МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)» (МАИ)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект
по дисциплине «Базы данных»
Тема: «Информационная система для управления ML-экспериментами и моделями»

Студент: Ибрагимов Д.М.
Группа: М8О-308Б-23
Преподаватель: Грубенко М.Д.
Оценка: _____
Дата: _____
Подпись: _____

Москва 2025

Содержание

Введение

Объект исследования — информационная система для управления жизненным циклом ML-экспериментов и моделей. Предмет исследования — проектирование реляционной базы данных, поддерживающей аудит, аналитические запросы и интеграцию с backend-API. Цель работы — разработать систему, демонстрирующую проектирование БД, ограничения целостности, функции, триггеры, оптимизацию запросов и batch-import.

1. Аналитическая часть

Системы типа MLflow/Weights&Biases фиксируют конфигурации запусков, метрики и артефакты. В курсовой работе важно обеспечить целостность данных и воспроизводимость, а также поддержать аналитические отчёты. Ключевые сценарии: регистрация пользователей, создание проектов и экспериментов, запуск runs, логирование метрик, хранение ссылок на артефакты и сравнение результатов.

1.1. Требования к системе

Согласно заданию, система должна демонстрировать полный цикл проектирования БД и интеграцию с backend:

- не менее 9–10 таблиц, связи 1:1, 1:N, N:M, каждая таблица с 5+ полями;
- ключевые сущности предметной области и журнал аудита изменений;
- ограничения PK/FK/UNIQUE/CHECK/NOT NULL и каскадные действия;
- объём данных: крупные таблицы 500–1000 записей, транзакционные — 5000+;
- batch-import с логированием ошибок построчно;
- CRUD через API, аналитические запросы через SQL;

- минимум 3 VIEW и 2 функции (скалярная и табличная), триггеры на аудит и агрегаты;
- индексы и демонстрация улучшения производительности до/после (планы выполнения — в приложении);
- Swagger/OpenAPI документация, контейнеризация через Docker Compose;
- отсутствие секретов в коде и параметризация SQL-запросов.

1.2. Теоретические основы

Реляционная модель описывает данные в виде таблиц (отношений) с ключами и ограничениями, что позволяет обеспечивать целостность и согласованность при изменениях. В работе применяются принципы нормализации до 3НФ: данные разделены на сущности (users, projects, experiments), а зависимости фиксируются внешними ключами. Это уменьшает дублирование и упрощает поддержку связей.

Для обеспечения надёжности используются транзакции: операции, затрагивающие несколько таблиц (например, создание run и run_configs), выполняются атомарно. Отдельный журнал аудита фиксирует изменения строк с указанием операции и автора, что облегчает контроль качества данных.

Индексация применяется к полям в условиях WHERE/JOIN/ORDER BY. Для аналитики используются VIEW, а для типовых отчётов — функции, инкапсулирующие сложную логику вычислений.

2. Проектная часть

2.1. Архитектура системы

Система построена по трёхзвенной схеме: клиент (Swagger/Frontend) → backend API → PostgreSQL. Компоненты развёрнуты через Docker Compose и обмениваются данными по внутренней сети; конфигурация передаётся через переменные окружения.

В составе развертывания выделены сервисы:

- frontend — статический интерфейс (дашборды) на Nginx, обращается к API по HTTP;
- backend — FastAPI-приложение, реализует CRUD, отчёты и batch-import;
- db — PostgreSQL с ограничениями, индексами, VIEW, функциями и триггерами.

Backend реализует слоистую архитектуру:

- слой API (routers) — HTTP-эндпойнты и маршрутизация запросов;
- слой схем (schemas) — валидация и сериализация данных;
- слой доменной модели (models) — ORM-сущности и связи;
- слой доступа к данным (db) — сессии, транзакции, конфигурация соединений;
- слой безопасности (auth/permissions) — JWT-аутентификация и RBAC-проверки.

CRUD-операции выполняются через ORM, а аналитика — через параметризованные SQL-запросы, функции и представления. Все SQL-объекты (VIEW, функции, триггеры) хранятся в sql/ и применяются миграциями Alembic, что обеспечивает воспроизводимость схемы. Поток обработки запроса включает проверку JWT, контроль роли в организации/проекте, выполнение транзакции и фиксацию аудита через триггеры. Backend устанавливает идентификатор пользователя в контексте БД (set_config('app.user_id', ...)), чтобы аудит фиксировал автора изменений. Batch-import записывает задания в batch_import_jobs и ошибки в batch_import_errors без остановки всего процесса. Артефакты моделей хранятся в виде ссылок (URI), что исключает хранение бинарных данных в БД.

2.2. Проектирование структуры базы данных

Схема включает 18 таблиц, обеспечивающих связи 1:1, 1:N и N:M:

- users, organizations, org_members
- ml_projects, project_members
- datasets, dataset_versions
- experiments, runs, run_configs
- metric_definitions, run_metric_values
- artifacts, run_artifacts
- audit_log
- batch_import_jobs, batch_import_errors
- project_metric_summary

Ключевые связи:

- 1:1 — runs → run_configs.
- 1:N — projects → datasets; experiments → runs.
- N:M — runs ↔ artifacts (через run_artifacts).

Ограничения целостности реализованы через PK, FK, UNIQUE, CHECK и NOT NULL, с каскадным удалением/обновлением для зависимых сущностей. Поля времени: created_at / updated_at (где применимо). Все изменения схемы выполняются через Alembic-миграции. ER-диаграмма представлена в приложении.

2.3. Описание ключевых таблиц

Ниже приведён укрупнённый перечень основных сущностей и их назначение:

Таблица	**Назначение и ключевые поля**
users	Пользователи системы: email, full_name, password_hash, is_active, created_at.
organizations	Организации/команды: name, description, created_by, created_at.
org_members	Членство пользователей в организациях: role, joined_at, is_active, UNIQUE(org_id, user_id).
ml_projects	ML-проекты в рамках организации: name, status, description, created_at.
project_members	Роли пользователей в проектах: role, added_at, is_active, UNIQUE(project_id, user_id).
datasets / dataset_versions	Датасеты и версии: version_label, storage_uri, content_hash, size_bytes.
experiments / runs	Эксперименты и запуски: status, started_at, finished_at, git_commit, notes.
run_configs	Конфигурации запусков (1:1): params_json, env_json, command_line, seed.
metric_definitions / run_metric_values	Справочник метрик и значения: goal, scope, step, value.

artifacts / run_artifacts	Артефакты и связи с runs: uri, artifact_type, checksum, alias.
audit_log	Журнал аудита: table_name, operation, row_pk, changed_by, old_data, new_data.
batch_import_jobs / batch_import_errors	Журнал batch-импорта: status, source_format, stats_json, row_level ошибки.
project_metric_ summary	Агрегаты по проекту: best_value, best_run_id, sample_size.

Таблица 1: Перечень ключевых таблиц и их назначение

2.4. Ограничения целостности и каскады

Для корректности данных применяются:

- уникальность ключевых полей (email, имена в рамках org/project, ключи справочников);
- проверки статусов и ролей через CHECK (например, status IN (’active’,’archived’));
- каскадные действия: удаление проекта удаляет датасеты/ эксперименты/артефакты и связанные сущности;
- защита транзакционных таблиц: для run_metric_values запрещены отрицательные step.

2.5. Ролевая модель и безопасность

Доступ к данным реализован через JWT-авторизацию и роли в организациях/проектах. Для CRUD-операций проверяется минимально необходимая роль: viewer для чтения, editor/admin для модификаций. Для аудита используется set_config('app.user_id', ...) в транзакции. SQL-запросы выполняются только с параметризацией, что исключает SQL-инъекции. Секреты и учётные данные не хранятся в коде и передаются через .env.

2.6. Batch-import и транзакции

Batch-import поддерживает загрузку metrics и datasets из CSV/JSON. Каждая строка обрабатывается отдельно: успешные записи фиксируются, ошибки сохраняются в batch_import_errors без остановки всего job. В batch_import_jobs хранится статус, источник и статистика (inserted / errors). Транзакции используются для операций, затрагивающих несколько таблиц (создание run + run_configs, завершение run с финальными метриками).

2.7. Представления

Реализованы три VIEW:

- v_runs_with_final_metrics — финальные метрики по runs;
- v_best_runs_per_experiment — лучший run по ключевой метрике;
- v_project_quality_dashboard — агрегаты по проекту (success rate, медиана времени, best metric).

2.8. Функции и триггеры

Функции:

- fn_best_run_id — скалярная функция, возвращающая лучший run по метрике и цели (min/max/last).
- fn_experiment_leaderboard — табличная функция для top-N runs по метрике.

Триггеры:

- fn_audit_log — аудит INSERT/UPDATE/DELETE для ключевых таблиц.
- fn_update_project_metric_summary — поддержка агрегатов в project_metric_summary.

Аудит реализован как универсальная trigger-function, которая записывает старые и новые значения в JSONB. Агрегирующий триггер пересчитывает лучшие значения метрик по проекту с учётом цели (min/max/last).

2.9. API и взаимодействие с БД

API предоставляет CRUD для всех сущностей и отчётные endpoints. Отчёты вызывают функции и представления (leaderboard, best-run, dashboard). Batch-import логирует ошибки в batch_import_errors и сохраняет статистику в batch_import_jobs. Документация доступна через Swagger/ OpenAPI. Авторизация реализована по JWT (OAuth2 password flow), токен передаётся в заголовке Authorization: Bearer <token>.

2.10. Примеры бизнес-запросов

Ниже приведены примеры SQL-запросов из sql/business_queries.sql.

```
WITH final_metrics AS (
    SELECT r.experiment_id, r.run_id, r.run_name, rmv.value, md.goal,
           ROW_NUMBER() OVER (PARTITION BY r.experiment_id
             ORDER BY CASE WHEN md.goal='min' THEN rmv.value END ASC,
                      CASE WHEN md.goal='max' THEN rmv.value END DESC,
                      CASE WHEN md.goal='last' THEN rmv.recorded_at END DESC) AS rn,
           AVG(rmv.value) OVER (PARTITION BY r.experiment_id) AS avg_value
    FROM runs r
    JOIN run_metric_values rmv ON rmv.run_id = r.run_id AND rmv.step IS NULL
    JOIN metric_definitions md ON md.metric_id = rmv.metric_id
    WHERE md.key = :metric_key AND rmv.scope = :scope
)
SELECT experiment_id, run_id, run_name, value AS best_value, avg_value,
       value − avg_value AS delta_vs_avg
FROM final_metrics
WHERE rn = 1;
```

Листинг 1: Лидерборд по эксперименту с дельтой к среднему

```
SELECT dv.dataset_id,
       dv.dataset_version_id,
       dv.version_label,
       AVG(rmv.value) AS avg_metric_value,
       COUNT(DISTINCT r.run_id) AS runs_count
FROM dataset_versions dv
JOIN runs r ON r.dataset_version_id = dv.dataset_version_id
JOIN run_metric_values rmv ON rmv.run_id = r.run_id AND rmv.step IS NULL
JOIN metric_definitions md ON md.metric_id = rmv.metric_id
WHERE md.key = :metric_key AND rmv.scope = :scope
GROUP BY dv.dataset_id, dv.dataset_version_id, dv.version_label
ORDER BY dv.version_label;
```

Листинг 2: Тренд метрик по версиям датасета

3. Технологическая часть

3.1. Контейнеризация и запуск

Проект разворачивается через Docker Compose. Backend автоматически применяет миграции Alembic при старте. Последовательность запуска:

1. docker compose up --build
2. docker compose exec backend python scripts/seed.py
3. Открыть Swagger: /docs

3.2. Интерфейс пользователя

Полноценный фронтенд не является обязательным, однако для демонстрации реализован лёгкий интерфейс на Vite + React. Он отображает обзор по проекту, лидерборд, список запусков и сведения о batch-import. Основной демонстрационный интерфейс остаётся Swagger, где доступны все CRUD-операции и отчёты.

3.3. Тестирование и устойчивость

В ходе проверки выполнены:

- загрузка seed-данных (1000 runs, 44000 metric values);
- проверка аудита: в audit_log сформировано 3005 записей после seed-загрузки;
- вызов функций: fn_best_run_id возвращает корректный run_id для выбранного эксперимента;
- проверка отчётов и производительности (см. приложение с анализом производительности);
- проверка авторизации и ролей: доступ к данным ограничен RBAC, Swagger использует OAuth2.

Ошибки одиночных строк в batch-import фиксируются без остановки всего job, что обеспечивает устойчивость обработки.

4. Оптимизация и производительность

Созданы индексы для полей в WHERE/JOIN/ORDER BY, включая частичный индекс для финальных метрик (ix_rmv_final_metric). Планы выполнения запросов и результаты измерений приведены в приложении и демонстрируют улучшение производительности до/после индексов. Каждый запрос запускался 3 раза, в таблице указана медиана времени выполнения. Перед сериями выполнялись CHECKPOINT и DISCARD ALL, что снижает влияние кэшей на измерения. Тесты проводились на seed-наборе данных

(1000 runs, 44000 metric values). Для сравнения временно удаляются индексы, затем выполняются те же запросы после их создания.

Ниже приведено сравнение времени выполнения двух ключевых запросов (данные получены на seed-наборе):

Запрос	**До индексов, ms**	**После индексов, ms**	**Коэф.**
Лидерборд эксперимента (fn_experiment_leaderboard)	2.208	0.267	8.3x
Тренд по версиям датасетов (avg финальных метрик)	1.990	0.945	2.1x

Таблица 2: Сравнение времени выполнения ключевых запросов

Заключение

Система реализует полноценную реляционную модель для управления ML-экспериментами, поддерживает аудит, batch-import, аналитические отчёты и демонстрацию производительности. Решение готово к защите и соответствует требованиям курсовой работы.

Приложения

Приложение А. ER-диаграмма

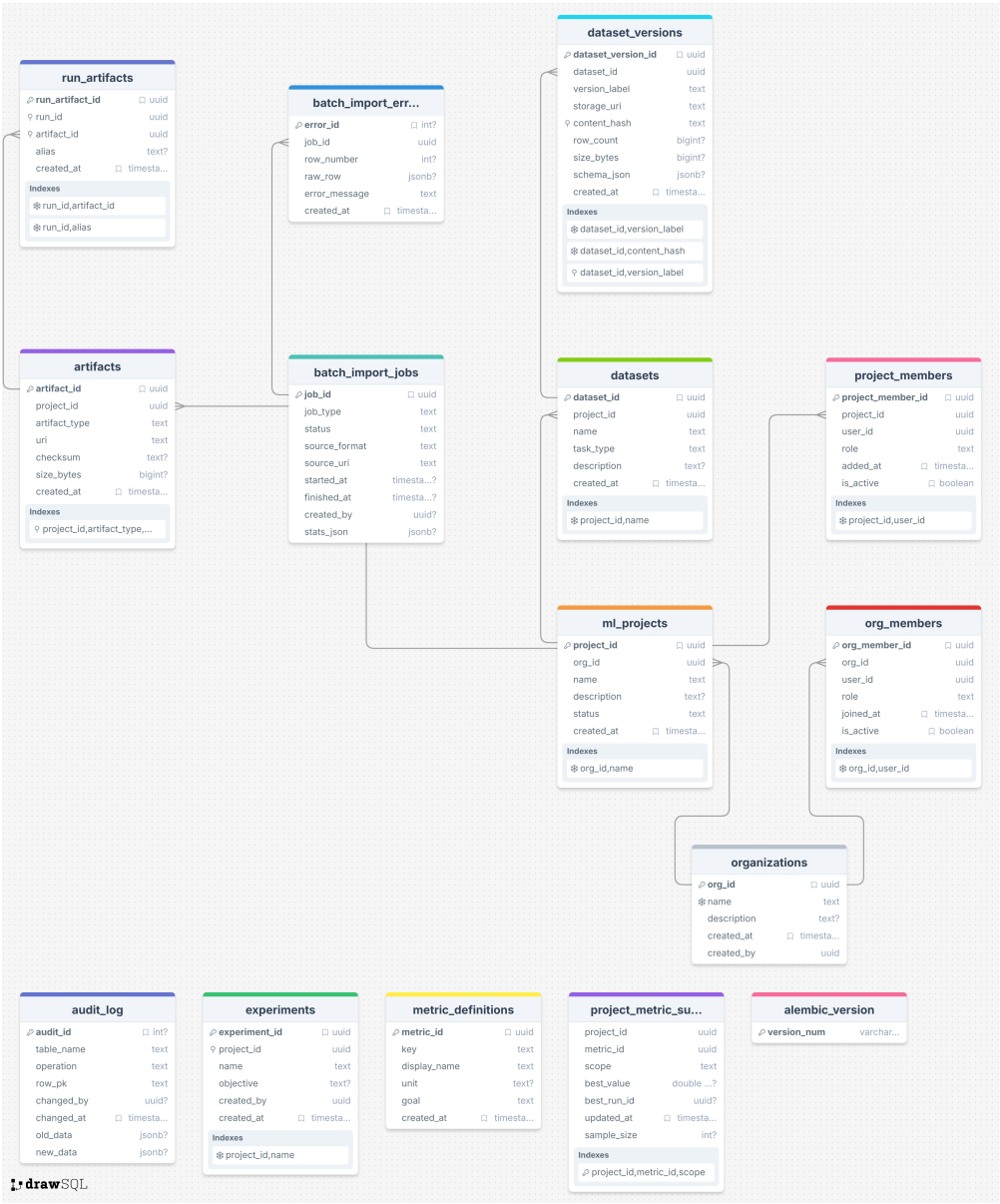

Рис. 1: ER-диаграмма базы данных (DrawSQL)

Приложение Б. Список основных endpoints

- CRUD:
  - /api/users
  - /api/orgs
  - /api/org-members

  - /api/projects
  - /api/project-members
  - /api/datasets
  - /api/dataset-versions
  - /api/experiments
  - /api/runs
  - /api/run-configs
  - /api/run-metric-values
  - /api/metric-definitions
  - /api/artifacts
  - /api/run-artifacts
  - /api/audit-log
  - /api/batch-import-jobs
  - /api/batch-import-errors
  - /api/project-metric-summary
- Отчёты:
  - /api/reports/experiments/{experiment_id}/leaderboard
  - /api/reports/experiments/{experiment_id}/best-run
  - /api/reports/projects/{project_id}/dashboard
- Авторизация:
  - /api/auth/register
  - /api/auth/login
  - /api/auth/token

Приложение B. Пример использования API

В примере используются значения из .env: SEED_TEST_USER_EMAIL и SEED_TEST_USER_PASSWORD.

```
export API_URL=http://localhost:8000
export API_EMAIL=your@email
export API_PASSWORD=your_password
export NO_PROXY=localhost,127.0.0.1

TOKEN=$(curl −s −X POST "$API_URL/api/auth/token" \
  −H "Content−Type: application/x−www−form−urlencoded" \
  −d "username=$API_EMAIL&password=$API_PASSWORD" | \
```

```
  python3 −c "import sys, json; print(json.load(sys.stdin)['access_token'])")

PROJECT_ID=$(curl −s −H "Authorization: Bearer $TOKEN" \
  "$API_URL/api/projects" | \
  python3 −c "import sys, json; print(json.load(sys.stdin)[0]['project_id'])")

curl −s −H "Authorization: Bearer $TOKEN" \
  "$API_URL/api/reports/projects/$PROJECT_ID/dashboard"

EXP_ID=$(curl −s −H "Authorization: Bearer $TOKEN" \
  "$API_URL/api/experiments" | \
  python3 −c "import sys, json; print(json.load(sys.stdin)[0]['experiment_id'])")

curl −s −H "Authorization: Bearer $TOKEN" \
  "$API_URL/api/reports/experiments/$EXP_ID/leaderboard?metric_key=accuracy&scope=val&limit=3"
```

Листинг 3: Получение токена и вызов отчётов

Приложение Г. Анализ производительности запросов

В приложении зафиксированы условия и сценарии проверки производительности:

- два ключевых запроса: лидерборд эксперимента и тренд по версиям датасетов;
- замеры до и после индексации, по 3 прогона каждого запроса;
- среда измерений: seed-набор (1000 runs, 44000 metric values), сброс кэшей через CHECKPOINT и DISCARD ALL.

Лидерборд: до индексов (прогон 2)

```
QUERY PLAN
──────────────────────────────────────────────────────────────────────
 Function Scan on fn_experiment_leaderboard (cost=3.75..13.75 rows=1000 width=96) (actual time=2.066..2.067 rows=10 loops=1)
   Buffers: shared hit=3575
   InitPlan 1 (returns $0)
     −> Limit (cost=3.50..3.50 rows=1 width=24) (actual time=0.021..0.021 rows=1 loops=1)
           Buffers: shared hit=2
           −> Sort (cost=3.50..3.75 rows=100 width=24) (actual time=0.020..0.020 rows=1 loops=1)
                 Sort Key: experiments.created_at
                 Sort Method: top−N heapsort Memory: 25kB
                 Buffers: shared hit=2
                 −> Seq Scan on experiments (cost=0.00..3.00 rows=100 width=24) (actual time=0.005..0.011 rows=100 loops=1)
                       Buffers: shared hit=2
 Planning Time: 0.066 ms
 Execution Time: 2.078 ms
```

Лидерборд: после индексов (прогон 2)

```
QUERY PLAN
──────────────────────────────────────────────────────────────────────
 Function Scan on fn_experiment_leaderboard (cost=3.75..13.75 rows=1000 width=96) (actual time=0.257..0.258 rows=10 loops=1)
```

```
  Buffers: shared hit=63
  InitPlan 1 (returns $0)
    -> Limit (cost=3.50..3.50 rows=1 width=24) (actual time=0.018..0.018 rows=1 loops=1)
          Buffers: shared hit=2
          -> Sort (cost=3.50..3.75 rows=100 width=24) (actual time=0.017..0.017 rows=1 loops=1)
                Sort Key: experiments.created_at
                Sort Method: top-N heapsort Memory: 25kB
                Buffers: shared hit=2
                -> Seq Scan on experiments (cost=0.00..3.00 rows=100 width=24) (actual time=0.004..0.009 rows=100 loops=1)
                      Buffers: shared hit=2
Planning Time: 0.033 ms
Execution Time: 0.267 ms
```

Тренд по версиям датасета: до индексов (прогон 2)

```
QUERY PLAN
------------------------------------------------------------------------------
 Limit (cost=1314.15..1314.20 rows=20 width=51) (actual time=2.192..2.196 rows=20 loops=1)
   Buffers: shared hit=589
   InitPlan 1 (returns $0)
     -> Limit (cost=0.15..8.17 rows=1 width=16) (actual time=0.004..0.004 rows=1 loops=1)
           Buffers: shared hit=2
           -> Index Scan using metric_definitions_key_key on metric_definitions (cost=0.15..8.17 rows=1 width=16) (actual time=0.004..0.004
                 rows=1 loops=1)
                 Index Cond: (key = 'accuracy'::text)
                 Buffers: shared hit=2
   -> Sort (cost=1305.98..1307.36 rows=551 width=51) (actual time=2.191..2.194 rows=20 loops=1)
         Sort Key: (avg(rmv.value)) DESC
         Sort Method: top-N heapsort Memory: 29kB
         Buffers: shared hit=589
         -> HashAggregate (cost=1284.43..1291.32 rows=551 width=51) (actual time=2.068..2.125 rows=826 loops=1)
               Group Key: e.project_id, dv.dataset_version_id
               Batches: 1 Memory Usage: 297kB
               Buffers: shared hit=589
               -> Hash Join (cost=51.45..1278.92 rows=551 width=43) (actual time=0.168..1.922 rows=1000 loops=1)
                     Hash Cond: (r.experiment_id = e.experiment_id)
                     Buffers: shared hit=589
                     -> Hash Join (cost=47.20..1273.16 rows=551 width=43) (actual time=0.152..1.826 rows=1000 loops=1)
                           Hash Cond: (r.dataset_version_id = dv.dataset_version_id)
                           Buffers: shared hit=587
                           -> Hash Join (cost=40.50..1264.96 rows=551 width=40) (actual time=0.135..1.533 rows=1000 loops=1)
                                 Hash Cond: (rmv.run_id = r.run_id)
                                 Buffers: shared hit=583
                                 -> Seq Scan on run_metric_values rmv (cost=0.00..1223.00 rows=551 width=24) (actual time=0.006..1.304
                                       rows=1000 loops=1)
                                       Filter: ((step IS NULL) AND (metric_id = $0) AND (scope = 'val'::text))
                                       Rows Removed by Filter: 43000
                                       Buffers: shared hit=565
                                 -> Hash (cost=28.00..28.00 rows=1000 width=48) (actual time=0.127..0.128 rows=1000 loops=1)
                                       Buckets: 1024 Batches: 1 Memory Usage: 87kB
                                       Buffers: shared hit=18
                                       -> Seq Scan on runs r (cost=0.00..28.00 rows=1000 width=48) (actual time=0.002..0.059 rows=1000
                                             loops=1)
                                             Buffers: shared hit=18
                           -> Hash (cost=5.20..5.20 rows=120 width=19) (actual time=0.016..0.017 rows=120 loops=1)
                                 Buckets: 1024 Batches: 1 Memory Usage: 14kB
                                 Buffers: shared hit=4
```

```
                                    -> Seq Scan on dataset_versions dv (cost=0.00..5.20 rows=120 width=19) (actual time=0.001..0.009 rows
                                        =120 loops=1)
                                        Buffers: shared hit=4
                            -> Hash (cost=3.00..3.00 rows=100 width=32) (actual time=0.014..0.015 rows=100 loops=1)
                                    Buckets: 1024 Batches: 1 Memory Usage: 15kB
                                    Buffers: shared hit=2
                                    -> Seq Scan on experiments e (cost=0.00..3.00 rows=100 width=32) (actual time=0.002..0.007 rows=100 loops=1)
                                        Buffers: shared hit=2
Planning:
  Buffers: shared hit=17
Planning Time: 0.239 ms
Execution Time: 2.228 ms
```

Тренд по версиям датасета: после индексов (прогон 2)

```
QUERY PLAN
------------------------------------------------------------------------
 Limit (cost=703.52..703.57 rows=20 width=51) (actual time=0.822..0.825 rows=20 loops=1)
   Buffers: shared hit=84
   InitPlan 1 (returns $0)
     -> Limit (cost=0.15..8.17 rows=1 width=16) (actual time=0.004..0.005 rows=1 loops=1)
          Buffers: shared hit=2
          -> Index Scan using metric_definitions_key_key on metric_definitions (cost=0.15..8.17 rows=1 width=16) (actual time=0.004..0.004
              rows=1 loops=1)
              Index Cond: (key = 'accuracy'::text)
              Buffers: shared hit=2
   -> Sort (cost=695.35..696.75 rows=559 width=51) (actual time=0.821..0.823 rows=20 loops=1)
        Sort Key: (avg(rmv.value)) DESC
        Sort Method: top-N heapsort Memory: 29kB
        Buffers: shared hit=84
        -> HashAggregate (cost=673.49..680.48 rows=559 width=51) (actual time=0.681..0.754 rows=826 loops=1)
             Group Key: e.project_id, dv.dataset_version_id
             Batches: 1 Memory Usage: 297kB
             Buffers: shared hit=84
             -> Hash Join (cost=73.46..667.90 rows=559 width=43) (actual time=0.207..0.550 rows=1000 loops=1)
                  Hash Cond: (r.experiment_id = e.experiment_id)
                  Buffers: shared hit=84
                  -> Hash Join (cost=69.21..662.12 rows=559 width=43) (actual time=0.191..0.457 rows=1000 loops=1)
                       Hash Cond: (r.dataset_version_id = dv.dataset_version_id)
                       Buffers: shared hit=82
                       -> Hash Join (cost=62.51..653.89 rows=559 width=40) (actual time=0.173..0.357 rows=1000 loops=1)
                            Hash Cond: (rmv.run_id = r.run_id)
                            Buffers: shared hit=78
                            -> Bitmap Heap Scan on run_metric_values rmv (cost=22.01..611.91 rows=559 width=24) (actual time
                                =0.042..0.126 rows=1000 loops=1)
                                 Recheck Cond: ((metric_id = $0) AND (scope = 'val'::text) AND (step IS NULL))
                                 Heap Blocks: exact=50
                                 Buffers: shared hit=60
                                 -> Bitmap Index Scan on ix_rmv_final_metric (cost=0.00..21.87 rows=559 width=0) (actual time
                                     =0.036..0.036 rows=1000 loops=1)
                                     Index Cond: ((metric_id = $0) AND (scope = 'val'::text))
                                     Buffers: shared hit=10
                            -> Hash (cost=28.00..28.00 rows=1000 width=48) (actual time=0.129..0.129 rows=1000 loops=1)
                                 Buckets: 1024 Batches: 1 Memory Usage: 87kB
                                 Buffers: shared hit=18
                                 -> Seq Scan on runs r (cost=0.00..28.00 rows=1000 width=48) (actual time=0.002..0.061 rows=1000
                                     loops=1)
                                     Buffers: shared hit=18
```

```
                        -> Hash (cost=5.20..5.20 rows=120 width=19) (actual time=0.016..0.016 rows=120 loops=1)
                              Buckets: 1024 Batches: 1 Memory Usage: 14kB
                              Buffers: shared hit=4
                              -> Seq Scan on dataset_versions dv (cost=0.00..5.20 rows=120 width=19) (actual time=0.001..0.009 rows
                                    =120 loops=1)
                                    Buffers: shared hit=4
                  -> Hash (cost=3.00..3.00 rows=100 width=32) (actual time=0.015..0.015 rows=100 loops=1)
                        Buckets: 1024 Batches: 1 Memory Usage: 15kB
                        Buffers: shared hit=2
                        -> Seq Scan on experiments e (cost=0.00..3.00 rows=100 width=32) (actual time=0.003..0.008 rows=100 loops=1)
                              Buffers: shared hit=2
Planning:
  Buffers: shared hit=31
Planning Time: 0.372 ms
Execution Time: 0.856 ms
```

Полные планы выполнения можно получить командой psql -f sql/perf_demo.sql или скриптом scripts/run_perf_demo.sh.

Приложение Д. Ссылки на материалы

- https://github.com/L0ckR/DB_Course_MAI_2025