#### Санкт-Петербургский Политехнический Университет Петра Великого

Физико-механический институт

Высшая школа прикладной математики и вычислительной физики

# 

# 

# 

# Отчет по курсу «базы данных»

##### 

##### 

##### 

##### 

##### 

##### 

##### 

##### 

##### 

##### 

##### Студент: Дрекалов Никита Сергеевич

##### Группа: 5030102/20001

##### Преподаватель: Крашенинников Сергей Леонидович

## Формулировка задания

#### Исходное задание – компания по разработке ПО:

Список текущих разработок компании.  
Список разработчиков, список тестеров (тестеры находят ошибки, разработчики, допустившие ошибки, их исправляют) – каждый может участвовать не более, чем в двух проектах.  
У проекта есть план – набор блоков кода, приписанные к блокам разработчики, время старта каждого блока, дедлайн сдачи готового (протестированного и исправленного) блока.  
Разработчики передают тестерам блоки (есть дата передачи и имя тестера). Время на тестирование – не более 10% от времени написания блока. Время на исправление ошибки – 1 день. Время написания блока + время тестирования + время исправления должно вписаться до дедлайна сдачи блока. Багов в блоке может не быть.  
По каждому проекту – список багов (обязательные поля: категория, кто заметил, когда, кто будет исправлять, срок для исправления, фактическое исправление).  
  
Требуется поддержка:

* Приема на работу;
* Перевода в новый проект;
* Завершения проекта;
* Планирования и старта нового проекта;
* Отслеживания текущего состояния проекта (по всем блокам проекта);
* Увольнения разработчика или тестера за N просрочек (по его вине) дедлайнов блоков с генерацией письма «товарищу» об этом
* Отчеты:
  + Список текущих разработок компании, с указанием занятых в проекте;
  + Список персонала, с указанием истории каждого (в каких проектах участвовал, упоряд. по времени старта проектов);
  + Качество разработки – диаграмма по завершенным проектам со следующим показателем – (общее количество багов)/(продолжительность проекта в человеко-днях)

#### Дополнительные соглашения предметной области:

Блок плана может быть создан только в паре с разработчиком и не может существовать без него. Разработчик увольняется => все его блоки планов закрываются, и на незавершенные задачи следует открывать новые.

Аналогично с тестами/багами: они создаются с привязкой к конкретному работнику и закрываются вместе с его увольнением.

#### Итоговое задание:

* Спроектировать и реализовать информационную систему, соответствующую исходному заданию с учетом дополнительных соглашений предметной области. Необходима следующая архитектура:

Список текущих разработок компании.  
Список разработчиков, список тестеров (тестеры находят ошибки, разработчики, допустившие ошибки, их исправляют) – каждый может участвовать не более, чем в двух проектах.  
У проекта есть план – набор блоков кода, приписанные к блокам разработчики, которые владеют блоками без возможности смены, время старта каждого блока, дедлайн сдачи готового (протестированного и исправленного) блока.  
Разработчики передают тестерам блоки (есть дата передачи и имя тестера) без возможности переназначить тестера. Время на тестирование – не более 10% от времени написания блока. Время на исправление ошибки – 1 день. Время написания блока + время тестирования + время исправления должно вписаться до дедлайна сдачи блока. Багов в блоке может не быть.  
По каждому проекту – список багов (обязательные поля: категория, кто заметил, когда, кто будет исправлять, срок для исправления, фактическое исправление).  
  
Требуется поддержка:

* + Приема на работу;
  + Перевода в новый проект;
  + Завершения проекта;
  + Планирования и старта нового проекта;
  + Отслеживания текущего состояния проекта (по всем блокам проекта);
  + Увольнения разработчика или тестера за N просрочек (по его вине) дедлайнов блоков с генерацией письма «товарищу» об этом
  + Отчеты:
    - Список текущих разработок компании, с указанием занятых в проекте;
    - Список персонала, с указанием истории каждого (в каких проектах участвовал, упоряд. по времени старта проектов);
    - Качество разработки – диаграмма по завершенным проектам со следующим показателем – (общее количество багов)/(продолжительность проекта в человеко-днях)
* Замерить время выполнения стандартных операций для таблиц, содержащих 1000, 10000, 100000 записей.
* Сделать выводы.

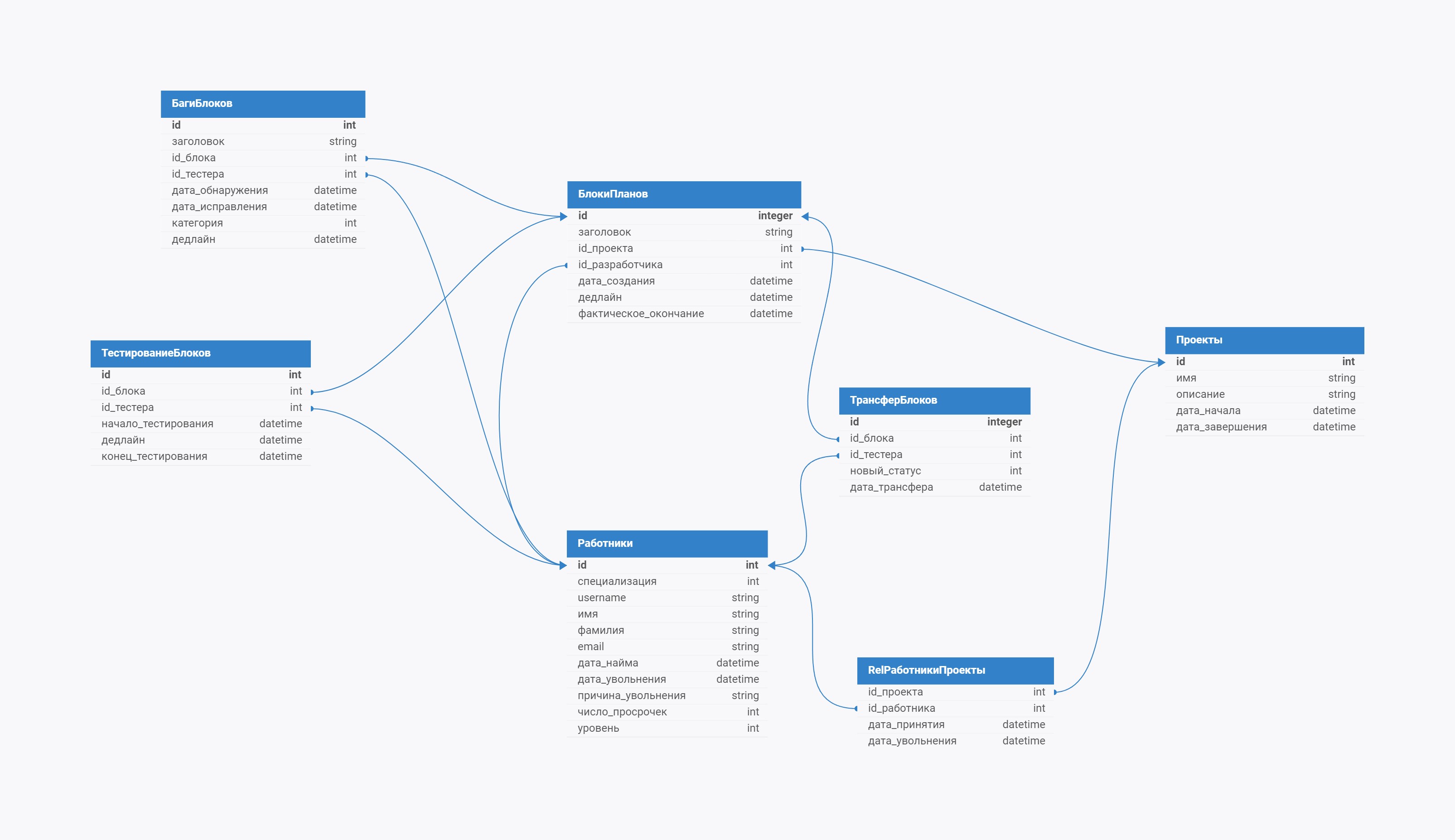
## 

## Используемый программный инструментарий

* *IDE:* WebStorm + PyCharm
* *СУБД:* PostgreSQL
* *Сервер:* Python FastAPI сервер. Работа с бд осуществляется через SQLAlchemy с движком psycopg
* *Интерфейс:* сайт на ReactJS

## Схема базы данных

<https://dbdesigner.page.link/LDimux69Uw96k2ow9>



Таблицы в базе данных находятся в 1NF, потому что:

* В таблицах нет должно дублирующих строк
* В каждой ячейке таблиц хранятся атомарные значения
* В столбцах хранятся данные одного типа
* Отсутствуют массивы и списки в любом виде.

Таблицы в базе данных находятся в 2NF, потому что:

* Таблицы находятся в первой нормальной форме
* Таблицы имеют первичный ключ
* Для таблицы m-n связи проект-работник, столбцы неключевые столбцы зависят от полного ключа (пары проект-работник), так как относятся именно к взаимоотношениям работник аи проекта.

Таблицы в базе данных находятся в 3NF, потому что:

* Таблицы находятся во второй нормальной форме
* В таблицах отсутствует транзитивная зависимость: неключевые столбцы не зависят от значений других неключевых столбцов:
  + «Работники»: все столбцы связаны напрямую с работником, т. е. зависят только от ключевого столбца
  + «Проекты»: имя, описание, дата начала и завершения связаны напрямую с проектом.
  + «Блоки планов»: заголовок блока, внешние ключи, дата создания, дедлайн и фактическое завершения относятся напрямую к блоку.
  + «Тестирование Блоков»: внешние ключи, даты начала и конца тестирования относятся напрямую к тесту.
  + «Баги блоков»: заголовок бага, внешние ключи, категория и даты относятся напрямую к багу.
  + «Трансфер блоков»: внешние ключи, новый статус и дата относятся напрямую к трансферу.
  + «Rel Работники Проекты»: даты принятия и увольнения относятся напрямую к паре первичных ключей.

Таблицы в базе данных находятся в BCNF, потому что:

* Единственный составной ключ в таблице отношения Проекты-Работники не зависит от других столбцов таблицы.

Таблицы в базе данных находятся в 4NF, потому что:

* В таблицах отсутствуют нетривиальные многозначные зависимости. Это следует из того, что при проектировке я делил абсолютно все 1:m и m:n связи, но если конкретно по таблицам:
  + «Работники» имеет в себе только независимые друг от друга столбцы (зависимые только от первичного ключа).
  + «Проекты» аналогично имеет в себе только столбцы с простейшими данными.
  + «Блоки планов» имеет в себе простейшие данные и связана внешними ключами с другими таблицами.
  + «Тестирование Блоков» опять же, имеет только независимые друг от друга столбцы и 2 внешних ключа.
  + «Баги блоков» аналогично.
  + «Трансфер блоков» имеет 2 внешних ключа и 2 дополнительных столбца – дата и новое состояние, которые очевидно не связаны друг с другом.
  + «Rel Работники Проекты» имеет составной первичный ключ и 2 столбца с датой принятия и увольнения работника, которые тоже не имеют связующее значение.

Таблицы в базе данных находятся в 5NF, потому что:

* Их всегда можно декомпозировать без потерь по первичному ключу.

## Основные запросы предметной области

#### Прием на работу:

Добавляем строку работника в базу данных

async def add\_worker(worker: WorkerAddDTO) -> int | None:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 try:  
 query = (  
 insert(WorkersORM)  
 .values(\*\*worker.model\_dump())  
 .returning(WorkersORM.id)  
 )  
  
 worker\_id = (await session.execute(query)).scalar\_one()  
  
 await session.commit()  
 return worker\_id  
 except IntegrityError:  
 return None

#### *Перевод в новый проект*

Проверяем валидность параметров. Затем добавляем запись в mn таблице о переводе, в старой записи проставляем дату увольнения и также закрываем все связанные с работником блоки

async def transfer\_worker(worker\_id: int, new\_project\_id: int, old\_project\_id: int | None) -> bool:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 get\_new\_project\_query = (  
 select(ProjectsORM)  
 .options(selectinload(ProjectsORM.workers))  
 .where(  
 and\_(  
 ProjectsORM.id == new\_project\_id,  
 ProjectsORM.end\_date == None  
 )  
 )  
 )  
 get\_worker\_query = (  
 select(WorkersORM)  
 .where(  
 and\_(  
 WorkersORM.id == worker\_id,  
 WorkersORM.fire\_date == None  
 )  
 )  
 )  
  
 new\_project = (await session.execute(get\_new\_project\_query)).scalar\_one\_or\_none()  
 worker = (await session.execute(get\_worker\_query)).scalar\_one\_or\_none()  
  
 if new\_project is None or worker is None:  
 return False  
  
 try:  
 insert\_query = (  
 insert(RelProjectsWorkersORM)  
 .values(  
 worker\_id=worker\_id,  
 project\_id=new\_project\_id,  
 project\_fire\_date=None  
 )  
 ).on\_conflict\_do\_update(  
 index\_elements=[RelProjectsWorkersORM.worker\_id, RelProjectsWorkersORM.project\_id],  
 set\_=dict(  
 worker\_id=worker\_id,  
 project\_id=new\_project\_id,  
 project\_fire\_date=None  
 )  
 )  
  
 await session.execute(insert\_query)  
 except IntegrityError as e:  
 print("Cringe")  
  
 if old\_project\_id is None:  
 count\_projects\_query = (  
 select(count()).select\_from(RelProjectsWorkersORM)  
 .where(  
 and\_(  
 RelProjectsWorkersORM.worker\_id == worker\_id,  
 RelProjectsWorkersORM.project\_fire\_date == None  
 )  
 )  
 )  
  
 count\_result = (await session.execute(count\_projects\_query)).scalar\_one()  
 print(count\_result)  
  
 if count\_result > 2:  
 return False  
  
 else:  
 end\_project\_query = (  
 update(RelProjectsWorkersORM)  
 .where(and\_(  
 RelProjectsWorkersORM.project\_id == int(old\_project\_id),  
 RelProjectsWorkersORM.worker\_id == worker\_id),  
 RelProjectsWorkersORM.project\_fire\_date == None  
 )  
 .values(project\_fire\_date=datetime.datetime.now(datetime.UTC))  
 .returning(ProjectsORM.id)  
 )  
  
 old\_project\_id = (await session.execute(end\_project\_query)).scalar\_one\_or\_none()  
  
 if old\_project\_id is None:  
 return False  
  
 await session.commit()  
 return True

#### Завершение проекта

Закрываем проект (проставляем дату завершения), увольняем всех работников (аналогично, проставляем дату завершения в mn таблице) и закрываем все блоки планов.

async def close\_project(project\_id: int) -> ProjectOnCloseDTO | None:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 # Close project  
 project\_query = (  
 update(ProjectsORM)  
 .where(  
 and\_(  
 ProjectsORM.id == project\_id,  
 ProjectsORM.end\_date == None  
 )  
 )  
 .values(end\_date=datetime.datetime.now(datetime.UTC))  
 .returning(ProjectsORM.id)  
 )  
  
 project\_id = (await session.execute(project\_query)).scalar\_one\_or\_none()  
  
 if project\_id is None:  
 return None  
  
 # Fire all workers from project  
 workers\_query = (  
 update(RelProjectsWorkersORM)  
 .where(  
 and\_(  
 RelProjectsWorkersORM.project\_id == project\_id,  
 RelProjectsWorkersORM.project\_fire\_date == None  
 )  
 )  
 .values(project\_fire\_date=datetime.datetime.now(datetime.UTC))  
 .returning(RelProjectsWorkersORM.worker\_id)  
 )  
 # Select all opened plan blocks id in project  
 plan\_blocks\_query = (  
 select(PlanBlocksORM.id)  
 .where(  
 and\_(  
 PlanBlocksORM.project\_id == project\_id,  
 PlanBlocksORM.end\_date == None  
 )  
 )  
 )  
  
 workers\_id = (await session.execute(workers\_query)).scalars().all()  
 plan\_blocks\_id = (await session.execute(plan\_blocks\_query)).scalars().all()  
  
 # Close all plan blocks bugs, tests  
 await close\_plan\_blocks(list(map(int, plan\_blocks\_id)))  
  
 await session.commit()  
 return ProjectOnCloseDTO(  
 project\_id=project\_id,  
 workers\_id=workers\_id,  
 plan\_blocks\_id=plan\_blocks\_id  
 )

#### Планирование и старт нового проекта

Добавляем строку о проекте в таблицу

async def add\_project(project: ProjectAddDTO) -> int | None:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 try:  
 query = (  
 insert(ProjectsORM)  
 .values(\*\*project.model\_dump())  
 .returning(ProjectsORM.id)  
 )  
  
 project\_id = (await session.execute(query)).scalar\_one()  
  
 await session.commit()  
 return project\_id  
 except IntegrityError:  
 return None

#### Отслеживание текущего состояния проекта

Есть возможность получить строку проекта, работников и блоки планов (через Join)

async def get\_project(project\_id: int) -> ProjectDTO | None:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 project\_orm = await session.get(ProjectsORM, project\_id)  
  
 return ProjectDTO.model\_validate(project\_orm, from\_attributes=True) if project\_orm else None

async def get\_workers\_from\_project(project\_id: int) -> list[WorkerByProjectDTO]:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 query = (  
 select(WorkersORM.\_\_table\_\_.columns, RelProjectsWorkersORM.project\_hire\_date, RelProjectsWorkersORM.project\_fire\_date)  
 .select\_from(RelProjectsWorkersORM)  
 .where(RelProjectsWorkersORM.project\_id == project\_id)  
 .join(WorkersORM, RelProjectsWorkersORM.worker\_id == WorkersORM.id)  
 )  
  
 workers\_orm = (await session.execute(query)).all()  
  
 workers = [WorkerByProjectDTO.model\_validate(worker\_orm, from\_attributes=True) for worker\_orm in workers\_orm]  
  
 await session.commit()  
 return workers

async def get\_plan\_blocks(project\_id: int) -> list[PlanBlockDTO]:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 query = (  
 select(  
 PlanBlocksORM.\_\_table\_\_.columns,  
 PlanBlocksTransferORM.new\_status.label("status"),  
 PlanBlocksTransferORM.date.label("status\_date")  
 )  
 .distinct(PlanBlocksORM.id)  
 .where(PlanBlocksORM.project\_id == project\_id)  
 .join(PlanBlocksTransferORM, PlanBlocksTransferORM.block\_id == PlanBlocksORM.id)  
 .order\_by(PlanBlocksORM.id, PlanBlocksTransferORM.date.desc())  
 )  
  
 plan\_blocks\_orm = (await session.execute(query)).all()  
  
 return [PlanBlockDTO.model\_validate(plan\_block, from\_attributes=True) for plan\_block in plan\_blocks\_orm]

#### Увольнения разработчика или тестера за N просрочек (по его вине) дедлайнов блоков с генерацией письма «товарищу» об этом

Увольнение работника – проставление даты увольнения и закрытие всех связанных с ним блоков плана. При увольнении пишется письмо на почту работнику (мою почту) через mailopost.

async def fire\_worker(worker\_id: int, fire\_reason: str) -> WorkerOnFireDTO | None:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 # Set worker fire info  
 query = (  
 update(WorkersORM)  
 .where(  
 and\_(  
 WorkersORM.id == worker\_id,  
 WorkersORM.fire\_date == None  
 )  
 )  
 .values(  
 fire\_date=datetime.datetime.now(datetime.UTC),  
 fire\_reason=fire\_reason  
 )  
 .returning(WorkersORM)  
 )  
  
 worker\_orm = (await session.execute(query)).scalar\_one\_or\_none()  
 if worker\_orm is None:  
 return None  
 worker = WorkerDTO.model\_validate(worker\_orm, from\_attributes=True)  
  
 # Fire worker from all projects  
 projects\_query = (  
 update(RelProjectsWorkersORM)  
 .where(  
 and\_(  
 RelProjectsWorkersORM.worker\_id == worker\_id,  
 RelProjectsWorkersORM.project\_fire\_date == None  
 )  
 )  
 .values(project\_fire\_date=datetime.datetime.now(datetime.UTC))  
 .returning(RelProjectsWorkersORM.project\_id)  
 )  
  
 if worker.specialization\_code == SpecializationCode.developer:  
 # Select all plan blocks related to developer  
 plan\_blocks\_query = (  
 select(PlanBlocksORM.id)  
 .where(  
 and\_(  
 PlanBlocksORM.developer\_id == worker\_id,  
 PlanBlocksORM.end\_date == None  
 )  
 )  
 )  
  
 plan\_blocks\_id = (await session.execute(plan\_blocks\_query)).scalars().all()  
 # Close all plan blocks  
 await close\_plan\_blocks(list(map(int, plan\_blocks\_id)))  
 else:  
 # Close all block test and bugs related to tester  
 block\_testing\_query = (  
 update(BlockTestingORM)  
 .where(  
 and\_(  
 BlockTestingORM.tester\_id == worker\_id,  
 BlockTestingORM.end\_date == None  
 )  
 )  
 .values(end\_date=datetime.datetime.now(datetime.UTC))  
 .returning(BlockTestingORM.id)  
 )  
 block\_bugs\_query = (  
 update(BlockBugsORM)  
 .where(  
 and\_(  
 BlockBugsORM.tester\_id == worker\_id,  
 BlockBugsORM.fix\_date == None  
 )  
 )  
 )  
 block\_testings\_id = (await session.execute(block\_testing\_query)).scalars().all()  
 block\_bugs\_id = (await session.execute(block\_bugs\_query)).scalars().all()  
  
 projects\_id = (await session.execute(projects\_query)).scalars().all()  
  
 await session.commit()  
 return WorkerOnFireDTO(  
 \*\*worker.model\_dump(),  
 projects\_id=projects\_id,  
 )

async def add\_overdue(worker\_id: int) -> WorkerOnFireDTO | None:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 query = (  
 update(WorkersORM)  
 .where(WorkersORM.id == worker\_id)  
 .values(overdue\_count=WorkersORM.overdue\_count+1)  
 .returning(WorkersORM)  
 )  
  
 worker = (await session.execute(query)).scalars().first()  
  
 if worker is None or worker.overdue\_count < 7:  
 await session.commit()  
 return None  
  
 await session.commit()  
 return await fire\_worker(worker\_id, "Число просрочек превысило допустимый лимит")

#### Список текущих разработок компании, с указанием занятых в проекте

Пользуясь силой SQLAlchemy, получаю список проектов со списками работников одной командой (можно было а. через Join и с фильтрацией по уникальности b. Через 2 SELECT с распределением работников по проектам (так делает алхимия внутри)

async def projects\_list() -> list[ProjectReportDTO]:  
 session: AsyncSession  
 async with async\_session\_factory() as session:  
 query = (  
 select(ProjectsORM)  
 .options(selectinload(ProjectsORM.workers))  
 .order\_by(ProjectsORM.end\_date.desc(), ProjectsORM.name)  
 )  
  
 projects\_orm = (await session.execute(query)).scalars().all()  
  
 return [ProjectReportDTO.model\_validate(project, from\_attributes=True) for project in projects\_orm]

#### Список персонала с указанием истории каждого

Аналогично проектам.

async def personal\_list() -> list[WorkerReportDTO]:  
 session: AsyncSession  
 async with (async\_session\_factory() as session):  
 query = (  
 select(WorkersORM)  
 .options(selectinload(WorkersORM.projects))  
 .order\_by(WorkersORM.fire\_date.desc(), WorkersORM.username)  
 )  
  
 workers\_orm = (await session.execute(query)).scalars().all()  
  
 return [WorkerReportDTO.model\_validate(worker, from\_attributes=True) for worker in workers\_orm]

#### Качество разработки

Через проекты -> блоки планов -> баги получаем список багов по проектам, суммируем их число и делим на длительность проекта – получаем качество разработки. (Да, это выгружает 2/3 бд в оперативку, но пойдет… Можно было через COUNT)

async def get\_development\_quality() -> list[ProjectQualityDTO]:  
 session: AsyncSession  
 async with (async\_session\_factory() as session):  
 query = (  
 select(ProjectsORM)  
 .where(ProjectsORM.end\_date != None)  
 .order\_by(ProjectsORM.start\_date)  
 .options(  
 selectinload(ProjectsORM.plan\_blocks)  
 .selectinload(PlanBlocksORM.block\_bugs)  
 )  
 )  
  
 projects\_with\_blocks = (await session.execute(query)).scalars().all()  
 projects\_qualities: list[ProjectQualityDTO] = []  
  
 for project in projects\_with\_blocks:  
 start\_date: datetime.datetime = project.start\_date  
 end\_date: datetime.datetime = project.end\_date  
 project\_duration: int = (end\_date - start\_date).days  
 if project\_duration == 0:  
 project\_duration = 1  
 bug\_cnt = 0  
 for plan\_block in project.plan\_blocks:  
 bug\_cnt += len(plan\_block.block\_bugs)  
  
 projects\_qualities.append(ProjectQualityDTO(  
 project\_name=project.name,  
 quality=bug\_cnt/project\_duration  
 ))  
  
 return projects\_qualities

## Реализация стандартных операций

**Функция тестирования (общая)**

async def do\_tests(exp\_num: int, query1000, query10000, query100000, do\_commit: bool = False):  
 async with async\_session\_factory() as session:  
 open\_query = (  
 select(TestModel1000)  
 .where(TestModel1000.id == 1)  
 )  
 await session.execute(open\_query)  
  
 for i in range(3):  
 delta\_time = 0  
 for j in range(exp\_num):  
 start\_time = time.time\_ns()  
 await session.execute(query1000 if i == 0 else (query10000 if i == 1 else query100000))  
 end\_time = time.time\_ns()  
 delta\_time += (end\_time - start\_time) / exp\_num / 1000  
 print(f"{1000 \* 10 \*\* i} строк - {int(delta\_time)}")  
  
 if do\_commit:  
 await session.commit()

##### Поиск по ключевому полю

async def search\_by\_key(exp\_num: int):  
 query1000 = (  
 select(TestModel1000)  
 .where(TestModel1000.id == 900)  
 )  
 query10000 = (  
 select(TestModel10000)  
 .where(TestModel10000.id == 9000)  
 )  
 query100000 = (  
 select(TestModel100000)  
 .where(TestModel100000.id == 12345)  
 )  
  
 print("Поиск по ключевому полю")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Поиск по неключевому полю

async def search\_by\_non\_key(exp\_num: int):  
 query1000 = (  
 select(TestModel1000)  
 .where(TestModel1000.fake\_unique == "900")  
 )  
 query10000 = (  
 select(TestModel10000)  
 .where(TestModel10000.fake\_unique == "9000")  
 )  
 query100000 = (  
 select(TestModel100000)  
 .where(TestModel100000.fake\_unique == "90000")  
 )  
  
 print("Поиск по НЕ ключевому полю")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Поиск по маске

async def search\_by\_mask(exp\_num: int):  
 query1000 = (  
 select(TestModel1000)  
 .where(TestModel1000.some\_string.icontains("i"))  
 )  
 query10000 = (  
 select(TestModel10000)  
 .where(TestModel10000.some\_string.icontains("i"))  
 )  
 query100000 = (  
 select(TestModel100000)  
 .where(TestModel100000.some\_string.icontains("i"))  
 )  
  
 print("Поиск по маске")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Добавление записи

async def add\_one\_row(exp\_num: int):  
 query1000 = (  
 insert(TestModel1000)  
 .values(  
 name="some name",  
 some\_string="some string",  
 date=datetime.datetime.now(),  
 fake\_unique="1"  
 )  
 )  
 query10000 = (  
 insert(TestModel10000)  
 .values(  
 name="some name",  
 some\_string="some string",  
 date=datetime.datetime.now(),  
 fake\_unique="1"  
 )  
 )  
 query100000 = (  
 insert(TestModel100000)  
 .values(  
 name="some name",  
 some\_string="some string",  
 date=datetime.datetime.now(),  
 fake\_unique="1"  
 )  
 )  
  
 print("Добавление записи")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Добавление группы записей

async def add\_many\_rows(exp\_num: int):  
 rows: list[dict[str, str]] = list()  
  
 for i in range(100):  
 rows.append({  
 "name": "some name",  
 "some\_string": "some string",  
 "date": datetime.datetime.now(),  
 "fake\_unique": "1"  
 })  
  
 query1000 = (  
 insert(TestModel1000)  
 .values(rows)  
 )  
 query10000 = (  
 insert(TestModel10000)  
 .values(rows)  
 )  
 query100000 = (  
 insert(TestModel100000)  
 .values(rows)  
 )  
  
 print("Добавление группы записей")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Изменение записи

async def update\_row\_by\_key(exp\_num: int):  
 query1000 = (  
 update(TestModel1000)  
 .where(TestModel1000.id == 900)  
 .values(some\_string="changed some string")  
 )  
 query10000 = (  
 update(TestModel10000)  
 .where(TestModel10000.id == 9000)  
 .values(some\_string="changed some string")  
 )  
 query100000 = (  
 update(TestModel100000)  
 .where(TestModel100000.id == 90000)  
 .values(some\_string="changed some string")  
 )  
  
 print("Изменение записи по ключевому полю")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

async def update\_row\_by\_non\_key(exp\_num: int):  
 query1000 = (  
 update(TestModel1000)  
 .where(TestModel1000.fake\_unique == "900")  
 .values(some\_string="changed some string")  
 )  
 query10000 = (  
 update(TestModel10000)  
 .where(TestModel10000.fake\_unique == "9000")  
 .values(some\_string="changed some string")  
 )  
 query100000 = (  
 update(TestModel100000)  
 .where(TestModel100000.fake\_unique == "90000")  
 .values(some\_string="changed some string")  
 )  
  
 print("Изменение записи по НЕ ключевому полю")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Удаление записи

async def delete\_row\_by\_key(exp\_num: int):  
 query1000 = (  
 delete(TestModel1000)  
 .where(TestModel1000.id == 900)  
 )  
 query10000 = (  
 delete(TestModel10000)  
 .where(TestModel10000.id == 9000)  
 )  
 query100000 = (  
 delete(TestModel100000)  
 .where(TestModel100000.id == 90000)  
 )  
 print("Удаление записи по ключевому полю")  
 await do\_tests(exp\_num, query1000, query10000, query100000)  
  
  
async def delete\_row\_by\_non\_key(exp\_num: int):  
 query1000 = (  
 delete(TestModel1000)  
 .where(TestModel1000.fake\_unique == "900")  
 )  
 query10000 = (  
 delete(TestModel10000)  
 .where(TestModel10000.fake\_unique == "9000")  
 )  
 query100000 = (  
 delete(TestModel100000)  
 .where(TestModel100000.fake\_unique == "90000")  
 )  
 print("Удаление записи по НЕ ключевому полю")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Удаление группы записей

async def delete\_rows(exp\_num: int):  
 to\_delete = range(100, 200)  
  
 query1000 = (  
 delete(TestModel1000)  
 .where(TestModel1000.id.in\_(to\_delete))  
 )  
 query10000 = (  
 delete(TestModel10000)  
 .where(TestModel10000.id.in\_(to\_delete))  
 )  
 query100000 = (  
 delete(TestModel100000)  
 .where(TestModel100000.id.in\_(to\_delete))  
 )  
  
 print("Удаление группы записей")  
 await do\_tests(exp\_num, query1000, query10000, query100000)  
  
  
async def delete\_rows\_by\_non\_key(exp\_num: int):  
 to\_delete = range(100, 200)  
 to\_delete = list(map(lambda el: str(el), to\_delete))  
  
 query1000 = (  
 delete(TestModel1000)  
 .where(TestModel1000.fake\_unique.in\_(to\_delete))  
 )  
 query10000 = (  
 delete(TestModel10000)  
 .where(TestModel10000.fake\_unique.in\_(to\_delete))  
 )  
 query100000 = (  
 delete(TestModel100000)  
 .where(TestModel100000.fake\_unique.in\_(to\_delete))  
 )  
  
 print("Удаление группы записей по неключевому полю")  
 await do\_tests(exp\_num, query1000, query10000, query100000)

##### Сжатие базы данных

async def delete\_200\_rows(exp\_num: int):  
 to\_delete = range(100, 200)  
  
 query1000 = (  
 delete(TestModel1000)  
 .where(TestModel1000.id.in\_(to\_delete))  
 )  
 query10000 = (  
 delete(TestModel10000)  
 .where(TestModel10000.id.in\_(to\_delete))  
 )  
 query100000 = (  
 delete(TestModel100000)  
 .where(TestModel100000.id.in\_(to\_delete))  
 )  
  
 print("Удаление 200 записей по неключевому полю с коммитом")  
 await do\_tests(exp\_num, query1000, query10000, query100000, True)  
  
  
async def remain\_200\_rows(exp\_num: int):  
 to\_delete1000 = range(200, 1000)  
 to\_delete10000 = range(200, 1000)  
 to\_delete100000 = range(200, 10000)  
  
 query1000 = (  
 delete(TestModel1000)  
 .where(TestModel1000.id.in\_(to\_delete1000))  
 )  
 query10000 = (  
 delete(TestModel10000)  
 .where(TestModel10000.id.in\_(to\_delete10000))  
 )  
 query100000 = (  
 delete(TestModel100000)  
 .where(TestModel100000.id.in\_(to\_delete100000))  
 )  
  
 print("Оставление 200 записей по неключевому полю с коммитом")  
 await do\_tests(exp\_num, query1000, query10000, query100000, True)

async def vacuum\_tables():  
 print("Сжатие базы данных")   
 conn: AsyncConnection  
 async with async\_engine.connect() as conn:  
 await conn.execution\_options(isolation\_level="AUTOCOMMIT")  
 for i in range(3):  
 start\_time = time.time\_ns()  
 await conn.execute(text(f"VACUUM test\_model\_{1000 \* 10 \*\* i}"))  
 end\_time = time.time\_ns()  
 delta\_time = (end\_time - start\_time) / 1000  
 print(f"{1000 \* 10 \*\* i} строк - {int(delta\_time)}")

## Сравнение временных затрат при реализации стандартных операций

|  |  |  |  |
| --- | --- | --- | --- |
| Число записей | 1000 | 10000 | 100000 |
| 1. Поиск по ключевому полю, мкс | 559 | 561 | 519 |
| 1. Поиск по не ключевому полю, мкс | 490 | 1550 | 10223 |
| 1. Поиск по маске, мкс | 10^4 | 1.1\*10^5 | 9.9\*10^5 |
| 1. Добавление записи, мкс | 419 | 480 | 544 |
| 1. Добавление группы записей (100), мкс | 7227 | 6976 | 7496 |
| 1. Изменение записи (определение изменяемой записи по ключевому полю), мкс | 391 | 459 | 520 |
| 1. Изменение записи (определение изменяемой записи по не ключевому полю), мкс | 500 | 1840 | 10756 |
| 1. Удаление записи (определение удаляемой записи по ключевому полю), мкс | 322 | 286 | 360 |
| 1. Удаление записи (определение удаляемой записи по не ключевому полю), мкс | 459 | 1486 | 10242 |
| 1. Удаление группы записей (100), мкс | 1310 | 1477 | 1478 |
| 1. Удаление группы записей по неключевому полю (100), мкс | 1197 | 2602 | 13714 |
| 1. Сжатие базы данных (после удаления из БД 200 строк), мкс | 2019 | 2995 | 8070 |
| 1. Сжатие базы данных (после удаления, в результате которого в БД остается 200 строк), мкс | 4991 | 2999 | 4998 |

## Выводы

Запросы по ключевому полю выполняются за O(1). Т. е. для поиска по

ключевому полю СУБД использует хэш-функцию.

Запросы по неключевому полю выполняются за O(n). Т. е. для поиска по неключевому полю СУБД идёт по всем строкам таблицы и проверяет совпадения.

Запрос по маске выполняется в среднем за O(n\*k), где k – длина строкового столбца.

Сжатие базы данных выполняется за линию относительно оставшихся числа строк в таблице. Это связано с тем, что для сжатия создается новая таблица, куда копируются только неудаленные данные, а старая просто удаляется.