
Dota 2 The International
Building a Data Warehouse
from Raw CSV Data
Аналитика качества данных в небольшом DWH прототипе
The International

Описание задачи проекта

Цель: **Построить аналитическое хранилище (DWH) для анализа турниров The International по Dota 2 за 10 лет**

Постановка задачи: **построить воспроизводимую ETL-цепочку, нормализовать данные, загрузить их в базу и создать аналитические витрины.**

Задачи:

1. Собрать и унифицировать: данные из разнородных источников (разные форматы CSV за разные годы)

2. Очистить и нормализовать: привести к единой схеме, исправить типы, обработать пропуски

3. Обеспечить качество: внедрить автоматические проверки данных на всех этапах

4. Спроектировать DWH: реализовать классическую архитектуру Stage → DDS → DM

5. Подготовить аналитические витрины: для ответов на бизнес-вопросы

Бизнес-вопросы, на которые отвечает проект

- Какие команды самые успешные в истории TI?

- Как менялись призовые по годам?

- Игроки из каких стран заработали больше всего?

- Какие герои были в мете в разные годы?

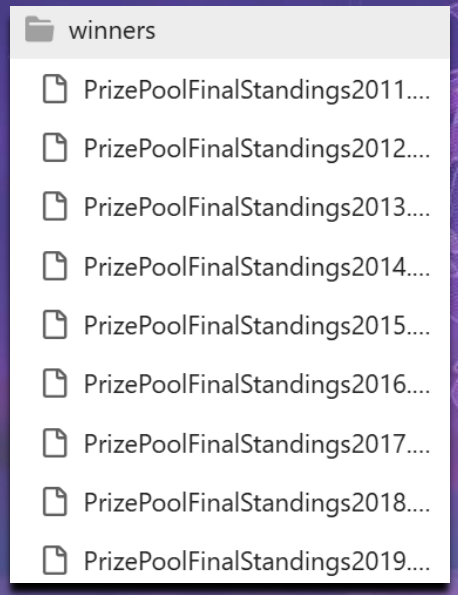
- Как менялись составы команд-победителей?

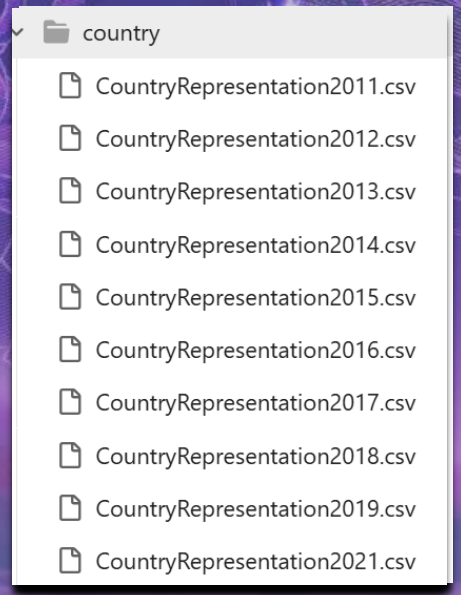
Источники данных и их "сырые" проблемы

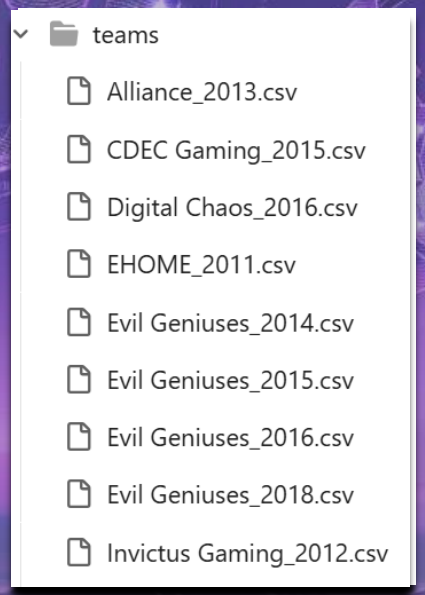
Данные с Kaggle: DOTA 2 The International Dataset (2011–2021)

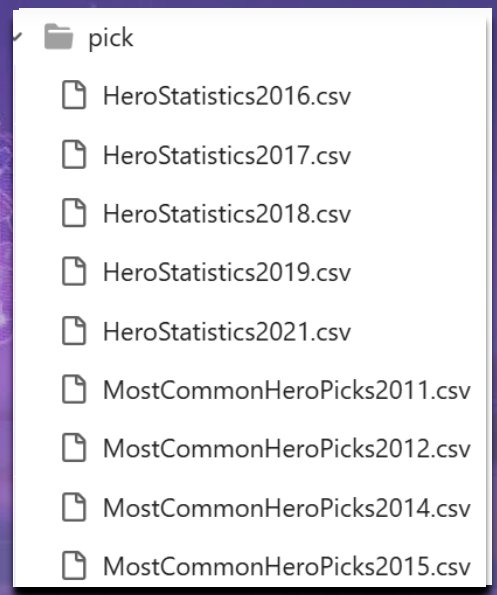
Задача: объединить разрозненные CSV файлы, каждый год в разных форматах

Сущность	Формат	Проблемы
Победители (winners)	PrizePoolFinalStandings{год}.csv	Разные названия колонок: 'Team' / 'Team Name', '$USD' / 'Price'
Страны (country)	CountryRepresentation{год}.csv	Игроки записаны через запятую в одной ячейке
Пики героев (picks)	MostCommonHeroPicks{год}.csv (2011-2015) HeroStatistics{год}.csv (2016-2021)	Два разных формата исходных данных Разные названия колонок: 'Number of times picked' / 'Times Picked'
Составы команд (teams)	{команда}_{год}.csv	Техническая колонка 'Unnamed: 0' на самом деле хранит позицию игрока









ETL-процесс (Jupyter Notebook)

Extract: загрузка CSV из GitHub

Данные автоматически загружаются из GitHub-репозитория.

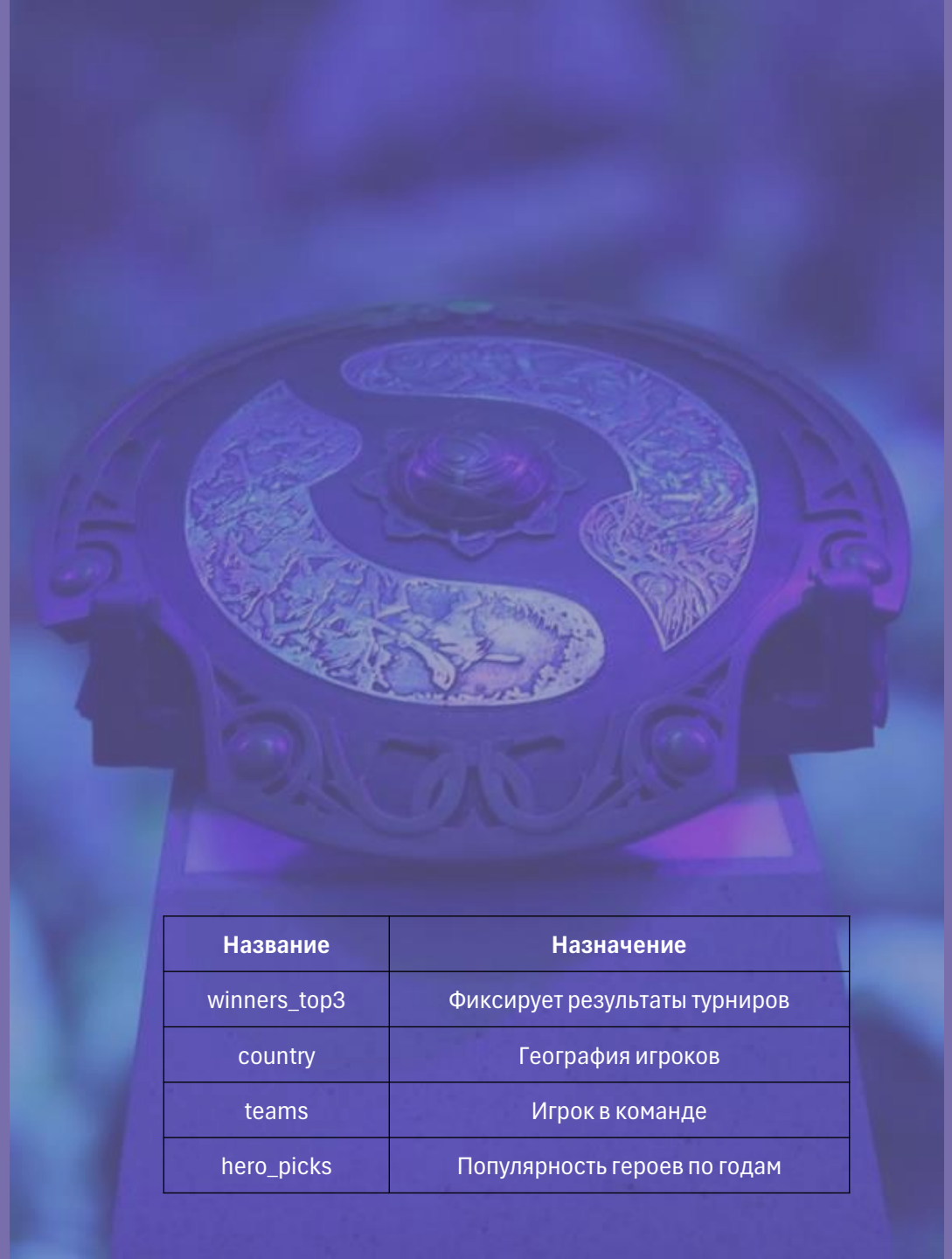
Отдельные CSV для каждого года.

Разные источники:

- результаты турнира,
- игроки по странам,
- составы команд,
- статистика пиков героев.

Что сделано:

- реализована итерационная загрузка файлов по годам;
- каждому набору данных добавлен признак year;
- данные объединены в единые датафреймы.



Название	Назначение
winners_top3	Фиксирует результаты турниров
country	География игроков
teams	Игрок в команде
hero_picks	Популярность героев по годам

ETL-процесс (Jupyter Notebook)

Transform: Преобразование данных

На этапе трансформации выполнена стандартизация и очистка:

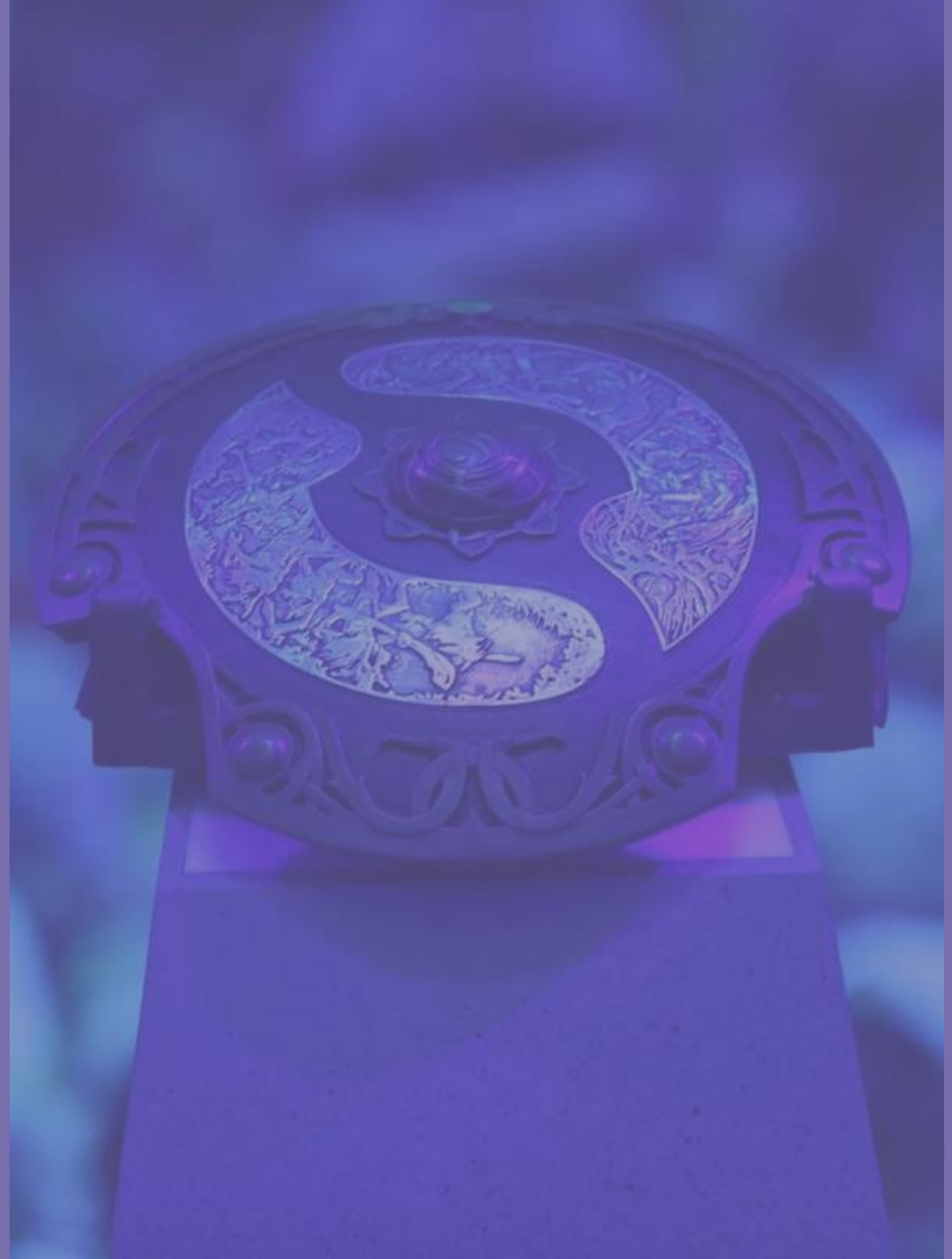
Нормализация структуры (унифицированы названия колонок; приведена схема данных разных лет к единому формату)

Очистка значений (удалены валютные символы $, %, ,; строки преобразованы в числовые типы)

Нормализация сущностей (списки игроков разделены на отдельные строки; устранены дубликаты; проверена уникальность ключей)

Контроль качества данных (проверены пропуски (NULL); проверены типы данных; выполнена базовая статистическая проверка)

Данные стали пригодны для аналитической модели.



Source Data Problems & Data Cleaning

Проблема 1: Отсутствие контекста (год, команда)

В исходных файлах отсутствуют колонки с годом, с названием команды.
Это делает аналитику невозможной: без года нельзя увидеть динамику, без команды нет понимания, кто есть кто

Решение: добавили год и название команды при загрузке из метаданных

Добавляем год из метаданных при загрузке

df['year'] = row['year']

Все записи теперь привязаны к контексту - мы знаем, к какому турниру относится каждая строка, получена корректная структура

```
country_raw = [] # Создаем пустой список для сбора "сырых" данных за все года

for _, row in metadata_df.iterrows():
    year = row['year']
    url = row['country_csv']

    df = pd.read_csv(url)
    df['year'] = year  # добавляем год для дальнейшей аналитики
```

```
# Базовый URL, где хранятся CSV-файлы с составами команд
base_url = "https://raw.githubusercontent.com/alexandraryzvanova-sketch/DWH_quality/main/data/raw/teams/"

dfs = [] # Создаем пустой список для сбора данных

# Проходим по всем уникальным командам из таблицы победителей
# .drop_duplicates() убирает повторы (одна команда может побеждать несколько раз)
for _, row in winners_top3[['team_name', 'year']].drop_duplicates().iterrows():
    team = row['team_name'] # Название команды
    year = row['year'] # Год турнира

    # Заменяем пробелы в названии команды на %20 для корректного URL
    # URL-кодирование: пробел в интернете пишется как %20
    team_url = team.replace(' ', '%20')
    # Формируем полный URL: базовый адрес + команда + _ + год + .csv
    url = f"{base_url}{team_url}_{year}.csv"

    df = pd.read_csv(url) # Загружаем CSV-файл с составом команды
    df['team_name'] = team
    df['year'] = year # Добавляем колонки с названием команды и годом
```

Source Data Problems & Data Cleaning

Проблема 2: Разные названия одних и тех же колонок

Одни и те же колонки назывались по-разному в разные года. Это делало невозможным автоматическое объединение данных.

Решение: функции normalize_columns() проверяет все варианты и приводит к единому стандарту.
Таким образом, данные разных лет становятся совместимыми.

Сущность	Годы	Было	Стало
winners_top3	2011-2021	'Team' / 'Team Name'	`team_name`
winners_top3	2011-2021	'$USD' / 'Price'	`prize_usd`
winners_top3	2011-2021	'PricePoolPercent' / 'Pcnt' / 'Percent'	`prize_percent`
hero_picks	2011-2015	Number of times picked' / 'Times Picked'	`times_picked`
hero_picks	2016-2021	'Times Picked'	`times_picked`
teams	2011-2021	'Player Names' / 'Players'	`player_name`

```python
def normalize_columns(df):

    """
    Функция для приведения названий колонок к единому формату

    Аргументы:
        df: исходный датафрейм с "сырыми" данными за конкретный год

    Возвращает:
        df: датафрейм с приведенными к единому стандарту названиями колонок

    Принцип работы:
    Проверяем, какие названия колонок есть в исходном файле, и создаем новые
    колонки со стандартизированными именами: team_name, prize_usd, prize_percent
    """

    df = df.copy() # Создаем копию датафрейма, чтобы не изменять оригинальные данные

    if 'Team Name' in df.columns:
        df['team_name'] = df['Team Name']
    elif 'Team' in df.columns:
        df['team_name'] = df['Team']

    if '$USD' in df.columns:
        df['prize_usd'] = df['$USD']
    elif 'Price' in df.columns:
        df['prize_usd'] = df['Price']

    if 'PricePoolPercent' in df.columns:
        df['prize_percent'] = df['PricePoolPercent']
    elif 'Pcnt' in df.columns:
        df['prize_percent'] = df['Pcnt']
    elif 'Percent' in df.columns:
        df['prize_percent'] = df['Percent']

    return df
```

Source Data Problems & Data Cleaning

Проблема 3: Несогласованные форматы значений

Числовые показатели хранились как строки, валюта содержала символ $ и разделители , проценты содержали %
Из-за этого показатели нельзя было агрегировать или анализировать.

Решение:
Выполнена очистка значений от служебных символов и последующее приведение типов к числовому формату (numeric).

Это обеспечило корректность дальнейшего анализа.

```
# ОЧИСТКА КОЛОНКИ С ПРИЗОВЫМИ (prize_usd)
# Удаление знака доллара и запятых
winners_top3['prize_usd'] = winners_top3['prize_usd'].astype(str).str.replace('[$,]', '', regex=True)
# ОЧИСТКА КОЛОНКИ С ПРОЦЕНТАМИ (prize_percent)
# # Удаление знака процента и запятых
winners_top3['prize_percent'] = winners_top3['prize_percent'].astype(str).str.replace('[%,]', '', regex=True)

# ПРЕОБРАЗОВАНИЕ В ЧИСЛОВЫЕ ТИПЫ
winners_top3['prize_usd'] = pd.to_numeric(winners_top3['prize_usd'])
winners_top3['prize_percent'] = pd.to_numeric(winners_top3['prize_percent'])
```

Source Data Problems & Data Cleaning

Проблема 4: Мульти-значения в одной ячейке

В таблицах стран список игроков хранился одной строкой:

Player1, Player2, Player3, Player4, Player5

Это нарушает принципы нормализации данных и не позволяет строить связи.

Решение:
1. Разбили строку в список: .str.split(',')

2. Разобрали в отдельные строки: .explode('player_name')

3. Почистили пробелы: `.str.strip()`

В результате была получена корректная структура уровня игрока.

```
# Разбиваем строку с игроками в список
country_df['player_name'] = country_df['Players'].str.split(',')

# Превращаем список в строки (explode)
country_exploded_df = country_df.explode('player_name')
```

Source Data Problems & Data Cleaning

Проблема 5: Технические колонки как бизнес-данные

В исходных файлах состава команд (teams) была колонка `'Unnamed: 0'`, являющаяся не просто техническим индексом, а техническим столбцом позиции члена команды, добавляем роль player-other

Решение:

1. Переименовали в team_position

2. Превратили в бизнес-атрибут

В результате была получена корректная структура уровня игрока.

```
teams_full_df = teams_full_df.rename(
    columns={'Unnamed: 0': 'team_position'}
)
```

```
# Применяем функцию lambda к каждой строке колонки team_position
# x in [0,1,2,3,4] проверяет, является ли позиция игровой
teams_full_df['role'] = teams_full_df['team_position'].apply(
    lambda x: 'player' if x in [0, 1, 2, 3, 4] else 'other'
)
```

Source Data Problems & Data Cleaning

Проблема 6: Данные хранятся в отдельных файлах по годам

Каждый турнир представлен отдельным CSV-файлом, поэтому данные невозможно анализировать как единую таблицу.

Данные по пикам героев представлены в двух разных форматах, зависящих от года проведения турнира.

Решение:
Был сформирован список годов проведения турнира и реализована пакетная загрузка файлов с добавлением признака year. Данные по пикам для корректной обработки разделяются на OLD и NEW группы.

Это позволило объединить разрозненные источники в единые датасеты.

```
# Создам таблицу с годами, в которые проходили соревнования, (в 2020 был ковид, не было инта)

years_df = pd.DataFrame({
    'year': [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2021]
})
```

Для корректной обработки данные разделяются на OLD и NEW группы.

```
    # Годы с разными форматами данных
    old_years = [2011, 2012, 2014, 2015]
    new_years = [2016, 2017, 2018, 2019, 2021]
```

ПРОВЕРКА КАЧЕСТВА ДАННЫХ (ЭТАП DATA QUALITY)

```python
def validate_teams(df):
    """
    Проверка качества данных о составах команд.

    Проверяем:
    1. Отсутствие NULL в ключевых полях
    2. Уникальность записей (один игрок в команде в год - одна запись)
    3. Бизнес-правило: в команде минимум 5 игроков
    """

    errors = []


    # 1. Отсутствие NULL
    if df['player_name'].isna().any():
        errors.append("player_name contains NULLs")

    if df['team_name'].isna().any():
        errors.append("team_name contains NULLs")

    if df['year'].isna().any():
        errors.append("year contains NULLs")
    # 2. Проверка гранулярности
    dup = df.duplicated(subset=['player_name', 'team_name', 'year']).sum()
    if dup > 0:
        errors.append(f"Duplicate roster rows detected: {dup}")

    # 3. Минимальный размер команды
    # В Dota 2 в команде минимум 5 игроков
    # Фильтруем только игроков (не тренеров) и считаем их по командам и годам

    roster_size = df[df['role'] == 'player'].groupby(['team_name', 'year']).size()

    # Находим команды, где игроков меньше 5
    bad_teams = roster_size[roster_size < 5]

    if not bad_teams.empty:
        errors.append(f"Teams with <5 players detected: {bad_teams.index.tolist()}")

    # Итог
    if errors:
        for err in errors:
            logger.error(err)
        raise ValueError("Team roster validation failed")

    logger.info("Teams validation passed")
```

1. Проверка на пропуски (NULL / пустые значения)

• Отсутствие NULL в ключевых полях (year, team_name, player_name, Hero, Country, Place)

2. Проверка типов данных

• Числовые поля (prize_usd, prize_percent, times_picked) должны иметь тип данных число, а не строка

• Корректное приведение после очистки от символов ($, %, запятые)

3. Бизнес-правила (смысловые ограничения)

• Призовые (prize_usd) и количество пиков (times_picked) не могут быть отрицательными

• Места (Place) могут быть только 1st, 2nd, 3rd

• Страна (Country) не может состоять из цифр

• В команде должно быть минимум 5 игроков (с ролью player)

4. Уникальность и гранулярность

• winners_top3: один год - одно место (year + Place)

• hero_picks: один год - один герой (year + Hero)

• teams: один игрок в одной команде в один год (player_name + team_name + year)

• country: один игрок из одной страны в один год (player_name + Country + year)

5. Полнота и логическая согласованность

• В каждом году должно быть ровно 3 записи в таблице победителей

• Если турнир проводился должны быть данные о пиках героев (сумма times_picked за год > 0)

6. Корректность трансформаций

• Отсутствие пустых строк и лишних пробелов (после explode, т.е. не должно быть player_name = "")

• Правильно определена роль участников (player / other) на основе позиции в команде

ETL-процесс (Jupyter Notebook)

Load Preparation (Подготовка к загрузке)

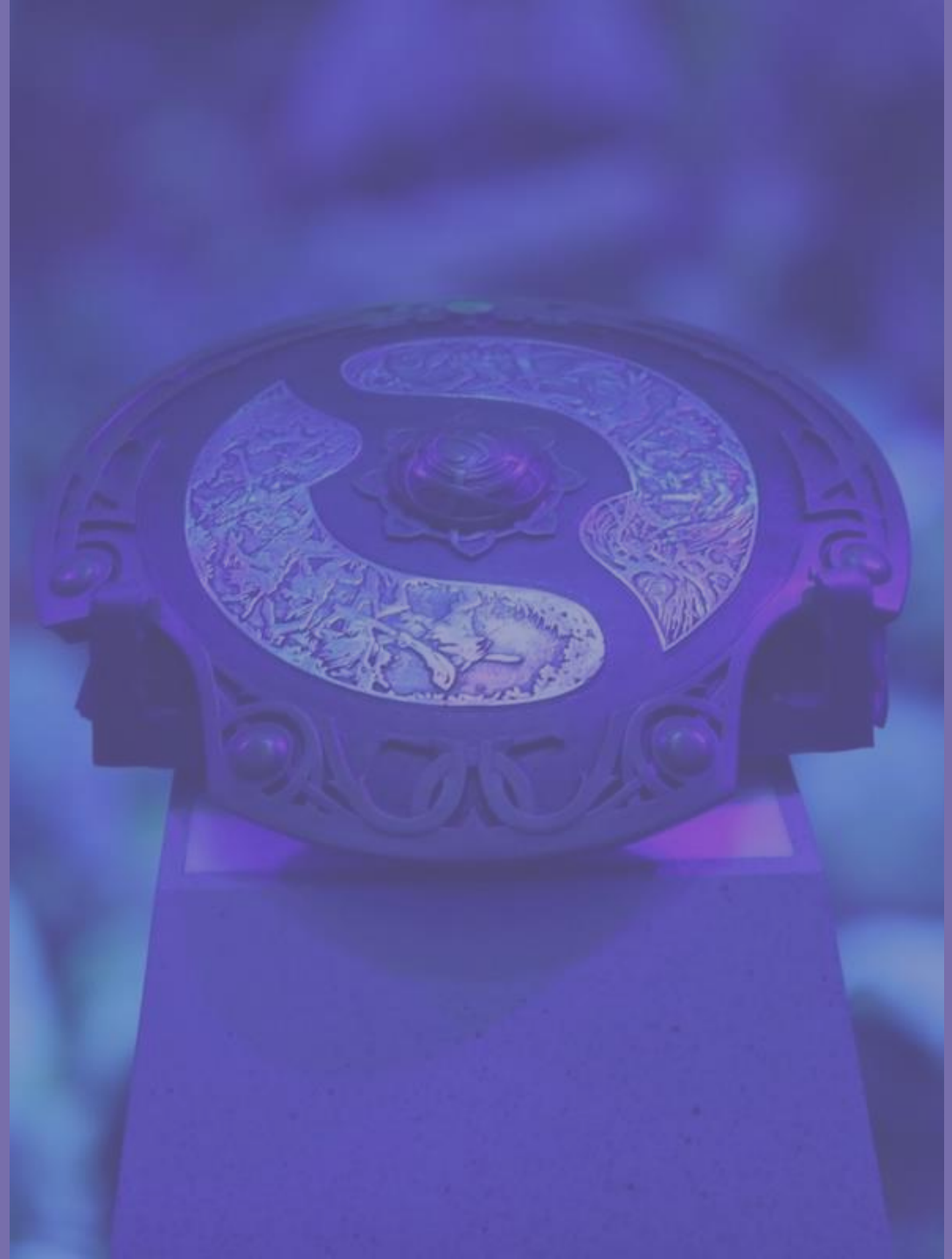
Очищенные данные сохраняются в:

output_path = "/content/"

После выгружаются на компьютер и сохраняются в репозиторий в: /data/processed/

В папке /data/processed/ появились файлы:

- winners_top3.csv
- country.csv
- hero_picks.csv
- teams.csv



SQL-процесс и слои DWH

После очистки и стандартизации данные были загружены в реляционную базу данных **PostgreSQL**, где реализована аналитическая модель типа **созвездие**, создана отдельная база данных **dota_dwh**.

Локальная PostgreSQL

Сервер: PostgreSQL, локально на компьютере

Подключение через Dbeaver

Назначение: хранение всех слоев DWH и построение витрин/фактов для аналитики

Слои DWH и их создание

Проект реализован по классической архитектуре Stage → DDS → DM → Marts

SQL-процесс и слой Stage

Назначение: сырой слой, содержит данные после первичной очистки в ETL, но до моделирования и нормализации в DDS, используется для изоляции источника от DWH-модели

Файлы в репозитории:

• sql/stage/create_stage_tables.sql — создание схемы и таблиц

• sql/stage/load_stage.md — инструкция по загрузке данных из CSV (data/processed/*.csv) вручную через DBeaver

Очищенные CSV из ETL-пайплайна сохраняются в папку /data/processed/

Далее они загружаются в PostgreSQL локально на компьютере с помощью DBeaver.

Схема загрузки:

Таблица Stage → CSV-файл

stage.winners_raw → winners_top3.csv

stage.teams_raw → teams.csv

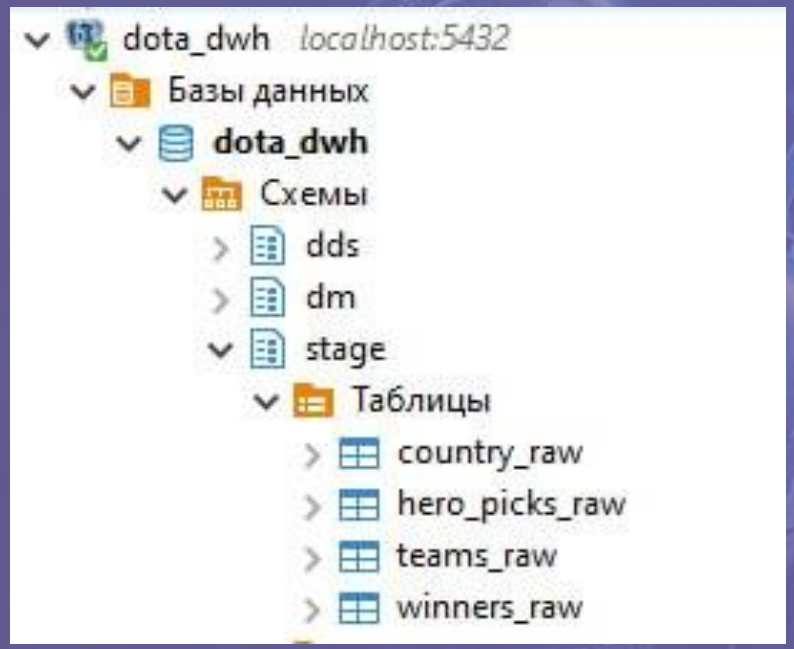
stage.country_raw → country.csv

stage.hero_picks_raw → hero_picks.csv

• load_stage.md инструкция как данные загружались в Stage вручную из очищенных CSV

SQL-процесс и слой Stage

В Stage данные хранятся в том виде, в котором они вышли из ETL-пайплайна, но еще не трансформировались под модель DWH. Это позволяет: перестраивать DDS без повторного запуска ETL, проверять качество данных до нормализации, всегда иметь доступ к оригиналу очищенных данных.




```sql
create schema if not exists stage;

create table if not exists stage.winners_raw (
    year int,
    place int,
    team_name text,
    prize_usd numeric,
    prize_percent numeric
);

create table if not exists stage.teams_raw (
    player_name text,
    team_name text,
    year int,
    role text
);

create table if not exists stage.country_raw (
    player_name text,
    country text,
    year int
);
```

Данные загружаются из подготовленных CSV (слой data/processed).

Так как PostgreSQL COPY читает файлы только с сервера, загрузка выполняется через клиент DBeaver.

Шаги загрузки

Для каждой таблицы выполнить:

1. Открыть DBeaver
2. Перейти в схему:

Database → Schemas → stage → Tables

3. Выбрать таблицу (например winners_raw)
4. Нажать ПКМ → Import Data
5. Выбрать формат CSV
6. Указать файл из папки проекта:

data/processed/winners_top3.csv

7. Проверить настройки: Delimiter: , Header: + Encoding: UTF-8
8. Finish

В моем проекте Stage содержит 4 таблицы, полностью соответствующие CSV-файлам из /data/processed/. Отсюда данные загружаются в DDS через SQL-скрипты с выделением уникальных значений.

SQL-процесс и слой DDS

Назначение: нормализованные таблицы измерений (dim_*), справочники

Слой DDS превращает 'сырые' данные из Stage в согласованные измерения. Устраняет дубликаты (каждое название хранится один раз), добавляет суррогатные ключи (ID) для быстрых JOIN, обеспечивает целостность данных (все факты ссылаются на одни и те же справочники), готовит данные для построения факт-таблиц в слое DM.

Файлы в репозитории:

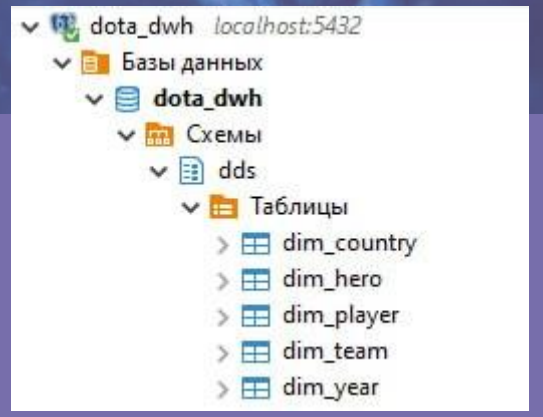
- sql/dds/create_dds_tables.sql — создание нормализованных таблиц
- sql/dds/load_dds_table.sql — загрузка данных в DDS из Stage

Измерения (dim_*):

Таблица	Ключ	Атрибуты
dim_year	year_id	year
dim_team	team_id	team_name
dim_player	player_id	player_name
dim_country	country_id	country_name
dim_hero	hero_id	hero_name

```sql
create schema if not exists dds;

--Измерение ГОД

create table if not exists dds.dim_year (
    year_id serial primary key,
    year int unique not null
);


--Измерение КОМАНДА

create table if not exists dds.dim_team (
    team_id serial primary key,
    team_name text unique not null
);
```

```sql
-- YEAR
insert into dds.dim_year (year)
select year from (
    select distinct year from stage.winners_raw
    union
    select distinct year from stage.teams_raw
    union
    select distinct year from stage.country_raw
    union
    select distinct year from stage.hero_picks_raw
) as all_years
order by year asc;
```



SQL-процесс и слой DM

Назначение: факты (fact_*) и аналитические витрины для построения дашбордов

В моем проекте:

4 факт-таблицы описывают все бизнес-процессы турниров

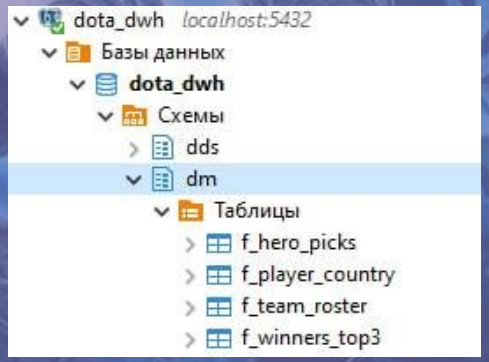
Схема созвездия позволяет строить сложные цепочки аналитики

4 витрины отвечают на ключевые вопросы: кто сколько заработал, какие герои популярны, кто чаще в топ-3

Все SQL-запросы для витрин лежат в репозитории и полностью воспроизводимы.

Файлы в репозитории:

- sql/dm/create_dm_tables.sql - создание таблиц фактов
- sql/dm/load_dm_facts.sql - загрузка фактов из DDS
- sql/dm/add_foreign_keys.sql - настройка внешних ключей и связей между таблицами

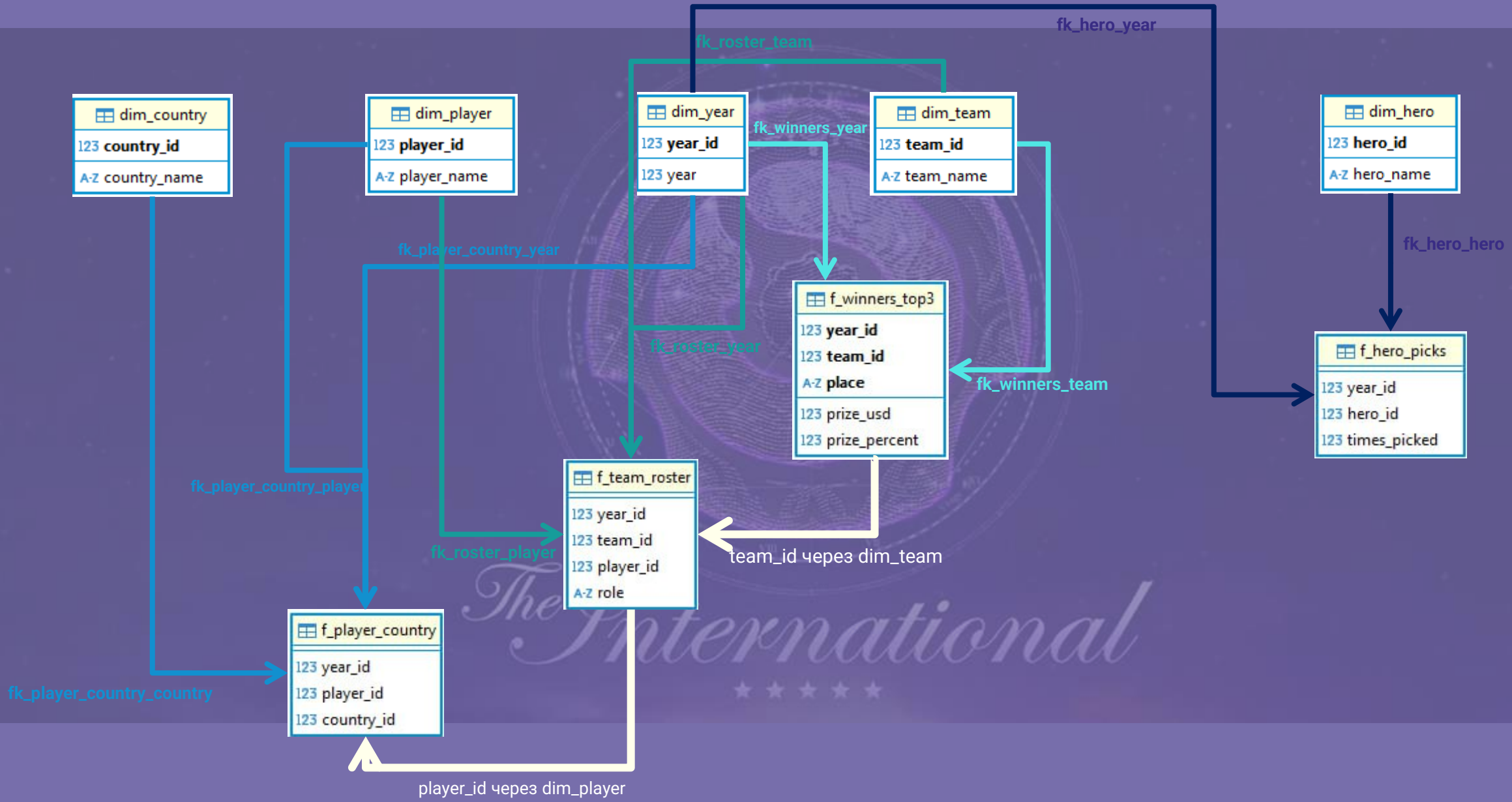



```sql
create schema if not exists dm;

-- Факт победителей
create table if not exists dm.f_winners_top3 (
    year_id int not null,
    team_id int not null,
    place varchar(10),
    prize_usd numeric,
    prize_percent numeric,
    primary key (year_id, team_id, place)
);
```

Таблица	Гранулярность	Ключи	Меры	Атрибуты
f_winners_top3	год + место	year_id, team_id	prize_usd, prize_percent	place
f_team_roster	год + команда + игрок	year_id, team_id, player_id	-	role
f_hero_picks	год + герой	year_id, hero_id	times_picked	-
f_player_country	год + игрок	year_id, player_id, country_id	-	-

```sql
-- f_winners_top3
-- Гранулярность: год + место, ключи: year_id, team_id

insert into dm.f_winners_top3 (year_id, team_id, place, prize_usd, prize_percent)
select
    y.year_id,
    t.team_id,
    s.place,
    s.prize_usd,
    s.prize_percent
from stage.winners_raw s
join dds.dim_year y on s.year = y.year
join dds.dim_team t on s.team_name = t.team_name;
```

Итоговая модель DWH, схема связей, созвездие



Итоговая модель DWH, схема связей, созвездие

Внешний ключ	Таблица	Куда ссылается	Что связывает
fk_winners_year	f_winners_top3	dim_year	Привязывает результаты к году
fk_winners_team	f_winners_top3	dim_team	Привязывает результаты к команде
fk_roster_year	f_team_roster	dim_year	Привязывает состав к году
fk_roster_team	f_team_roster	dim_team	Привязывает состав к команде
fk_roster_player	f_team_roster	dim_player	Привязывает состав к игроку
fk_hero_year	f_hero_picks	dim_year	Привязывает пики к году
fk_hero_hero	f_hero_picks	dim_hero	Привязывает пики к герою
fk_player_country_year	f_player_country	dim_year	Привязывает национальность к году
fk_player_country_player	f_player_country	dim_player	Привязывает национальность к игроку
fk_player_country_country	f_player_country	dim_country	Привязывает национальность к стране

Аналитические витрины

/sql/marts/ — аналитические витрины

Содержит SQL-запросы с готовыми агрегатами, отвечающими на ключевые бизнес-вопросы.

Витрины строятся на основе схемы dm (факты и измерения) и могут переиспользоваться в разных дашбордах.

Примеры:

mart_team_prize.sql — призовые по командам

mart_hero_popularity.sql — популярность героев по годам

/sql/dashboard_queries/ — запросы для дашбордов

Содержит специализированные SQL-запросы под конкретные визуализации в Yandex DataLens.

Каждый файл соответствует одному графику и содержит готовую агрегацию, сортировку и все необходимые поля.

Что включают: линейные графики динамики; карты с геокоординатами; таблицы; детальные составы с атрибутами

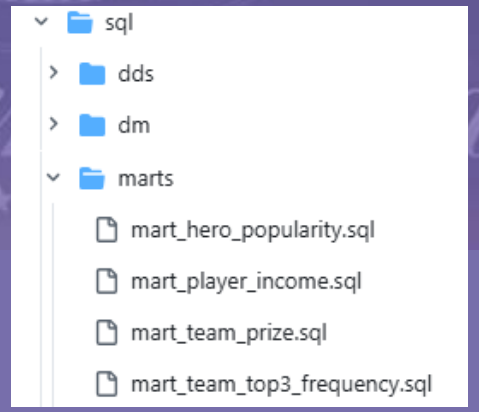
Процесс использования:

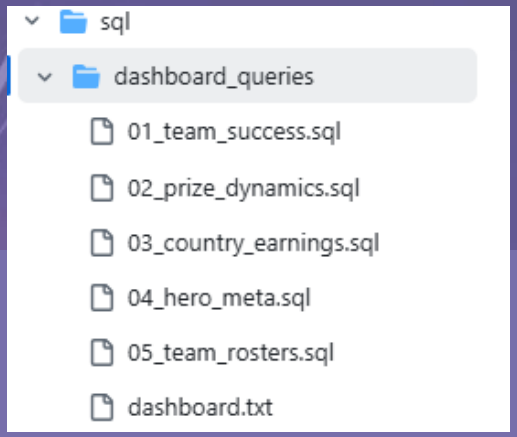
Выполнить запрос в PostgreSQL (схема dm / dds)

Экспортировать результат в CSV

Загрузить в DataLens и построить график

Добавить на дашборд





Анализ призовых по командам

mart_team_prize.sql - Анализ призовых по командам.

Гранулярность: год – команда

Метрики:

total_prize_usd - сумма выигранных призовых

Используется для:

динамики заработка команд по годам;

сравнения успешности команд.

```sql
SELECT
    dy.year,
    dt.team_name,
    SUM(f.prize_usd) AS total_prize_usd
FROM dm.f_winners_top3 f
JOIN dds.dim_team dt   ON dt.team_id = f.team_id
JOIN dds.dim_year dy   ON dy.year_id = f.year_id
GROUP BY dy.year, dt.team_name
ORDER BY total_prize_usd DESC;
```

year	team_name	total_prize_usd
2 021	Team Spirit	18 208 300
2 019	OG	15 620 181
2 018	OG	11 234 158
2 017	Team Liquid	10 862 683
2 016	Wings Gaming	9 139 002
2 015	Evil Geniuses	6 634 661
2 021	PSG.LGD	5 202 400
2 014	Newbee	5 025 029
2 019	Team Liquid	4 462 908
2 018	PSG.LGD	4 085 148
2 017	Newbee	3 950 067
2 021	Team Secret	3 601 600
2 016	Digital Chaos	3 427 126
2 019	PSG.LGD	3 089 706
2 015	CDEC Gaming	2 856 590
2 018	Evil Geniuses	2 680 879
2 017	LGD.Forever Young	2 592 231
2 015	LGD Gaming	2 211 554
2 016	Evil Geniuses	2 180 898
2 014	Vici Gaming	1 474 737
2 013	Alliance	1 437 190
2 014	Evil Geniuses	1 037 778
2 012	Invictus Gaming	1 000 000
2 011	Natus Vincere	1 000 000
2 013	Natus Vincere	632 364
2 013	Orange Esports	287 438
2 011	EHOME	250 000
2 012	Natus Vincere	250 000
2 011	Scythe Gaming	150 000
2 012	LGD Gaming	150 000

Оценка призовых игроков (с допущением равного деления командного выигрыша)

mart_player_income.sql - Оценка призовых игроков (с допущением равного деления командного выигрыша)

Допущение: приз команды делится поровну между 5 игроками основного состава.

Гранулярность: год – игрок

Метрики:

player_prize_usd — расчётный доход игрока

Позволяет анализировать:

самых успешных игроков;

распределение доходов по странам.

```sql
SELECT
    dp.player_name,
    dc.country_name,
    SUM(f.prize_usd / 5.0) AS estimated_income
FROM dm.f_winners_top3 f
JOIN dm.f_team_roster r    ON r.team_id = f.team_id AND r.year_id = f.year_id
JOIN dm.f_player_country pc ON pc.player_id = r.player_id AND pc.year_id = r.year_id
JOIN dds.dim_player dp ON dp.player_id = r.player_id
JOIN dds.dim_country dc ON dc.country_id = pc.country_id
GROUP BY dp.player_name, dc.country_name
ORDER BY estimated_income DESC;
```

player_name	country_name	estimated_income
JerAx	Finland	5 370 867,8
Topson	Finland	5 370 867,8
ana	Australia	5 370 867,8
N0tail	Denmark	5 370 867,8
TORONTOTOKYO	Russia	3 641 660
Collapse	Russia	3 641 660
Mira	Ukraine	3 641 660
Miposhka	Russia	3 641 660
Yatoro	Ukraine	3 641 660
KuroKy	Germany	3 191 591
GH	Lebanon	3 065 118,2
MinD_ContRoL	Bulgaria	3 065 118,2
Miracle-	Jordan	3 065 118,2
MATUMBAMAN	Finland	2 892 856,6
y`	China	2 868 280,4
Faith_bian	China	2 868 280,4
Ame	China	2 475 450,8
bLink	China	1 827 800,4
iceice	China	1 827 800,4
SumaiL	Pakistan	1 763 111,8
fy	China	1 729 918,2
w33	Romania	1 578 006,8
ppd	United States	1 534 487,8
xiao8	China	1 497 316,6
Chalice	China	1 434 970,8
xNova	Malaysia	1 434 970,8
Somnus丶M	China	1 434 970,8
zai	Sweden	1 364 055,2
UNiVeRsE	United States	1 326 932,2
Aui_2000	Canada	1 326 932,2
Fear	United States	1 326 932,2
Puppey	Estonia	1 096 792,8
SanSheng	China	1 055 005,8
NothingToSay	Malaysia	1 040 480
XinQ	China	1 040 480

Частота попадания команд в топ-3

mart_team_top3_frequency.sql - Частота попадания команд в топ-3.

Гранулярность: команда

Метрика:

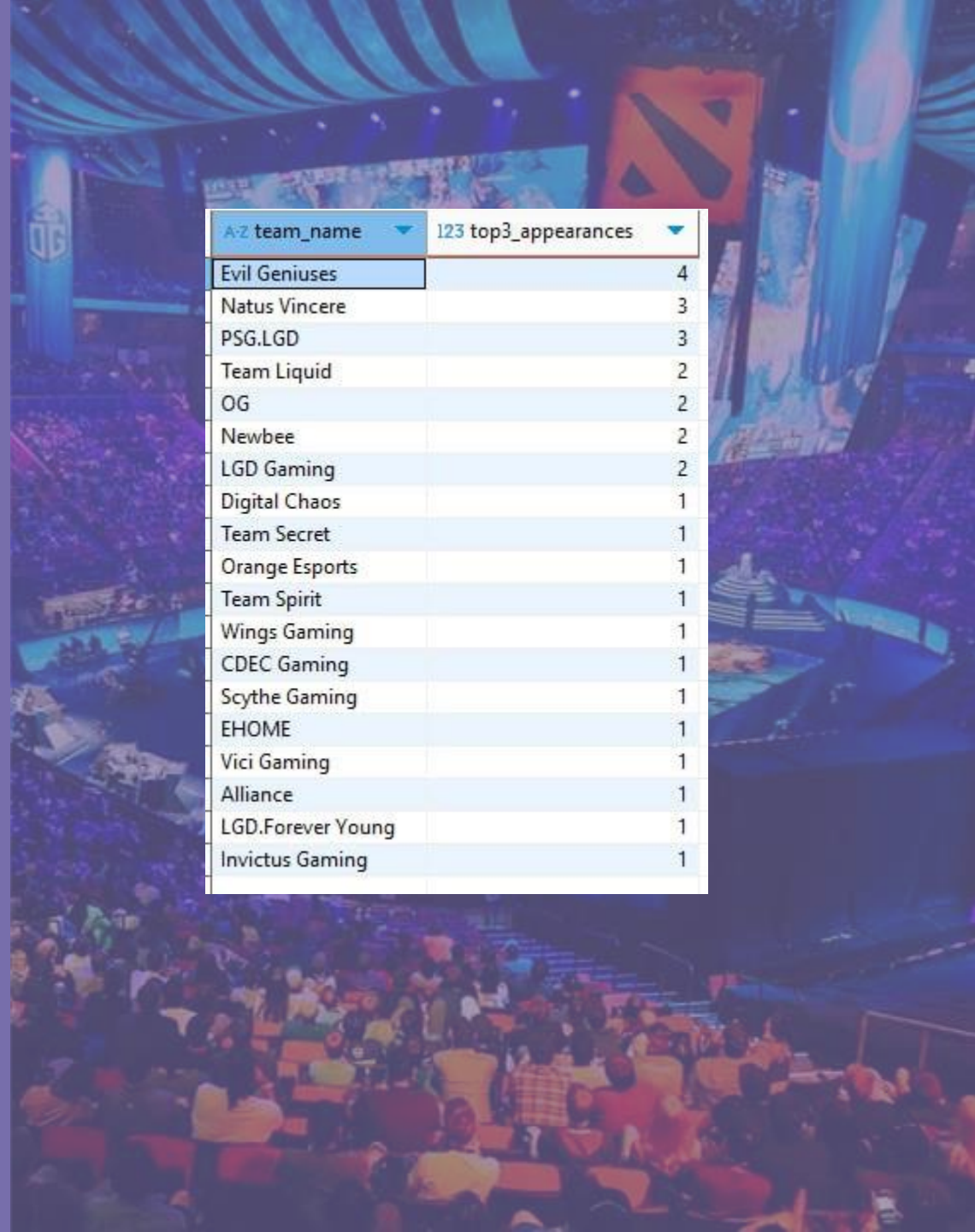
top3_count — количество попаданий в призовые места.

Используется для:

оценки стабильности команд;

построения рейтингов.

```sql
SELECT
    dt.team_name,
    COUNT(*) AS top3_appearances
FROM dm.f_winners_top3 f
JOIN dds.dim_team dt ON dt.team_id = f.team_id
GROUP BY dt.team_name
ORDER BY top3_appearances DESC;
```



team_name	top3_appearances
Evil Geniuses	4
Natus Vincere	3
PSG.LGD	3
Team Liquid	2
OG	2
Newbee	2
LGD Gaming	2
Digital Chaos	1
Team Secret	1
Orange Esports	1
Team Spirit	1
Wings Gaming	1
CDEC Gaming	1
Scythe Gaming	1
EHOME	1
Vici Gaming	1
Alliance	1
LGD.Forever Young	1
Invictus Gaming	1

Популярность героев по годам

mart_hero_popularity.sql - Популярность героев по годам.

Гранулярность: год - герой

Метрика:

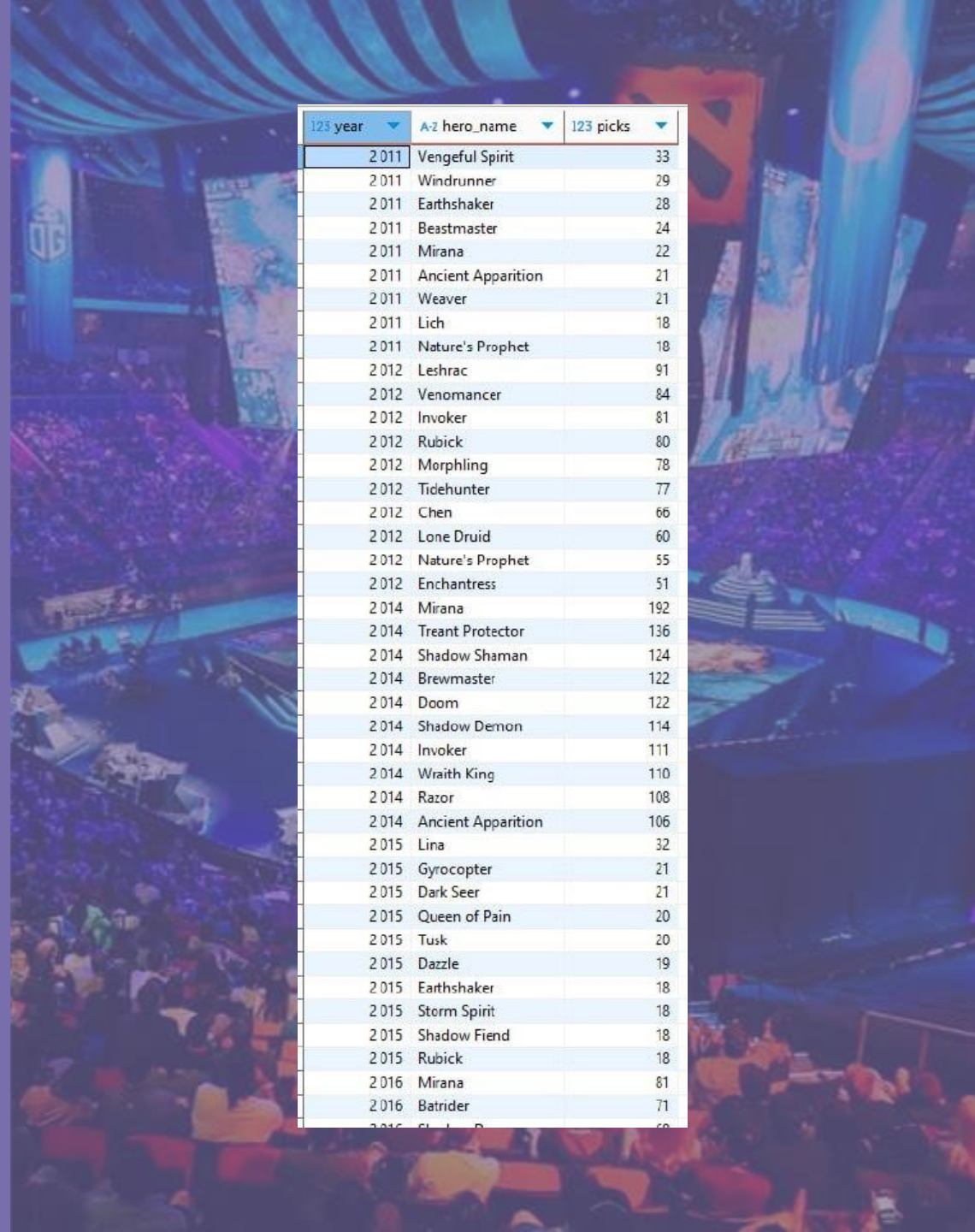
times_picked - количество выборов героя.

Позволяет анализировать:

мету игры;

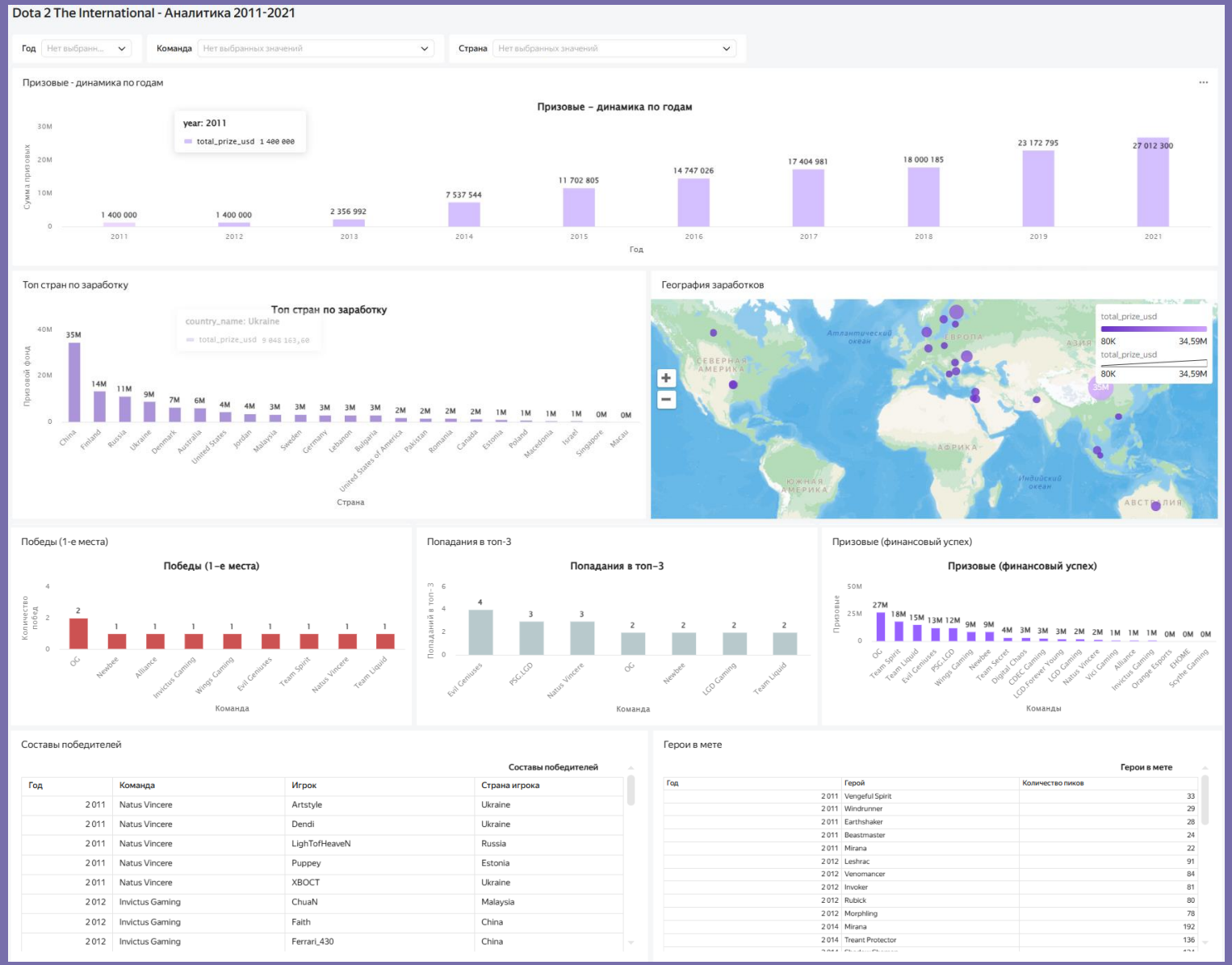
тренды популярности персонажей.

```sql
SELECT
    dy.year,
    dh.hero_name,
    SUM(f.times_picked) AS picks
FROM dm.f_hero_picks f
JOIN dds.dim_hero dh ON dh.hero_id = f.hero_id
JOIN dds.dim_year dy ON dy.year_id = f.year_id
GROUP BY dy.year, dh.hero_name
ORDER BY dy.year, picks DESC;
```



year	hero_name	picks
2011	Vengeful Spirit	33
2011	Windrunner	29
2011	Earthshaker	28
2011	Beastmaster	24
2011	Mirana	22
2011	Ancient Apparition	21
2011	Weaver	21
2011	Lich	18
2011	Nature's Prophet	18
2012	Leshrac	91
2012	Venomancer	84
2012	Invoker	81
2012	Rubick	80
2012	Morphling	78
2012	Tidehunter	77
2012	Chen	66
2012	Lone Druid	60
2012	Nature's Prophet	55
2012	Enchantress	51
2014	Mirana	192
2014	Treant Protector	136
2014	Shadow Shaman	124
2014	Brewmaster	122
2014	Doom	122
2014	Shadow Demon	114
2014	Invoker	111
2014	Wraith King	110
2014	Razor	108
2014	Ancient Apparition	106
2015	Lina	32
2015	Gyrocopter	21
2015	Dark Seer	21
2015	Queen of Pain	20
2015	Tusk	20
2015	Dazzle	19
2015	Earthshaker	18
2015	Storm Spirit	18
2015	Shadow Fiend	18
2015	Rubick	18
2016	Mirana	81
2016	Batrider	71

Результаты анализа, выводы по дашборду



Год	Команда	Игрок	Страна игрока
2011	Natus Vincere	Artstyle	Ukraine
2011	Natus Vincere	Dendi	Ukraine
2011	Natus Vincere	LighTofHeaveN	Russia
2011	Natus Vincere	Puppey	Estonia
2011	Natus Vincere	XBOCT	Ukraine
2012	Invictus Gaming	ChuaN	Malaysia
2012	Invictus Gaming	Faith	China
2012	Invictus Gaming	Ferrari_430	China

Год	Герой	Количество пиков
2011	Vengeful Spirit	33
2011	Windrunner	29
2011	Earthshaker	28
2011	Beastmaster	24
2011	Mirana	22
2012	Leshrac	91
2012	Venomancer	84
2012	Invoker	81
2012	Rubick	80
2012	Morphling	78
2014	Mirana	192
2014	Treant Protector	136

Команды: кто самый успешный?

OG: единственная команда с двумя победами (2018, 2019)

Team Spirit: финансовый рекордсмен (18 млн $ за 2021 год)

Natus Vincere и Team Secret: самые стабильные команды ранних лет (по 2 попадания в топ-3)

Успех можно измерять по-разному: OG по победам, Team Spirit по призовым

Динамика призовых:

Рост с 1 млн. $ (2011) до 18 млн $ (2021) более чем в 18 раз

Экспоненциальный скачок - с 2015 года призовые растут взрывными темпами

Индустрия киберспорта вышла на новый уровень

География успеха (страны):

Топ-3 страны по заработку: Китай, Финляндия, Россия

Восточная Европа и Китай главные представители кибер-талантов

Мета героев:

В 2021 году Tiny абсолютный король меты (192 пика)

Топ-герои меняются каждый год ни один герой не доминирует постоянно

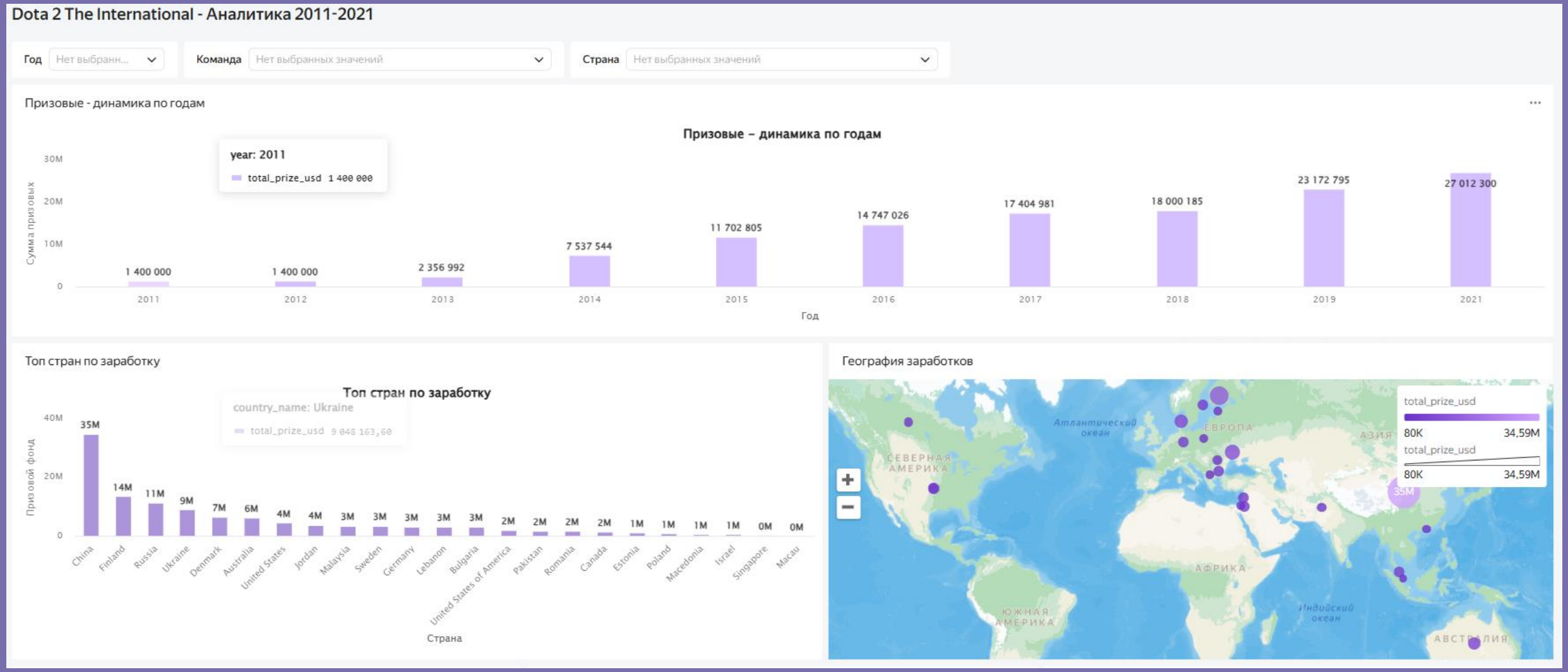
Мета Dota 2 очень динамична, патчи сильно влияют на выбор героев

Составы команд-победителей:

Победные составы редко сохраняются дольше одного-двух лет

Игроки мигрируют между командами, группируясь под конкретный турнир

DataLens Дашборд



DataLens Дашборд

Победы (1-е места)

Попадания в топ-3

Призовые (финансовый успех)

Составы победителей

Составы победителей

Год	Команда	Игрок	Страна игрока
2011	Natus Vincere	Artstyle	Ukraine
2011	Natus Vincere	Dendi	Ukraine
2011	Natus Vincere	LighTofHeaveN	Russia
2011	Natus Vincere	Puppey	Estonia
2011	Natus Vincere	XBOCT	Ukraine
2012	Invictus Gaming	ChuaN	Malaysia
2012	Invictus Gaming	Faith	China
2012	Invictus Gaming	Ferrari_430	China

Герои в мете

Герои в мете

Год	Герой	Количество пиков
2011	Vengeful Spirit	33
2011	Windrunner	29
2011	Earthshaker	28
2011	Beastmaster	24
2011	Mirana	22
2012	Leshrac	91
2012	Venomancer	84
2012	Invoker	81
2012	Rubick	80
2012	Morphling	78
2014	Mirana	192
2014	Treant Protector	136