



ETL пайплайны ChEMBL: Activity, Assay, TestItem, Target, Document

Пайплайн Activity (ChEMBL)

Источник данных: REST API ChEMBL (endpoint `/activity`) – возвращает записи биоактивности (биологические тесты молекул) ¹. Данные загружаются постранично через HTTP-запросы к API ChEMBL и представляются в формате JSON, который будет конвертирован в таблицу.

Ключевые поля: `activity_id` (уникальный ID активности), `molecule_chembl_id` (ChEMBL ID молекулы), `assay_chembl_id` (ChEMBL ID биоассая), а также связанные идентификаторы – например, `target_chembl_id` и `document_chembl_id` для ссылки на таргет и документ соответственно ² ³. Эти поля однозначно идентифицируют запись активности и ее связь с другими сущностями ChEMBL.

Описание стадий:

- **Extract:** извлекает полные данные активностей через ChEMBL API. Реализуется вызовом клиента ChEMBL или HTTP-запросов по батчам (с учетом pagination), сбором всех записей `/activity` и преобразованием ответа JSON в pandas DataFrame.
- **Transform:** выполняет очистку и нормализацию данных. Приводит типы колонок к заданным (например, числовые поля `value` / `standard_value` – к `float`, логические флаги – к `bool` и т.д.), убирает лишние пробелы в текстах, **удаляет пустые или отсутствующие значения** в критичных полях (например, записи без `activity_id` не проходят дальше). Также применяется специфичная обработка: например, парсинг вложенного объекта `ligand_efficiency` с метриками эффективности (LE, BEI, LLE, SEI) из JSON-формата в плоские поля или вычисление производных значений, если требуется. В этом минимальном пайплайне **внешнее обогащение не выполняется** – трансформация ограничена приведением данных к консистентному виду ¹.
- **Validate:** проверяет результат по схеме Pandera. Используется предопределенная схема валидации **ActivitySchema** (DataFrameSchema), где заданы все ожидаемые колонки и их типы. Схема отмечает обязательные поля (например, `activity_id` должен присутствовать и быть целым > 0) и проверяет форматы значений (например, корректность формата ChEMBL ID для связанных идентификаторов) ⁴. При валидации DataFrame приводится к строгой схеме (лишние колонки отбрасываются, недостающим могут присваиваться значения по умолчанию), и если все проверки проходят, возвращается тот же DataFrame; при несоответствии бросается ошибка.
- **Write:** стадия записи результатов делегируется базовому классу **PipelineBase** (через унифицированный writer). В рамках данного класса она не переопределяется, данные сохраняются в целевой формат (например, Parquet/CSV) с использованием общего механизма.

Скелет Python-класса:

```
class ChemblActivityPipeline(PipelineBase):  
    def extract(self) -> pd.DataFrame:  
        # Получение всех активностей через ChEMBL API (постстранично)  
        data = self.client.fetch_all(endpoint="/activity") # псевдо-код  
        обращения к API  
        df = pd.DataFrame(data) # конвертация списка записей в DataFrame  
        return df
```

```

def transform(self, df: pd.DataFrame) -> pd.DataFrame:
    # Нормализация типов согласно спецификации полей
    df = normalize_types(df, schema=self.config.fields)
    # Удаление записей без обязательных полей
    df = df.dropna(subset=["activity_id"])
    # Пример: парсинг объекта ligand_efficiency (если присутствует)
    if "ligand_efficiency" in df.columns:
        df = parse_ligand_metrics(df)
    return df

def validate(self, df: pd.DataFrame) -> pd.DataFrame:
    # Применение Pandera-схемы для валидации структуры и типов данных
    df = ActivitySchema.validate(df)
    return df

# Метод write() унаследован от PipelineBase (общая реализация записи
результатов)

```

Пайплайн Assay (ChEMBL)

Источник данных: REST API ChEMBL (endpoint `/assay`). Пайплайн выгружает информацию об опытах/биоассаях из ChEMBL, запрашивая ресурс `/assay` и постранично собирая все доступные записи (описания экспериментов, условий, привязки к мишеням и т.д.) ⁵. Данные приходят в JSON-формате и переводятся в pandas DataFrame.

Ключевые поля: `assay_chembl_id` (ChEMBL ID ассая, основной идентификатор эксперимента), `target_chembl_id` (ChEMBL ID связанного таргета, если ассай привязан к конкретной мишени), `document_chembl_id` (ChEMBL ID документа-источника данных по ассая) ⁶. Кроме того, в данных присутствуют характеристики ассая (тип, категория, ткань/клеточная линия и пр.), но перечисленные ID являются ключевыми для установления связей с другими сущностями.

Описание стадий:

- **Extract:** выполняет запросы к ChEMBL API по ресурсу `/assay` для получения всех записей биоассаяв. Полученные JSON-данные парсятся и агрегируются в DataFrame. Если задан фильтр или ограничение (например, извлечение только определенных Assay ID), дескриптор запроса настроит соответствующие параметры, иначе выгружается полный набор с поддержкой пагинации.

- **Transform:** приводит сырье данные ассая к стандартизированному виду. Производится нормализация типов: числовые поля (например, `confidence_score` – уровень достоверности соответствия таргету) конвертируются в `int`, логические флаги – в `bool`, строковые поля очищаются от лишних пробелов. Вложенные структуры, специфичные для ассая, могут быть разобраны: например, если в данных есть сложные вложенные поля (которые ChEMBL может отдавать для assay), они будут преобразованы в более плоский формат или отфильтрованы. Проводятся проверки целостности: записи без обязательных данных (например, без `assay_chembl_id` или `assay_type`) исключаются или помечаются. **Внешнее обогащение отсутствует** – трансформация ограничивается нормализацией полученной из ChEMBL информации.

- **Validate:** осуществляет проверку DataFrame схемой **AssaySchema** (Pandera). Схема гарантирует присутствие обязательных колонок (`assay_chembl_id` всегда должен быть, `assay_type` не NULL и пр.), соответствие типов (например, `confidence_score` – целое число 0–9) и формат ChEMBL ID для ссылок ⁷. Также могут выполняться дополнительные логические проверки –

например, если указаны таргет или документ, они должны соответствовать допустимому формату идентификатора. В случае успеха метод возвращает валидированный DataFrame; если найдены отклонения (неверные типы, отсутствующие ключевые поля), возбуждается ошибка валидации.

- **Write:** запись результатов (например, в файлы) выполняется унаследованным методом базового класса. Специфичная логика для ассая не требуется, используется общий **UnifiedOutputWriter** для сохранения, поэтому метод `write()` в данном классе не определяется явно.

Скелет Python-класса:

```
class ChembLAssayPipeline(PipelineBase):
    def extract(self) -> pd.DataFrame:
        # Извлечение всех ассаев через API ChEMBL
        data = self.client.fetch_all(endpoint="/assay")
        df = pd.DataFrame(data)
        return df

    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        # Приведение типов и очистка данных ассая
        df = normalize_types(df, schema=self.config.fields)
        df = df.dropna(subset=["assay_chembl_id", "assay_type"])
        # Дополнительная обработка специфичных вложенных полей (при необходимости)
        df = normalize_assay_data(df)
        return df

    def validate(self, df: pd.DataFrame) -> pd.DataFrame:
        # Проверка по Pandera-схеме AssaySchema
        df = AssaySchema.validate(df)
        return df

    # Метод write() наследуется от базового класса (универсальная запись результатов)
```

Пайплайн TestItem (ChEMBL)

Источник данных: REST API ChEMBL (endpoint `/molecule`). Этот пайплайн предназначен для выгрузки данных о **тестовых элементах** – химических соединениях (молекулах), которые испытываются в ассеях. Из ChEMBL загружаются сведения о молекулах: идентификаторы, структуры, свойства и иерархия (родительское соединение и др.)⁸. В случае необходимости пайплайн может обрабатывать определенный список ChEMBL ID (через параметры запуска) либо выгружать все молекулы постранично.

Ключевые поля: `molecule_chembl_id` (уникальный идентификатор молекулы в ChEMBL), `parent_molecule_chembl_id` (ChEMBL ID родительского соединения, если данная запись – производное, например соль или изомер)⁹¹⁰. Помимо этих ключей, в структуру данных молекулы входят **основные свойства** (название `pref_name`, максимальная фаза исследования `max_phase` и др.), **структурные представления** (канонический SMILES, InChI, молекулярная формула) и вложенные объекты: `molecule_properties` (логР, масса, формула и т.п.) и `molecule_structures` (SMILES, InChI, InChI Key). Эти вложенные поля содержат подробную информацию о соединении.

Описание стадий:

- **Extract:** обращается к ChEMBL API по ресурсу `/molecule` для получения данных о молекулах. Если предусмотрена выборка по списку ID (например, через опции CLI), на этапе извлечения выполняются батч-запросы по заданным `molecule_chembl_id`. В общем случае выгружаются все молекулы постранично, ответ JSON (содержит поля молекулы и вложенные объекты) собирается и преобразуется в DataFrame.
- **Transform:** проводит подготовку и нормализацию сведений о молекулах. Все указанные в конфигурации поля приводятся к требуемым типам: числовые свойства (`alogp`, `full_mwt` и пр. внутри `molecule_properties`) конвертируются в `float`, строки (например, `pref_name`) очищаются от пробелов, бинарные флаги (`max_phase` может трактоваться как целое) приводятся к числовому или категориальному виду. Вложенные объекты могут быть преобразованы: например, для удобства анализа можно извлечь ключевые свойства из `molecule_properties` (массу, формулу) в отдельные колонки DataFrame либо сохранить вложенные поля как структуры (словарь/JSON) в ячейках. Пустые или нулевые значения свойств могут заменяться на `Nan/None` для единобразия. **Внешнее обогащение (через PubChem)** в минимальной реализации **не выполняется** – трансформация ограничена данными ChEMBL (например, не подтягиваются дополнительные свойства или синонимы извне) [11](#) [12](#).
- **Validate:** использует схему Pandera `TestItemSchema` для проверки корректности данных молекул. Схема гарантирует наличие ключевого поля `molecule_chembl_id` и правильность его формата (`CHEMBLXXXX`), проверяет типы базовых атрибутов (например, `max_phase` — `int`, `molecule_type` — категория/строка) и допустимость значений (например, `max_phase` в диапазоне 0–4). Также проверяется целостность и структура сложных полей: вложенные объекты и списки либо валидируются через кастомные проверки, либо игнорируются схемой, но ключевые их части (если вынесены в отдельные колонки) должны соответствовать ожидаемым типам. При успешной валидации возвращается исходный DataFrame; если данные не соответствуют схеме (например, отсутствует `molecule_chembl_id` или неверный тип `max_phase`), выбрасывается исключение.
- **Write:** этап записи реализован в базовом классе и не меняется для `TestItem`. После валидации DataFrame передается унифицированному механизму записи, который сохраняет данные молекул в указанный формат/хранилище.

Скелет Python-класса:

```
class TestItemChembelPipeline(PipelineBase):  
    def extract(self) -> pd.DataFrame:  
        # Извлечение данных молекул из ChEMBL API  
        data = self.client.fetch_all(endpoint="/molecule",  
ids=self.config.get("ids"))  
        df = pd.DataFrame(data)  
        return df  
  
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:  
        # Нормализация полей молекулы: типы, строки, вложенные объекты  
        df = normalize_types(df, schema=self.config.fields)  
        # Обработка вложенных структур, например разворачивание свойств  
        if "molecule_properties" in df.columns:  
            df["full_mwt"] = df["molecule_properties"].apply(lambda prop:  
prop.get("full_mwt"))  
            # Удаление пустых идентификаторов  
            df = df.dropna(subset=["molecule_chembl_id"])
```

```

    return df

    def validate(self, df: pd.DataFrame) -> pd.DataFrame:
        # Валидация по схеме TestItemSchema (Pandera)
        df = TestItemSchema.validate(df)
        return df

    # Метод write() используется из базового класса (общий для всех
    пайплайнов)

```

Пайpline Target (ChEMBL)

Источник данных: REST API ChEMBL (endpoint `/target`). Пайpline загружает данные о таргетах (биологических мишениях: белки, комплексы, клеточные линии и т.п.) из ChEMBL. Для каждой записи таргета API возвращает общую информацию: тип мишени (например, SINGLE PROTEIN, PROTEIN FAMILY), название, организм, таксономию и список компонентов (например, если таргет — комплекс, перечисляются входящие белки) ¹³. Запросы выполняются к ресурсу `/target` (с поддержкой постраничной выборки), результат собирается и преобразуется в DataFrame.

Ключевые поля: `target_chembl_id` (уникальный ChEMBL ID таргета), `pref_name` (стандартное имя/описание мишени), `organism` (организм, от которого этот таргет), `target_type` (категория таргета: отдельный белок, семья, клеточная линия и т.д.), а также сложноструктурные поля: `target_components` (список компонентов таргета, например UniProt ID белков) и `cross_references` (список внешних идентификаторов в других базах) ¹³ ¹⁴. Основной ключ — `target_chembl_id`, остальные поля дают контекст и связи, включая таксономический ID (`tax_id`) для организма.

Описание стадий:

- **Extract:** выполняет извлечение всех записей таргетов через ChEMBL API `/target`. Полученные JSON-данные содержат информацию о таргете и вложенные списки компонентов `ixref` (кросс-референсов). Эти ответы преобразуются в DataFrame; вложенные списки на этапе извлечения могут временно храниться как объекты (списки словарей) внутри ячеек DataFrame. Если требуется извлечь конкретные таргеты, механизм дескриптора может ограничить выборку по списку ID или применить фильтры.

- **Transform:** обрабатывает структуру таргет-данных для приведения к необходимому виду. Базовая нормализация включает приведение типов: `tax_id` в `int` (если не NULL), булевое поле `species_group_flag` — к `bool`. Текстовые поля (`pref_name`, `organism`) очищаются от лишних пробелов. Вложенные поля `target_components` и `cross_references` могут остаться как есть (списками внутри колонок) либо сериализоваться в стандартизованный формат (например, JSON-строку) — в зависимости от требований схемы. В минимальной реализации глубокое развертывание этих списков не производится, они хранятся как списки объектов. Внешнее обогащение (например, подтягивание данных из UniProt или IUPHAR) **не выполняется** — трансформация ограничивается данными ChEMBL. Также проверяется целостность: если `target_type` отсутствует или некорректно указан, такая запись может быть отброшена, так как этот атрибут обязательный ¹⁵.

- **Validate:** финальная валидация данных через схему **TargetSchema** (Pandera). Схема требует наличие поля `target_chembl_id` (если его нет — данные невалидны) ¹⁶, проверяет типы основных колонок (`target_type` должен быть строкой из известного набора, `tax_id` — целое или NULL и т.д.). Так как `target_components` и `cross_references` — списки, базовая Pandera-схема может объявить их типом "object" или использовать специальные проверки для элементов списка; в минимальной версии проверяется только наличие этих колонок и тип как Python list.

При валидации убеждаемся, что каждый таргет имеет тип (не пустой `target_type`) и идентификатор. Если все условия выполнены, возвращается DataFrame; иначе возникает ошибка.

- **Write:** запись результатов таргет-данных выполняется базовым механизмом PipelineBase. Пайплайн не переопределяет метод записи – DataFrame после валидации передается unified writer для сохранения (например, формирование выходного файла с фиксированным порядком колонок).

Скелет Python-класса:

```
class ChEMBLTargetPipeline(PipelineBase):
    def extract(self) -> pd.DataFrame:
        # Извлечение данных таргетов из ChEMBL API
        data = self.client.fetch_all(endpoint="/target")
        df = pd.DataFrame(data)
        return df

    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        # Нормализация типов основных полей
        df = normalize_types(df, schema=self.config.fields)
        # Оставляем списки компонентов ixref как есть (или конвертируем в
        JSON при необходимости)
        # Удаляем записи без ключевых данных
        df = df.dropna(subset=["target_chembl_id", "target_type"])
        return df

    def validate(self, df: pd.DataFrame) -> pd.DataFrame:
        # Валидация по Pandera-схеме TargetSchema
        df = TargetSchema.validate(df)
        # Дополнительно убеждаемся, что ключевой идентификатор присутствует
        assert not df["target_chembl_id"].isnull().any()
        return df

    # Метод write() используется от базового класса (стандартная логика
    # сохранения)
```

Пайплайн Document (ChEMBL)

Источник данных: REST API ChEMBL (endpoint `/document`). Данный пайплайн выгружает метаданные научных публикаций, патентов и других источников, связанных с биологической активностью. Через `/document` API ChEMBL предоставляет для каждого документа такие поля, как заголовок, список авторов, название журнала, год издания, DOI, PubMed ID, патентный номер (для патентов) и пр. ¹⁷ ¹⁸. Все записи документов из ChEMBL (или ограниченный набор по фильтрам) извлекаются постранично и собираются в DataFrame.

Ключевые поля: `document_chembl_id` (уникальный идентификатор документа в ChEMBL), а также внешние идентификаторы при их наличии: `doi` (Digital Object Identifier публикации), `pubmed_id` (ID статьи в PubMed), `patent_id` (номер патента, если тип документа – патент) ¹⁹ ²⁰. Эти поля позволяют однозначно ссылаться на источник данных и использовать внешние библиографические сервисы. Дополнительно каждый документ содержит обязательный заголовок `title` и классифицируется полем `doc_type` (тип источника: статья, патент, тезисы и

т.д.), которое всегда заполнено.

Описание стадий:

- **Extract:** загружает все записи документов из ChEMBL API (`/document`). Результаты содержат библиографические сведения (JSON-объекты для каждого документа). На этапе извлечения формируется DataFrame, где каждая строка – один документ. Поля с множественными значениями (например, список авторов) могут прийти как строка, разделенная точкой с запятой, либо как список; в ChEMBL обычно авторы представлены одной строкой, что сразу помещается в DataFrame.
- **Transform:** нормализует и очищает данные документа. Приводятся типы: числовые поля (`year` – в `int`, при пустом значении становится `NaN`; `chembl_release_id` – в `int`), флаги/коды – в соответствующий тип (при необходимости). Текстовые поля (`title`, `journal`, список `authors`) очищаются от незначимых символов (например, лишних запятых, точек), строки приводятся к единому стилю (обрезаются пробелы). Если `authors` предоставлены как список, они конвертируются в строку (например, объединяются через `;`) для согласованности с ожидаемой схемой. Поле `doc_type` проверяется и при необходимости нормализуется до фиксированного набора значений (например, `'PUBLICATION'`, `'PATENT'` и т.д.). **Внешнее обогащение** метаданными (через API PubMed, CrossRef и др.) **не выполняется** – трансформация ограничена данными ChEMBL. Таким образом, поля DOI или PubMed ID используются как есть, без дополняния информации о статьях.
- **Validate:** выполняется проверка с помощью схемы `DocumentSchema` (Pandera). Схема требует наличие ключевых полей: обязательны `document_chembl_id` и `title` (не пустые), `doc_type` – соответствует одному из предопределенных типов источников. Проверяются типы (`year` – целое или null, `authors` – строка, `doi` и `pubmed_id` – строки либо null, и т.д.). Для идентификаторов DOI может быть задан шаблон (regex), для PubMed ID – числовой формат. Схема также гарантирует строгий порядок колонок и отсутствие лишних столбцов. Кроме того, дополнительно убедимся, что если `doc_type` = "PATENT", поле `patent_id` не пустое (бизнес-правило). При успешной валидации возвращается DataFrame, несоответствие формату или отсутствие обязательных данных вызывает ошибку (например, если `document_chembl_id` отсутствует или `title` пустой, валидация провалится) ²¹.
- **Write:** сохранение результатов выполняется базовой реализацией PipelineBase. Пайплайн Document не переопределяет метод записи – валидированные данные документов будут автоматически сохранены с помощью общего writer (после успешной проверки схемы).

Скелет Python-класса:

```
class ChembldocumentPipeline(PipelineBase):  
    def extract(self) -> pd.DataFrame:  
        # Получение всех документов через ChEMBL API  
        data = self.client.fetch_all(endpoint="/document")  
        df = pd.DataFrame(data)  
        return df  
  
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:  
        # Нормализация полей документа: типы, строки  
        df = normalize_types(df, schema=self.config.fields)  
        # Объединение списка авторов в строку (если требуется)  
        if "authors" in df.columns and isinstance(df.at[0, "authors"], list):  
            df["authors"] = df["authors"].apply(lambda auth: "; ".join(auth))  
        # Удаляем записи без обязательных полей  
        df = df.dropna(subset=["document_chembl_id", "title", "doc_type"])
```

```

    return df

def validate(self, df: pd.DataFrame) -> pd.DataFrame:
    # Проверка по схеме DocumentSchema (Pandera)
    df = DocumentSchema.validate(df)
    # Дополнительная проверка: doc_type и связанные поля
    assert df[df["doc_type"] == "PATENT"]["patent_id"].notna().all()
    return df

# Метод write() унаследован (общая логика сохранения результатов)

```

Источник: Конфигурации и документация проекта BioETL подтвердили указанные поля, источники данных и этапы обработки для каждого пайплайна ¹ ². Все реализованные классы наследуются от `PipelineBase`, обеспечивающего базовый шаблон ETL (`extract → transform → validate → write`) ²². Enrichment (обогащение внешними данными) умышленно исключено во всех представленных пайплайнах, фокусируясь лишь на данных ChEMBL. Каждая реализация возвращает pandas DataFrame на этапах **extract, transform, validate**, используя Pandera-схемы для гарантий целостности перед записью результатов.

¹ INDEX.md

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/chembl/activity/INDEX.md>

² ³ activity.yaml

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/configs/pipelines/chembl/activity.yaml>

⁴ 00-activity-chembl-overview.md

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/chembl/activity/00-activity-chembl-overview.md>

⁵ ⁷ 00-assay-chembl-overview.md

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/chembl/assay/00-assay-chembl-overview.md>

⁶ assay.yaml

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/configs/pipelines/chembl/assay.yaml>

⁸ ¹¹ ¹² 00-testitem-chembl-overview.md

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/chembl/testitem/00-testitem-chembl-overview.md>

⁹ ¹⁰ molecule.yaml

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/configs/pipelines/chembl/molecule.yaml>

¹³ ¹⁴ ¹⁵ target.yaml

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/configs/pipelines/chembl/target.yaml>

¹⁶ 00-target-chembl-overview.md

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/chembl/target/00-target-chembl-overview.md>

17 18 19 20 **document.yaml**

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/configs/pipelines/chembl/document.yaml>

21 **00-document-chembl-overview.md**

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/chembl/document/00-document-chembl-overview.md>

22 **04-architecture-and-duplication-reduction.md**

<https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/project/04-architecture-and-duplication-reduction.md>