# Описание проекта

В качестве оперативной БД я решил использовать не статичный датасет, а забирать данные через API из сераиса eBird.org: так получилось, что бердвотчинг – мое хобби и давно хотел реализовать свой первый пет-проект DWH с моделью для аналитики, которая будет полезна для меня как активного пользователя сервиса.

## Суть сервиса оперативной БД:

* пользователи через мобильное приложение загружают информацию о наблюдаемых видах птиц (speciescode, comname, sciname) с привязкой к локациям (идентификация по locid, locname, lat, lon)
* данные упаковывают в отдельные чеклисты (единица отправки информации от пользователей) с уникальным id (subId)
* для каждого вида птиц добавляются факты – кол-во наблюдаемых особей (howmany), флаги редкости и достоверности наблюдений (obsvalid).
* чеклисты добавляются по мере их подачи с атрибутом временной метки подачи (obsdt)

Мы будем использовать запрос «recent observation» только для географии Georgia (мой персональный ключ доступа для некоммерческого образовательного проекта жестко зашит в исходных файлах DAG).

Образец запроса свежих записей (с сайта документации API версии 2.0 для eBird <https://documenter.getpostman.com/view/664302/S1ENwy59>):

curl --location 'https://api.ebird.org/v2/data/obs/KZ/recent' \

--header 'X-eBirdApiToken: {{x-ebirdapitoken}}'

Пример JSON для ответа:

[

{

"speciesCode": "hoocro1",

"comName": "Hooded Crow",

"sciName": "Серая ворона",

"locId": "L7884500",

"locName": "улица Старикова, Chundzha KZ-Almaty (43.5309,79.4551)",

"obsDt": "2020-01-21 16:35",

"howMany": 1,

"lat": 43.530936,

"lng": 79.455132,

"obsValid": true,

"obsReviewed": false,

"locationPrivate": true,

"subId": "S63619695"

},

{ .. }, { … }, { … } …

]

## Целевая схема

В качестве целевой выбрана трехступенчатая схема ковейера данных с независимой БД на каждой ступени:

1. Из оперативной БД информация загружается в **Mirroring db** (**MRR**). В эту БД данные реплицируются в исходном виде без обработки и трансформации (только перевод в реляционный формат из JSON document-based db). По сути это полная копия исходных данных, обеспечивающая избыточность данных для высокой доступности и отказоустойчивости (защита от потери данных, рекавери в случае сбоев).

БД состоит из одной таблицы **mrr\_fact\_recent\_observation** в которую реплицируются все поступающие JSON документы:

A screenshot of a computer

Description automatically generated

1. Из MRR информация забирается в **staging db** (**STG**). В этой БД промежуточного хранения, исходные данные собираются из внешних и внутренних источников (у нас только один источник – внутренняя база MRR с накопленными отчетами о наблюдениях) и подготавливаются для дальнейшей трансформации в целевую модель данных DWH – очищаются, трансформируются и интегрируются, чтобы гарантировать целевые консистентность и качество.

Т.к. мы получаем из внешнего источника изначально хорошо подготовленные и очищенные данные, то на уровне данной БД мы просто уменьшаем набор данных, отбросив неинформативные для целевой модели поля и отбрасываем ненадежные записи о наблюдениях (помеченные модераторами eBird через флаг obsValid = false).

Эта БД также состоит из одной таблицы наблюдений **stg\_fact\_observation**, подготовленной для трансформации в целевую модель DWH.

A screenshot of a computer

Description automatically generated

1. Из данных, накопленных на уровне STG, мы формируем целевую базу данных нашего **Data Warehouse** (**DWH**). Эта БД поддерживает модель данных для эффективной обработки аналитических запросов и бизнес-аналитики (отчеты, интерактивные дашборды и т.д.). Мы строим OLAP систему с классической схемой «Звезда» (**Star Schema**).

Схема «звезда» подразумевает формирование центральной таблицы фактов (**fact**), которая содержит сведения о «событиях», обычно с измеряемой числовой «мерой», и нескольких таблиц измерений (**dimension**), содержащих атрибуты для классификации событий-фактов и свзанные с таблицей фактов по внешнему ключу.

В нашем случае, мы строим следующие 4 таблицы:

* Таблица фактов **dwh\_fact\_observation**, в которую фиксируются факты о наблюдении конкретного вида птиц (числовое поле – howmany). В качестве первичного ключа выступает связка полей (subid – id для чеклиста и speciescode – id вида птицы). По внешним ключам speciescode, locid и obsdt мы связываем ее с тремя таблицами измерений.
* Таблица измерений **dwh\_dim\_location** хранит информацию о местах наблюдений с первичным ключом locid и атрибутами locname (название локации) + lat/lon (географические координаты точки в WGS-84).
* Таблица измерений **dwh\_dim\_species** хранит информацию о видах птиц целевой локации (Georgia) с первичным ключом каждого вида – speciescode и атрибутами: comname – общепринятое национальное название (в запросе к сервису выбраны русские названия видов птиц), sciname – латинское название вида.
* Таблица измерений **dwh\_dim\_dt**, классическая таблица измерений для интегарции по временным интервалам – первичный ключ временной метки (obsdt) разбивается на составляющие, актуальные для агрегации – day, month/month\_name, year, quarter.

ERD диаграмма целевой «звезды»:

A screenshot of a computer

Description automatically generated

Кроме того, в БД DWH хранятся две служебные таблицы:

* **etl\_log**, системный лог в который записываются события (event\_type - INFO, WARNING, ERROR) процесса ETL и соответствующая временная метка (ts)
* **high\_water\_mark**, в которую записываются текущие крайние временные отметки для загрузки и обработки в каждух из трех БД только накопившихся новых строк (дельта из строк с более свежими timestamp чем сохраненные) в каждом прогоне DAG. Ключ table\_id – имя целевой таблицы, current\_high\_ts – текущее значение **high\_water\_mark.**

A screenshot of a computer

Description automatically generated

БД реализованы в PostgreSQL.

## Процесс ETL

ETL реализован в Airflow c конвейером из трех task:

A screenshot of a computer

Description automatically generated

1. **load\_mrr\_from\_ebird\_task**. Загружает свежие записи (используя временную метку high\_water\_mark) из оперативной БД через запрос API. Для обработки данных и их трансформации из JSON в реляционную таблицу в демонстрационных целях применен Spark, хотя практической целесообразности применения Spark для таких небольших объемов нет (запросом мы отфильтровываем только наблюдения для Грузии) и можно было реализовать всю обработку в pandas data frame.
2. **Load\_stg\_from\_mrr\_task** и **load\_dwh\_from\_stg\_task**. На этих тасках производятся последовательные переходы новых данных MRR -> STG и STG ->DWH, также с использованием временного маркера high\_water\_mark. Для подключения к postgressql из python используется PostgresHook из Airflow (основан на адаптере psycopg2).

Сейчас DAG настроен на расписание – 1 раз в час в целях разработки. Учитывая малую динамичность датасета, вполне достаточо выгружать дельту и обновлять high\_water\_mark один раз в сутки.

В целевой базе данных DWH реализованы две функции и две процедуры (с использованием механизмов курсоров, исключений и записи аварийных событий в лог-таблицу etl\_log:

* Функция **get\_bird\_species\_by\_loc** – для получения набора записей видов птиц, отмеченных за все время наблюдений в заданной локации
* Функция **get\_high\_water\_mark** – для получения текущего значения high\_water\_mark для заданной таблицы. Эта функция используется в ETL процессе.
* Процедура **process\_new\_row** – основная процедура трансформации данных из staging в целевые dim\_ и fact\_ таблицы схемы звезда, используется в ETL процессе. Т.к. процедуры поддерживают механизм транзакций (в отличие от функций), то все преобразование исходной строки из стейджинга в 4 таблицы целевой модели происходит в рамках единой тразакции; в случае исключения транзакция откатывается.
* Процедура **write\_log** пишет запись о событии в таблицу etl\_log.

A screenshot of a computer code

Description automatically generated

Также в БД сохранена вьюшка **etl\_log\_view** для лога событий.

## Уровень презентации в Power BI

Модель данных:

A screenshot of a computer

Description automatically generated

Интерактивный дашборд:

A map with red dots

Description automatically generated

Интерактивный дашборд позволяет работать с целевой OLAP моделью в трех измерениях, получая необходимые срезы и аналитики по наблюдениям птиц:

1. Выбор вида птиц с визуализацией по локациям наблюдений и частоте наблюдений
2. Выбор локации и анализ всех зафиксированных в ней видов птиц и частоты наблюдений в локации
3. Агрегация частоты наблюдений для различных временных интервалов – день, месяц, квартал, год, за все время – как для отдельных видов птиц, так и для выбранных локаций, всего объма данных в модели.

На уровне модели Power BI примен DAX синтаксис для построения нескольких вычисляемых полей:

A screenshot of a computer

Description automatically generated

Подключение к исходному инстансу postgresql реализовано через ODBC драйвер:



Пока модель загружена небольшим первичным объемом данных за июль (менее 10 чеклистов) и продолжает наполняться по мере поступления новых чеклистов.

## Скрипты в проекте

* **install\_dependecies.sh** – информационный скрипт с необходимыми модулями Python для работы DAG в Airflow
* **ebird-airflow\_setup.sh** – скрипт для настройки подключений к БД PostgreSQL из Airflow.

USAGE: ebird-airflow-setup.sh <host> <port> <username> <password> (указать данные для инстанса postgres)

* **ebird-full-backup.sh** – скрипт для создания full backup всех трех рабочих БД (MRR, STG, DWH). Целевой формат - .tar (для восстановления через pg\_restore), чтобы изменить на .sql необходимо подправить скрипт.

USAGE: ebird-full-backup.sh <host> <port> <username> <password> <dirname> (указать данные для инстанса postgres и путь к целевой папке для бэкапа)