
Таб. номер	ФИО	Должность	Премия	Логин	Пароль
1	Иванов И.И.	инженер	30000	ivanovi	ivanov123
2	Петров П.П.	старший инженер	50000	petrovp	p1e2t3
3	Сидоров С.С.	менеджер проекта	30000	sidorovs	zayka88
4	Егоров Е.Е.	инженер	20000	egorove	qwerty
5	Новый Н.Н.	инженер	20000	novyin	a1111

Проекты	
ID	Название
1	Важный
2	Срочный
3	Скучный

Должность-Оклад	
Название	Оклад
инженер	50000
старший инженер	51000
ведущий инженер	70000
менеджер проекта	100000

Сотрудник-Проект	
Таб. номер	ID проекта
1	1
2	1
2	2
3	2
4	2



# SQL-запрос

```
# Вывод информации для менеджера проекта
# Соединяем таблицы Employees, PositionSalary, EmployeeProject
def show_manager_info(conn, project_id):
    cur = conn.cursor()
    cur.execute("SELECT E.Id, E.Name, P.Salary + E.Bonus As Pay"
               " FROM Employees AS E, PositionSalary AS P, "
               "      EmployeeProject AS EP"
               " WHERE E.Position = P.Position"
               " AND E.Id = EP.EmployeeId"
               " AND EP.ProjectId = :project_id",
               {'project_id': project_id})
    print("Информация для менеджера:")
    for row in cur.fetchall():
        print(dict(row))
```

Employee					
Id	Name	Position	Bonus	Login	Password
1	Иванов И.И.	инженер	30000	ivanovi	ivanov123
2	Петров П.П.	старший инженер	50000	petrovp	p1e2t3
3	Сидоров С.С.	менеджер проекта	30000	sidorovs	zayka88

PositionSalary	
Position	Salary
инженер	50000
старший инженер	51000
менеджер проекта	100000

Id	Name	Position	Salary	Bonus	ProjectID
1	Иванов И.И.	инженер	50000	30000	1
2	Петров П.П.	старший инженер	51000	50000	1
2	Петров П.П.	старший инженер	51000	50000	2
3	Сидоров С.С.	менеджер проекта	100000	30000	2

EmployeeProject	
EmployeeID	ProjectID
1	1
2	1
2	2
3	2





# Способы ухода от SQL

```
# Вывод информации для менеджера проекта
# Соединяем таблицы Employees, PositionSalary, EmployeeProject
def show_manager_info(conn, project_id):
    cur = conn.cursor()
    cur.execute("SELECT E.Id, E.Name, P.Salary + E.Bonus As Pay"
               " FROM Employees AS E, PositionSalary AS P, "
               "      EmployeeProject AS EP"
               " WHERE E.Position = P.Position"
               " AND E.Id = EP.EmployeeId"
               " AND EP.ProjectId = :project_id",
               {'project_id': project_id})
    print("Информация для менеджера:")
    for row in cur.fetchall():
        print(dict(row))
```

Отказываемся от  
чистого SQL

Employee					
Id	Name	Position	Bonus	Login	Password
1	Иванова И.И.	инженер	30000	ivanov	ivanov123
2	Петрова П.П.	старший инженер	50000	petrov	petrov3
3	Сидорова С.С.	менеджер проекта	100000	sidorov	sidorov8

PositionSalary	
Position	Salary
инженер	30000
старший инженер	50000
менеджер проекта	100000

Id	Name	Position	Salary	Bonus	ProjectId
1	Иванова И.И.	инженер	30000	30000	1
2	Петрова П.П.	старший инженер	50000	50000	1
2	Петрова П.П.	старший инженер	50000	50000	2
3	Сидорова С.С.	менеджер проекта	100000	30000	2

EmployeeProject	
EmployeeId	ProjectId
1	1
2	1
2	2
3	2

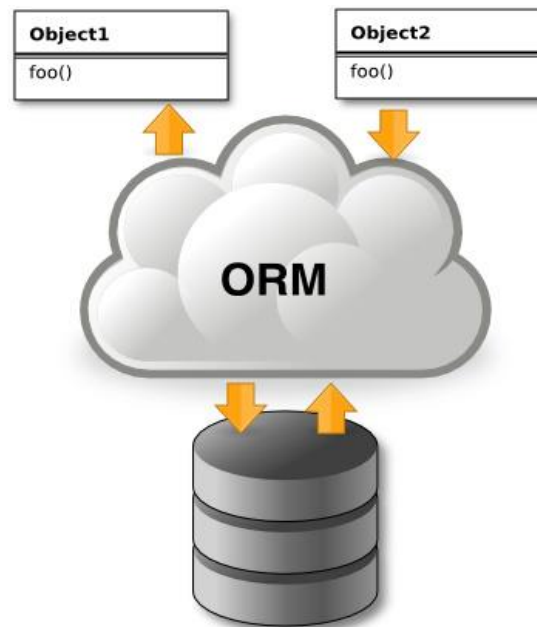
Отказываемся от  
реляционной модели



# ORM

ORM (Object-Relational Mapping – объектно-реляционное преобразование)  
- технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» (Wiki).

Известные ORM в Python: SQLAlchemy, DjangoORM, peewee, PonyORM, SQLAlchemy, Storm, quick\_orm и другие.





# Классы для таблиц

```
class PositionSalary(Base):  
    __tablename__ = 'position_salary'  
    position = Column(String, primary_key=True)  
    salary = Column(Integer, nullable=False)
```

PositionSalary	
Position	Salary
инженер	50000
старший инженер	51000
менеджер проекта	100000

```
class Employee(Base):  
    __tablename__ = 'employee'  
    id = Column(Integer, primary_key=True)  
    name = Column(String, nullable=False)  
    position = Column(String,  
        ForeignKey('position_salary.position'),  
        nullable=False)  
    bonus = Column(Integer, default=0)  
    login = Column(String, nullable=False, unique=True)  
    password = Column(String, nullable=False)
```

Employee					
Id	Name	Position	Bonus	Login	Password
1	Иванов И.И.	инженер	30000	ivanovi	ivanov123
2	Петров П.П.	старший инженер	50000	petrovp	p1e2t3
3	Сидоров С.С.	менеджер проекта	30000	sidorovs	zayka88



# Почему ORM – это выгодно

## Возможности

- Сокращение кода
- Единая парадигма программирования
- Независимость от диалекта SQL



## Profit

- Ускорение разработки
- Простота понимания всего кода
- Универсальность методов отладки
- Кросс-СУБД код

## Недостатки

- Медленнее чистого SQL
- Требуется больше памяти
- Уступает в полноте и гибкости



## Однако

Программист при необходимости может сам задать код SQL-запросов, который будет использоваться при тех или иных действиях





# SQLAlchemy

**SQLAlchemy** – библиотека Python для работы с базами данных по технологии ORM. Она позволяет ассоциировать пользовательские классы Python с таблицами баз данных, и объекты этих классов со строками в соответствующих таблицах.

**SQLAlchemy** предоставляет:

- ORM уровень;
- обобщенный API для работы с различными СУБД;
- интерфейс, достаточно близкий по полноте к чистому SQL;
- возможность использования прямых SQL-запросов.



# SQLAlchemy vs DB-API

# Добавление нового сотрудника с привязкой к проекту средствами SQLAlchemy

```
def add_new_employee_to_project(self, name, position, bonus, login, password, project_id):
    e = Employee(name=name, position=position, bonus=bonus, login=login, password=password)
    self._session.add(e)
    self._session.commit()
    self._session.add(EmployeeProject(employee_id=e.id, project_id=project_id))
    self._session.commit()
```

# Добавление нового сотрудника с привязкой к проекту средствами DB-API

```
def add_new_employee_to_project(conn, name, position, bonus, login, pwd, project_id):
    cur = conn.cursor()
    cur.execute("INSERT INTO Employees (Name, Position, Bonus, Login, Password) "
               " VALUES (:name, :position, :bonus, :login, :pwd)",
               {'name': name, 'position': position, 'bonus': bonus,
                'login': login, 'pwd': pwd})
    conn.commit()
    cur.execute("SELECT E.Id FROM Employees AS E "
               "WHERE E.Login = :login AND E.Password = :pwd",
               {'login': login, 'pwd': pwd})
    employee_id = dict(cur.fetchone())['Id']
    cur.execute("INSERT INTO EmployeeProject (EmployeeId, ProjectId) "
               " VALUES (:employeeId, :projectId)",
               {'employeeId': employee_id, 'projectId': project_id})
    conn.commit()
```



# Описание классов

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

```
Base = declarative_base()
```

```
class Employee(Base):
    __tablename__ = 'employee'
    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    position = Column(String, ForeignKey('position_salary.position'), nullable=False)
    bonus = Column(Integer, default=0)
    login = Column(String, nullable=False, unique=True)
    password = Column(String, nullable=False)
```

```
class Project(Base):
    __tablename__ = 'project'
    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
```

```
class PositionSalary(Base):
    __tablename__ = 'position_salary'
    position = Column(String, primary_key=True)
    salary = Column(Integer, nullable=False)
```

```
class EmployeeProject(Base):
    __tablename__ = 'employee_project'
    employee_id = Column(Integer, ForeignKey('employee.id'), primary_key=True)
    project_id = Column(Integer, ForeignKey('project.id'), primary_key=True)
```



# Подключение к БД

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

class DBClient:

    def __init__(self, dbtype='sqlite', dbname='/tmp.db', username=None, password=None):
        self._engine = self._get_engine(dbtype, dbname, username, password)

    def __enter__(self):
        self._session = sessionmaker(bind=self._engine)()
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        self._session.close_all()

    @staticmethod
    def _get_engine(dbtype, dbname, username=None, password=None):
        if username:
            if password:
                login = '{u}:{p}'.format(u=username, p=password)
            else:
                login = username
            dbstr = '{l}@{db}'.format(l=login, db=dbname)
        else:
            dbstr = dbname
        engine = create_engine('{}/{}/{}/'.format(dbtype, dbstr))
        return engine
```



# Конфигурирование БД

```
def create_schema(self):  
    # Создаем схему  
    Base.metadata.create_all(self._engine)  
  
def delete_schema(self):  
    # Удаляем схему  
    Base.metadata.drop_all(self._engine)
```





# Добавление записей

```
# Добавление записей в таблицу ДолжностьОклад
def insert_position(self, position, salary):
    self._session.add(PositionSalary(position=position, salary=salary))
    self._session.commit()

# Добавление записей в таблицу Проекты
def insert_project(self, name):
    p = Project(name=name)
    self._session.add(p)
    self._session.commit()
    return p.id

# Добавление записей в таблицу Сотрудники
def insert_employee(self, name, position, bonus, login, password):
    e = Employee(name=name, position=position, bonus=bonus,
                  login=login, password=password)
    self._session.add(e)
    self._session.commit()
    return e.id

# Добавление записей в таблицу СотрудникиПроекты
def add_employee_to_project(self, employee_id, project_id):
    self._session.add(EmployeeProject(employee_id=employee_id,
                                       project_id=project_id))

    self._session.commit()
```



# Создание БД

```
db_type = "sqlite"
db_name = "ogo.db"
db_exists = os.path.exists(db_name)

if not db_exists:
    with DBClient(db_type, db_name) as dbc:
        dbc.create_schema()

        dbc.insert_position("инженер", 50000)
        dbc.insert_position("старший инженер", 51000)
        dbc.insert_position("менеджер проекта", 100000)

        pid = dbc.insert_project("Важный")
        eid = dbc.insert_employee("Иванов И.И.", "инженер", 30000, "ivanovi", "ivanov123")
        dbc.add_employee_to_project(eid, pid)

        eid = dbc.insert_employee("Петров П.П.", "старший инженер", 50000, "petrovp", "ple2t3")
        dbc.add_employee_to_project(eid, pid)

        pid = dbc.insert_project("Срочный")
        dbc.add_employee_to_project(eid, pid)

        eid = dbc.insert_employee("Сидоров С.С.", "менеджер проекта", 30000,
                                   "sidorovs", "zayka88")
        dbc.add_employee_to_project(eid, pid)
```



# Чтение данных

# Проверка наличия пользователя в базе данных с указанным логином/паролем

```
def authentication(self, login, password):
    try:
        res = self._session.query(Employee.id, Employee.name,
                                    Employee.position, EmployeeProject.project_id).\
            join(EmployeeProject, EmployeeProject.employee_id == Employee.id). \
            filter(and_(Employee.login == login,
                        Employee.password == password)).\
            one()
        return res
    except MultipleResultsFound:
        print("Multiple Results Found")
    except NoResultFound:
        print("No Result Found")
    return None
```

# Проверка наличия указанного сотрудника в указанном проекте

```
def is_employee_in_project(self, employee_id, project_id):
    try:
        res = self._session.query(EmployeeProject.project_id).\
            filter(and_(EmployeeProject.employee_id == employee_id,
                        EmployeeProject.project_id == project_id)).\
            one()
        return True
    except MultipleResultsFound:
        print("Multiple Results Found")
    except NoResultFound:
        print("No Result Found")
    return None
```



# Чтение данных

```
# Вывод информации по сотруднику (соединяем таблицы Employees, PositionSalary)
def show_employee_info(self, employee_id):
    res = self._session.query(Employee.id, Employee.name,
                              (PositionSalary.salary +
                               Employee.bonus).label("Pay")).\
        filter(and_(Employee.position == PositionSalary.position,
                     Employee.id == employee_id)).\
        all()
    print("Информация для сотрудника:")
    for row in res:
        print(row)
    return res

# Вывод информации по проекту (соединяем таблицы Employees, PositionSalary, EmployeeProject)
def show_manager_info(self, project_id):
    res = self._session.query(Employee.id, Employee.name,
                              (PositionSalary.salary +
                               Employee.bonus).label("Pay")). \
        filter(and_(Employee.position == PositionSalary.position,
                     Employee.id == EmployeeProject.employee_id,
                     EmployeeProject.project_id == project_id)). \
        all()
    print("Информация для менеджера:")
    for row in res:
        print(row)
    return res
```



# Изменение данных

# Изменение премии сотрудника

```
def update_employee_bonus(self, employee_id, new_bonus):  
    e = self._session.query(Employee).get(employee_id)  
    if e:  
        e.bonus = new_bonus  
        self._session.add(e)  
        self._session.commit()
```

# Удаление сотрудника из проекта (но не из базы данных)

```
def delete_employee_from_project(self, employee_id, project_id):  
    ep = self._session.query(EmployeeProject).get((employee_id, project_id))  
    if ep:  
        self._session.delete(ep)  
        self._session.commit()
```





# Тестовый пример

```
with DBClient(db_type, db_name) as dbc:
    res = dbc.authentication(login, pwd)
    if res:
        user = res._asdict()
        print("Здравствуйте, {}".format(user['name']))
        if user['position'] == "менеджер проекта":
            dbc.show_manager_info(user['project_id'])

    id_upd = int(input("Изменение премии. ID сотрудника (0 - отмена): "))
    if id_upd:
        if (id_upd != user['id'] and
            dbc.is_employee_in_project(id_upd, user['project_id'])):
            new_bonus = input("Новая премия: ")
            dbc.update_employee_bonus(id_upd, new_bonus)
            dbc.show_manager_info(user['project_id'])
        else:
            print("Невозможно изменить премию для данного сотрудника")

    id_del = int(input("Удаление сотрудника. ID сотрудника (0 - отмена): "))
    if id_del:
        if id_del != user['id']:
            dbc.delete_employee_from_project(id_del, user['project_id'])
            dbc.show_manager_info(user['project_id'])
        else:
            print("Невозможно удалить данного сотрудника из проекта")
    else:
        dbc.show_employee_info(user['id'])
    else:
        print("Доступ запрещен")
```



# Тестовый вывод

```

Логин: sidorovs
Пароль: zayka88
Здравствуйте, Сидоров С.С.
Информация для менеджера:
(2, 'Петров П.П.', 101000)
(3, 'Сидоров С.С.', 130000)
Изменение премии. ID сотрудника (0 - отмена): 2
Новая премия: 60000
Информация для менеджера:
(2, 'Петров П.П.', 111000)
(3, 'Сидоров С.С.', 130000)
Удаление сотрудника. ID сотрудника (0 - отмена): 0
    
```

```

Логин: sidorovs
Пароль: zayka88
Здравствуйте, Сидоров С.С.
Информация для менеджера:
(2, 'Петров П.П.', 111000)
(3, 'Сидоров С.С.', 130000)
Изменение премии. ID сотрудника (0 - отмена): 0
Удаление сотрудника. ID сотрудника (0 - отмена): 2
Информация для менеджера:
(3, 'Сидоров С.С.', 130000)
    
```



# Смена предметной области

## Сотрудники

Таб. номер  
ФИО  
Должность  
Премия  
Логин  
Пароль



## Пользователи

ID  
ФИО  
Роль  
Рейтинг  
Логин  
Пароль

## Проекты

ID  
Название



## Группы

ID  
Название

## СотрудникиПроекты

Таб. номер сотрудника  
ID проекта



## ПользователиГруппы

ID пользователя  
ID группы



# Проектируем социальную сеть

- Данных будет много (vk.com – более 460 млн. пользователей)
- Окончательной схемы данных нет (сами сущности и их атрибуты еще будут неоднократно добавляться и удаляться)

Пользователи							
ID	ФИО	Роль	Рейтинг	Логин	Пароль	Статус	Аватар
1	Иванов И.И.	участник	300	ivanovi	ivanov123		
2	Петров П.П.	участник	250	petrovp	p1e2t3		
3	Сидоров С.С.	модератор	1000	sidorovs	zayka88		file1.jpg
...							
1000000000	Новый Н.Н.	участник	0	novyin	a1111	offline	

Группы			
ID	Название	Тип	Описание
1	Киноманы		
2	WoT		
...			
100000	Йога	закрытая	Для любителей йоги

ПользовательГруппа	
ID пользователя	ID группы
1	1
2	1
2	2
...	
1000001	100000

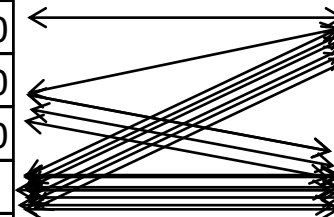


# SQL? No!

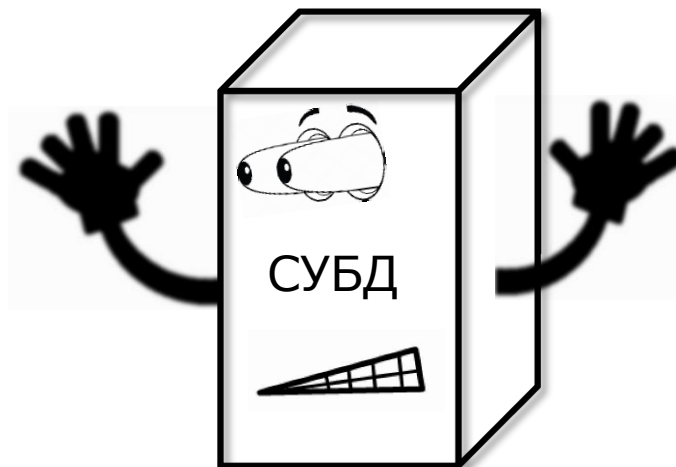
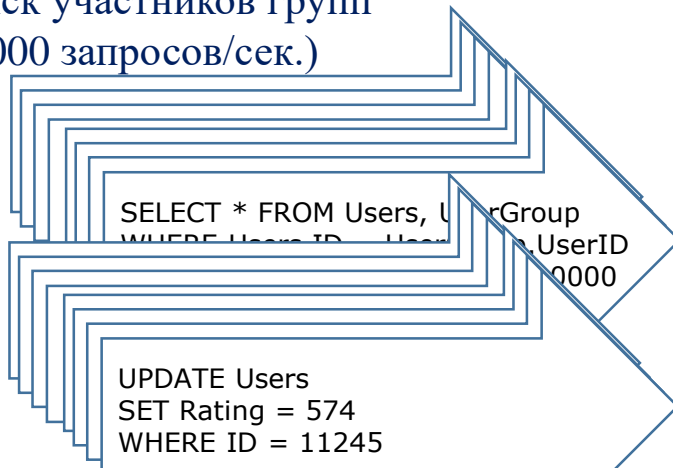
Если скорость обработки запроса  $> 1$  сек., то время ожидания  
тысяча первого запроса  $> 1000$  сек. (~17 минут).

Пользователи			
ID	ФИО	Роль	Рейтинг
1	Иванов И.И.	участник	300
2	Петров П.П.	участник	250
3	Сидоров С.С.	модератор	1000
...			
100000000	Новый Н.Н.	участник	0

ПользовательГруппа	
ID пользователя	ID группы
1	1
2	1
2	2
...	
1000001	100000



Поиск участников групп  
( $> 1000$  запросов/сек.)



Рейтинг пользователей по  
«лайкам» ( $> 10$  тыс. «лайков»/сек.)





# SQL + Шардирование

Разбиваем таблицы по серверам по простейшей хэш-функции  $f(x) = x \% 2$ :  
на каждом из N серверов в N раз меньше записей пользователей.

## Сервер 1

Users:  
 $ID \% 3 == 1$   
UserGroup:  
 $UserID \% 3 == 1$

Пользователи			
ID	ФИО	Роль	Рейтинг
1	Иванов И.И.	участник	300
...			
100000000	Новый Н.Н.	участник	0



ПользовательГруппа	
ID пользователя	ID группы
1	1
...	
1000000	100000

## Сервер 2

Users:  
 $ID \% 3 == 2$   
UserGroup:  
 $UserID \% 3 == 2$

Пользователи			
ID	ФИО	Роль	Рейтинг
2	Петров П.П.	участник	250
...			
99999998	Тапкин Т.Т.	участник	12



ПользовательГруппа	
ID пользователя	ID группы
2	1
...	
99999998	1200

## Сервер 3

Users:  
 $ID \% 3 == 0$   
UserGroup:  
 $UserID \% 3 == 0$

Пользователи			
ID	ФИО	Роль	Рейтинг
3	Сидоров С.С.	модератор	1000
...			
99999999	Уткин У.У.	участник	10



ПользовательГруппа	
ID пользователя	ID группы
3	4500
...	
99999999	300



# SQL + Шардирование

Добавляем новый атрибут Статус в таблицу Пользователи: т.к. должны обновиться все записи во всех таблицах – блокировка затронет все сервера.

Сервер 1



Пользователи			
ID	ФИО	Роль	Рейтинг
1	Иванов И.И.	участник	300
100000000	Новый Н.Н.	участник	0

Статус
в сети
...
в сети

Сервер 2



Пользователи			
ID	ФИО	Роль	Рейтинг
2	Петров П.П.	участник	250
99999998	Тапкин Т.Т.	участник	12

Статус
в сети
...
в сети

Сервер 3



Пользователи			
ID	ФИО	Роль	Рейтинг
3	Сидоров С.С.	модератор	1000
99999999	Уткин У.У.	участник	10

Статус
в сети
...
в сети



# Теорема CAP

В любой реализации распределенной базы данных возможно одновременно обеспечить не более двух из трех следующих свойств (Wiki):

- согласованность данных (Consistency) — во всех вычислительных узлах в один момент времени данные не противоречат друг другу;
- доступность (Availability) — любой запрос к распределенной системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают;
- устойчивость к разделению (Partition tolerance) — расщепление распределенной системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.

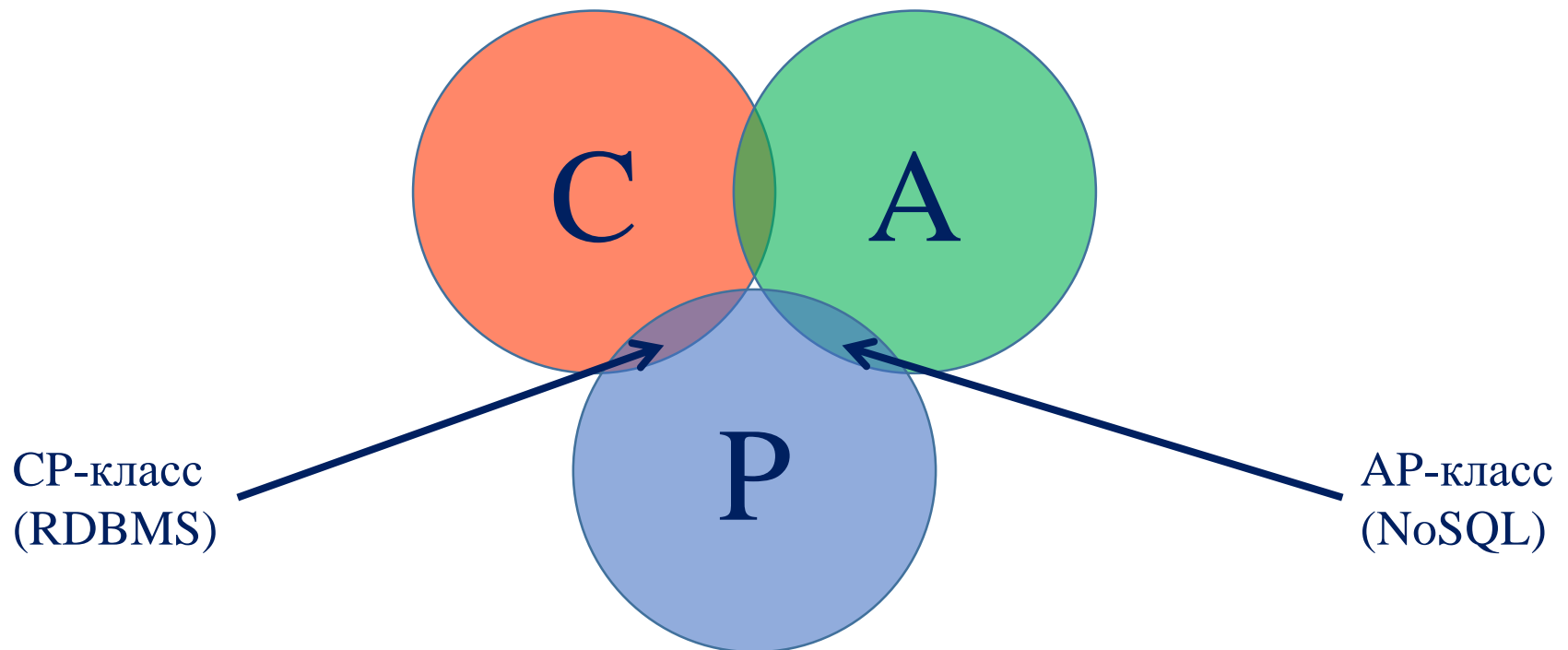
Очевидно, если мы правильно проектируем распределенную систему, то 3-е свойство будет выполнено всегда. А вот между первыми двумя часто приходится делать выбор.



# Классы CAP

Реляционные системы (RDBMS) условно ближе к CP-классу (в приоритете — согласованность).

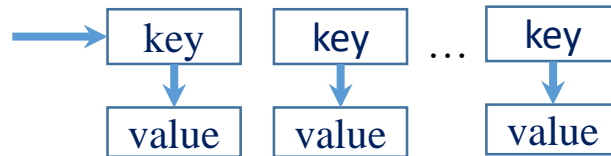
NoSQL-решения условно ближе к AP-классу (в приоритете — доступность и масштабируемость).



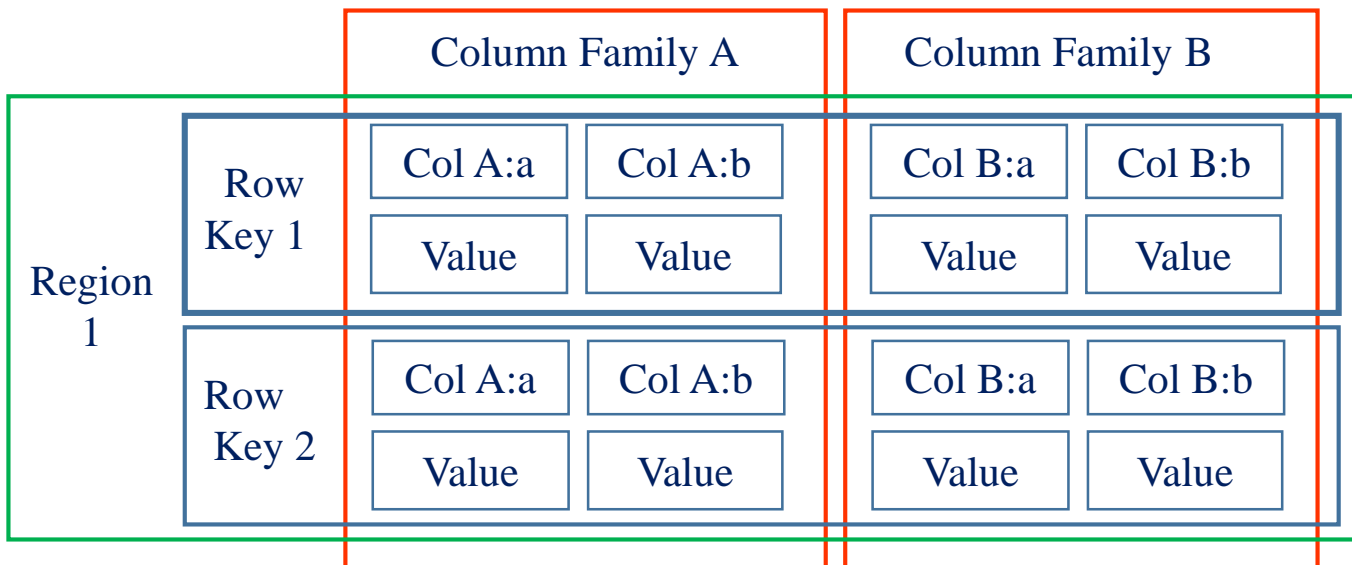


# NoSQL БД

## Key-value



## Wide-column

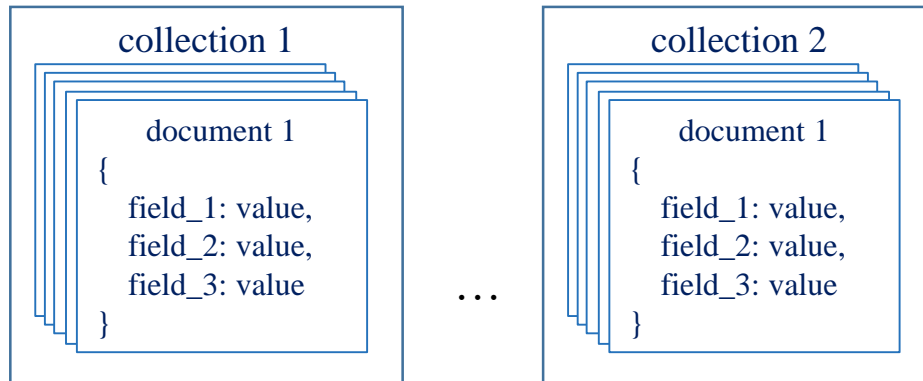




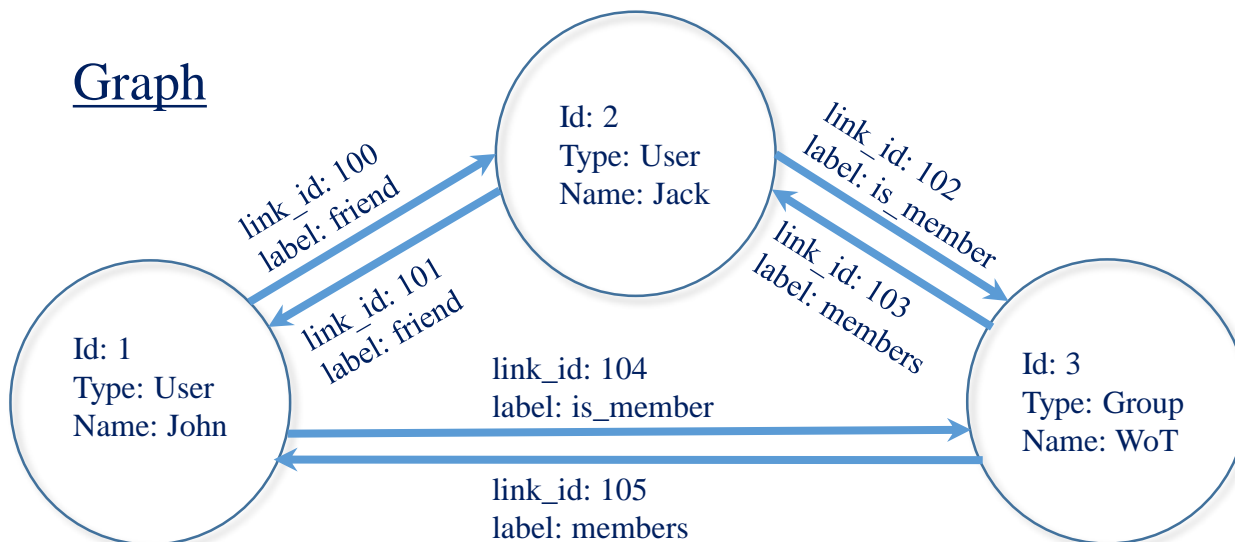


# NoSQL БД

## Document-oriented



## Graph





# ACID

Atomicity (Атомарность) — транзакция не может быть зафиксирована в системе частично: либо полное выполнение, либо полная отмена.

Consistency (Согласованность) — завершенная транзакция сохраняет согласованность базу данных.

Isolation (Изолированность) — во время выполнения транзакции параллельные транзакции не должны оказывать влияния на ее результат.

Durability (Долговечность) — низкоуровневые проблемы (например, обесточивание системы) не должны менять результат завершенной транзакции.



# BASE

Basic Availability (Базовая доступность) — допускается отказ в обслуживании для незначительной части сессий при сохранении доступности для большинства из них.

Soft state (Неустойчивое состояние) — допускается жертвовать долговременным хранением состояния сессии (например, промежуточных результатов выборок) при обеспечении фиксации обновлений для критичных операций.

Eventual consistency (Согласованность в конечном счете) — допускается обеспечивать согласованность только для отдельных частей БД (в основном, для конкретных записей), при условии последующей синхронизация всех узлов БД в фоновом режиме



# BASE вместо ACID

## ACID

- Atomicity (Атомарность)
- Consistency (Согласованность)
- Isolation (Изолированность)
- Durability (Долговечность)



## BASE

- Basic Availability (Базовая доступность)
- Soft state (Неустойчивое состояние)
- Eventual consistency (Согласованность в конечном счете)



# Schemaless

collection **users**

document

```
{
  id: 1,
  name: "Иванов И.И.",
  role: "участник",
  rating: 300,
  login: "ivanovi",
  password: "ivanov123",
  group_ids: [1]
}
```

document

```
{
  id: 2,
  name: "Петров И.И.",
  role: "участник",
  rating: 250,
  login: "petrovpi",
  password: "petrov123",
  group_ids: [1, 2]
}
```

...

document

```
{
  id: 1000,
  name: "Юрьев Ю.Ю.",
  role: "модератор",
  rating: 100,
  login: "youriev",
  password: "yoyo",
  avatar: "file15.png",
  status: "в сети"
}
```

`db.users.find( { id: 1 } );` # выполняется очень быстро

`db.users.find( { group_ids: 1 } );` # без индексов выполняется медленнее



# SQL и NoSQL интерфейсы

```
INSERT INTO Employees (Name, Position, Bonus, Login, Password)
VALUES ('Иванов И.И.', 'инженер', 30000, 'ivanovi', 'ivanov123');
```

```
db.users.insert( { id: 1, name: "Иванов И.И.", role: "участник", rating: 300, login: "ivanovi",
password: "ivanov123", group_ids: [1] } );
```

```
SELECT * FROM Employees;
```

```
SELECT Id, Name, Bonus FROM Employees WHERE Bonus > 20000 ORDER BY Bonus
DESC LIMIT 3;
```

```
db.users.find();
```

```
db.users.find( { rating: { $gt: 200 } }, { id: 1, name: 1, rating: 1 } ).sort( { rating : -1 } ).limit(3);
```

```
UPDATE Employees SET Bonus = 40000 WHERE Id = 1;
```

```
db.users.update( { id: 1 }, { $set: { rating: 400 } } );
```

```
DELETE FROM Employees WHERE Id = 5;
```

```
DELETE FROM Employees;
```

```
db.users.remove( { id: 5 } );
```

```
db.users.remove( { } );
```





# Эволюция БД

## A history of databases in No-tation

1970: **NoSQL = We have no SQL**

1980: **NoSQL = Know SQL**

2000: **NoSQL = No SQL!**

2005: **NoSQL = Not only SQL**

2013: **NoSQL = No, SQL!**



# SQL или NoSQL?

## Критерий №1: реляционность данных

Пробуем применить NoSQL-решение для исходной задачи:

collection **сотрудники**

document

```
{
  таб_номер: 1,
  фιο: "Иванов И.И.",
  должность: "инженер",
  бонус: 30000,
  id_проекта: 1
}
```

collection **должности**

document

```
{
  должность: "инженер",
  оклад: 50000
}
```

collection **проекты**

document

```
{
  id_проекта: 1,
  name: "Важный"
}
```



Если сотрудник участвует сразу в нескольких проектах, можно поменять тип id\_проекта сотрудника на Array: id\_проектов: [1]

**Но что если в разных проектах у сотрудника разные должности? Найти решение при нереляционном подходе можно, но это сложно.**




# SQL или NoSQL?

## Критерий №1: реляционность данных

В реляционном подходе задача решается элементарно: делаем должность атрибутом связи сотрудника и проекта. Данная предметная область реляционна, поэтому выбор в пользу SQL очевиден.

Сотрудники				
Таб. номер	ФИО	Премия	Логин	Пароль
1	Иванов И.И.	30000	ivanovi	ivanov123
2	Петров П.П.	50000	petrovp	p1e2t3
3	Сидоров С.С.	30000	sidorovs	zayka88
4	Егоров Е.Е.	20000	egorove	qwerty
5	Новый Н.Н.	20000	novyin	a1111

Проекты	
ID	Название
1	Важный
2	Срочный
3	Скучный



Должность-Оклад	
Название	Оклад
инженер	50000
старший инженер	51000
ведущий инженер	70000
менеджер проекта	100000

Сотрудник-Проект		
Таб. номер	ID проекта	Должность
1	1	инженер
2	1	старший инженер
2	2	инженер
3	2	менеджер
4	2	инженер



# SQL или NoSQL?

## Критерий №2: потенциальные запросы

Снова пробуем применить NoSQL-решение для исходной задачи:

- поиск информации по указанному сотруднику — **OK**
- сколько сотрудников в каждом из проектов — **???**



collection **сотрудники**

document

```
{
  таб_номер: 1,
  фιο: "Иванов И.И.",
  должность: "инженер",
  бонус: 30000,
  id_проектов: [1]
}
```

document

```
{
  таб_номер: 2,
  фιο: "Петров П.П.",
  должность: "старший инженер",
  бонус: 50000,
  id_проектов: [1, 2]
}
```

...

document

```
{
  таб_номер: 5,
  фιο: "Новый Н.Н.",
  должность: "инженер",
  бонус: 20000,
  id_проектов: []
}
```

Решение в реляционной модели:

**SELECT COUNT(EmployeeID), ProjectId FROM EmployeeProject  
GROUP BY ProjectId**





# SQL или NoSQL?

Пользователи							
ID	ФИО	Роль	Рейтинг	Логин	Пароль	Статус	Аватар
1	Иванов И.И.	участник	300	ivanovi	ivanov123		
2	Петров П.П.	участник	250	petrovp	p1e2t3		
3	Сидоров С.С.	модератор	1000	sidorovs	zayka88		file1.jpg
...							
100000000	Новый Н.Н.	участник	0	novyin	a1111	offline	

## collection users

### document

```
{
  id: 1,
  name: "Иванов И.И.",
  role: "участник",
  rating: 300,
  login: "ivanovi",
  password: "ivanov123",
  group_ids: [1]
}
```

### document

```
{
  id: 2,
  name: "Петров И.И.",
  role: "участник",
  rating: 250,
  login: "petrovp",
  password: "p1e2t3",
  group_ids: [1, 2]
}
```

...

### document

```
{
  id: 1000,
  name: "Юрьев Ю.Ю.",
  role: "модератор",
  rating: 100,
  login: "yourievu",
  password: "yoyo",
  avatar: "file15.png",
  status: "в сети"
}
```



# Тенденции

Реляционные системы (=> увеличение доступности данных):

- активная поддержка шардирования;
- совершенствование механизмов обработки данных.

NoSQL-системы (=> контроль согласованности данных):

- выполнение требований ACID (в MongoDB с июня 2018 года добавлена поддержка транзакций, удовлетворяющих требованиям ACID);
- приведение синтаксиса в соответствие с универсальным SQL

=> NewSQL?





**SELECT** questions **FROM** audience