Документация

Выпуск 6

XDAC - Data Access Connector

февр. 27, 2026

Содержание

ГЛАВА 1

Общие сведения

1.1 Знакомство с XDAC

XSQUARE-Data Access Connector (**XDAC**) - это автономный высокопроизводительный веб-сервер, который предоставляет API на основе источников данных - баз данных различного типа.

XDAC автоматически превращает схему вашей базы данных в полноценный RESTful API. Сервис поддерживает множество типов баз данных, предоставляя единый стандарт доступа к данным независимо от того, где они хранятся.

Использование XDAC позволяет сосредоточиться на данных, а не на реализации бизнес-логики, уменьшая при этом количество ошибок и дублирующего кода, что облегчает его поддержку. Таким образом, XDAC ускоряет разработку программного обеспечения, избавляя разработчика от рутинного процесса написания однотипного кода.

XDAC поддерживает следующие базы данных:

- PostgreSQL
- Microsoft SQL Server
- Oracle
- Firebird
- Ydb (Yandex DB)

1.2 Как работает XDAC?

1.2.1 Типы операций

XDAC поддерживает два основных типа операций:

grpc — вызов процедур и функций
: При подключении к базам данных XDAC получает набор процедур и функций. Полученный список записывается в кэш приложения, который далее используется для сопоставления запрашиваемых процедур и параметров, переданных в запросе.

Каждая процедура и функция из списка публикуется как endpoint приложения XDAC. Доступ осуществляется по наименованию процедуры или функции, которое указывается в пути вызова сервиса.

dml — CRUD с таблицами и представлениями

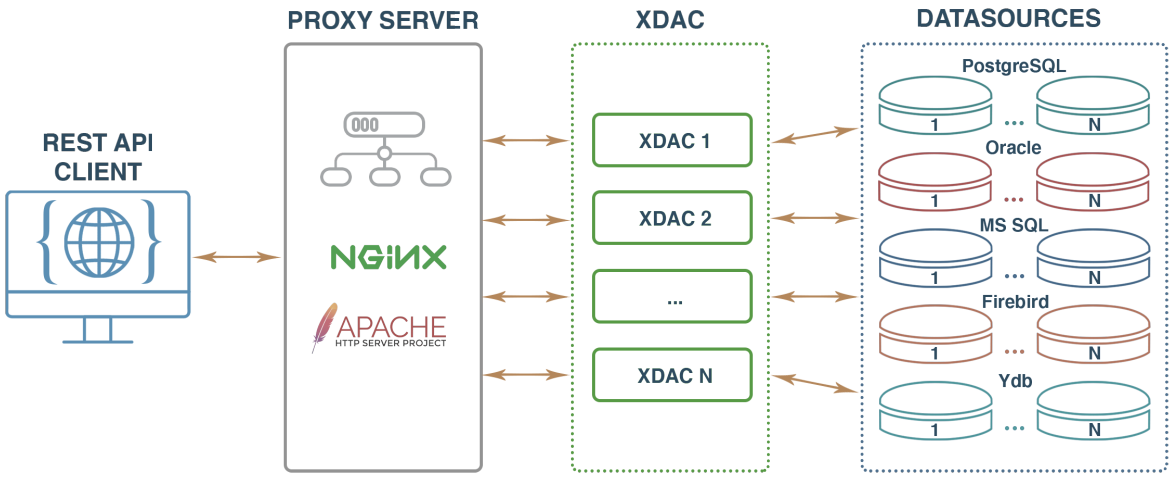
XDAC позволяет проводить операции чтения, записи, обновления и удаления данных в таблицах и представлениях с помощью HTTP-запросов соответствующего типа.

ГЛАВА 2

Архитектура и системные требования

2.1 Архитектура

Архитектура приложения представляет собой веб-сервер, который обрабатывает клиентские HTTP-запросы посредством предоставления доступа к процедурам, функциям, таблицам и представлениям из источников данных (баз данных) и возвращает результат клиенту.



2.2 Среда исполнения

Поддерживаемые архитектуры:

- x86-64

- ARM
- E2K
- Loongson

Поддерживаемые ОС:

- **DEB**-based – любые
- **RPM**-based - любые
- Debian 12 - рекомендуемая

Поддерживаемые СУБД:

- PostgreSQL
- Oracle
- Microsoft SQL Server
- Firebird
- Ydb

2.3 Системные требования

- CPU - 1 Ядро
- RAM - 100 Мб
- HDD - 100 Мб + Логи

Установка системы виртуализации/контейнеризации, операционной системы, базы данных осуществляется на усмотрение Администратора исходя из потребностей Организации.

ГЛАВА 3

Установка

Рассмотрим установку XDAC на примере ОС Debian.

Все команды следует выполнять с правами суперпользователя (root).

Создаем каталог для дистрибутива XDAC

```
mkdir /root/xsquare
```

переходим в каталог

```
cd /root/xsquare
```

Скачиваем/получаем дистрибутив в созданный каталог

```
wget https://lcdp.xsquare.ru/files/xdac/xsquare.xdac.6.0_latest_release.deb
```

Устанавливаем XDAC

```
dpkg -i xsquare.xdac.6.0_latest_release.deb
```

Проверяем состояние службы

```
systemctl status xsquare.xdac
```

ГЛАВА 4

Конфигурационные файлы

4.1 Файл конфигурации config.json

Для работы сервера отчетов необходимо, чтобы в директории с сервисом присутствовал файл конфигурации **config.json**.

Файл конфигурации содержит 2 раздела:

1. **app –** параметры работы приложения.

   Раздел **app** поддерживает следующие параметры:

   - **port** - строка. Определяет номер сетевого порта, на котором будет запущен сервис (по умолчанию - 8887)
   - **debug** - логическое значение. Включает режим отладки, при котором доступен подробный лог обработки запросов и расширенное описание ошибок.

2. **datasources –** массив источников данных, каждый из которых имеет типовой набор параметров для соединения и работы с БД.

   Каждый элемент datasources - описатель БД поддерживает следующие параметры:

   - **name** - строка. Имя источника данных. Данное имя будет использовано при регистрации эндпоинтов для выполнения запросов, поэтому данное поле не должно содержать символов недопустимых в url.
   - **dbType - строка. Тип базы данных источника данных. Поддерживаемые значения:**
     - postgresql
     - oracle
     - mssql
     - firebird
     - ydb
   - **host** - строка. Доменное имя или IP-адрес сервера БД.

- **port** - число. Сетевой порт сервера БД.
- **login** - строка. Имя пользователя для подключения к БД.
- **password** - строка. Пароль пользователя для подключения к БД.
- **dbName** - строка. Имя подключаемой базы на сервере БД.
- **parsingSchema** – строка. Определяет:
  - имя схемы в базе данных PostgreSQL и MS SQL Server, в которой будет производиться сканирование процедур и функций.
  - имя пакета в базе данных Oracle и Firebird, в котором будет производиться сканирование процедур и функций.
  - для Ydb не используется.
- **reloadCacheInterval** – число. Определяет временной интервал в секундах для обновления кэша процедур и функций. Значение по умолчанию: 0. При нулевом значении механизм обновления кэша не запускается.
- **minCons** - число. Минимальное количество сессий пула соединений
- **maxCons** - число. Максимальное количество сессий пула соединений
- **runtimeOptions** - набор параметров среды выполнения. Например «LC_NUMERIC»: «ru_RU.UTF-8» позволяет установить локализацию числовых форматов для БД postgresql.

В конфигурационном файле вы можете определить множество источников данных как одного так и разных типов БД.

Пример конфигурационного файла config.json

```
{
  "app": {
    "port": "8887",
    "debug": true
  },
  "datasources": [
    {
      "name": "pg_db",
      "dbType": "postgresql",
      "host": "localhost",
      "port": 5432,
      "login": "XDAC_USER",
      "password": "XDAC_PASSWORD1!",
      "dbName": "xdac_db",
      "parsingSchema": "api",
      "reloadCacheInterval": 300,
      "minCons": 1,
      "maxCons": 2,
      "runtimeOptions": {
        "LC_NUMERIC": "ru_RU.UTF-8"
      }
    },
    {
      "name": "firebird_db",
      "dbType": "firebird",
      "host": "localhost",
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
      "port": 3050,
      "login": "xdac_user",
      "password": "xdac_password1!",
      "dbName": "xdac_db",
      "parsingSchema": "PKG_API",
      "minCons": 1,
      "maxCons": 2
    },
    {
      "name": "ora_db",
      "dbType": "oracle",
      "host": "localhost",
      "port": 1521,
      "login": "XDAC_USER",
      "password": "xdac_password1!",
      "dbName": "xdac_db",
      "parsingSchema": "PKG_API",
      "minCons": 1,
      "maxCons": 2
    },
    {
      "name": "mssql_db",
      "dbType": "mssql",
      "host": "localhost",
      "port": 1433,
      "login": "XDAC_USER",
      "password": "XDAC_PASSWORD1!",
      "dbName": "xdac_db",
      "parsingSchema": "api",
      "minCons": 1,
      "maxCons": 2
    },
    {
      "name": "ydb_db",
      "dbType": "ydb",
      "host": "localhost",
      "port": 2135,
      "login": "xdac_user",
      "password": "xdac_password1!",
      "dbName": "Root/xdac_db",
      "parsingSchema": "api",
      "minCons": 1,
      "maxCons": 2
    }
  ]
}
```

Примечание

По умолчанию для подключения к БД Ydb используется безопасный вариант GRPC - grpcs. При необходимости использования grpc, нужно указать в runtimeOptions флаг «unsecure»: true.

Примечание

В режиме отладки (флаг «debug»: true) XDAC выводит в stdout список доступных процедур и функций, а также заполняет поле description ошибки внутренними сообщениями от БД.

ГЛАВА 5

XDAC - Data Access Connector

5.1 Эксплуатация

XDAC устанавливается в качестве службы, для запуска которой необходимо выполнить команду:

```
systemctl start xsquare.xdac
```

> **Примечание**
>
> При установке из deb или rpm пакета служба запускается автоматически по окончанию установки.

Для запуска XDAC необходимо убедиться в наличии файла лицензии xdac.lic и правильно настроенного конфигурационного файла config.json.

В случае изменения параметров конфигурационного файла необходимо перезапустить службу:

```
systemctl restart xsquare.xdac
```

Проверить состояние службы можно с помощью команды:

```
systemctl status xsquare.xdac
```

Ошибки в работе сервера приложения будут записаны в журнал, отобразить сообщения можно с помощью команды:

```
journalctl -u xsquare.xdac
```

Также можно проверять состояние сервера удаленно, с помощью GET-запроса к эндпоинту **status**:

```
curl http://localhost:8887/status
```

В ответ XDAC пришлет json следующего формата:

```
{
  "status":"UP",
  "version":"6.0.0.2",
  "build":"24.02.2026_00:09:39",
  "license": "информация о лицензии"
}
```

5.2 Настройка СУБД. Права пользователя.

Установка операционной системы, базы данных, реверс-прокси сервера, локали осуществляется на усмотрение Администратора исходя из потребностей Организации. Пример расширенной настройки можно найти в документации «Администрирование XSQUARE-PGHS», ГЛАВА «Расширенная установка»

Для сканирования каждой базы данных необходим пользователь с возможностью создания сессии и правами EXECUTE для соответствующего пакета/функции. Права доступа к зависимым объектам определяются на усмотрение Администратора. Для вывода ошибок связанных с правами необходимо установить в конфигурационном файле флаг «debug»:true для отображения расширенного лога ошибок.

Примечание

Примеры создания пользователя с необходимыми правами находятся в каталоге «example» дистрибутива XDAC.

ГЛАВА 6

Процедуры и функции - rpc

XDAC позволяет вызывать процедуры и функции из источников данных следующих типов БД:

- postgresql
- oracle
- mssql
- firebird

> **Примечание**
>
> Для ydb данный функционал не поддерживается в виду того, что ydb не использует процедуры и функции.

6.1 Сканирование процедур и функций

При старте XDAC устанавливает соединение с базой данных в соответствии с параметрами, указанными в конфигурационном файле config.json.

После успешного соединения сервер приложений сканирует процедуры и функции в указанной **схеме** или **пакете** базы данных и формирует **кэш процедур и функций**.

В кэш записываются только процедуры и функции удовлетворяющие следующим условиям:

- Процедуры с входными параметрами или без входных параметров, но без возвращаемого значения.
- Функции с входными параметрами или без входных параметров, с одним возвращаемым значением.

6.2 Вызов процедур и функций

Каждая процедура и функция, полученная в результате сканирования схемы (пакета), публикуется сервисом с префиксом **/xdac/[datasource_name]/rpc/**, где datasource_name - имя источника данных в config.json

```
http://localhost:8887/xdac/pg_db/rpc/function_name
```

Примечание

Так как регистрация эндпоинтов по именам источников данных происходит при старте XDAC, то при запросе url с некорректным именем источника данных будет возвращен код ошибки - 404.

В общем случае, для вызова процедуры или функции могут быть использованы GET и POST - запросы. При использовании GET-запроса, передаваемые параметры указываются в URL-адресе с использованием строки запроса:

```
curl "http://localhost:8887/xdac/pg_db/rpc/function_name?first_param=value&second_
↪param=value"
```

При использовании POST-запроса, передаваемые параметры необходимо включить в тело запроса json, наименование полей которого будет соответствовать наименованию аргументов, либо передавать как данные формы (multipart/form-data):

- Параметры в теле json:

```
curl -d '{ "first_param": value, "second_param": value  }' http://localhost:8887/xdac/
↪pg_db/rpc/function_name
```

- Параметры как данные формы:

```
curl --form first_param=value --form second_param=value localhost:8887/xdac/pg_db/rpc/
↪function_name
```

Для примера создадим для базы данных PostgreSQL функцию для сложения двух чисел следующего вида:

```
CREATE OR REPLACE FUNCTION api.add_numbers (
    first integer,
    second integer
)
RETURNS integer AS
$body$
BEGIN
    return first + second;
END;
$body$
LANGUAGE 'plpgsql'
```

Пример вызова функции add_numbers с помощью GET-запроса:

```
curl "http://localhost:8887/xdac/pg_db/rpc/add_numbers?first=20&second=30"
```

Ответ: 50

Пример вызова функции add_numbers с помощью POST-запроса с параметрами в теле json:

```
curl -d '{ "second": 5, "first": 6  }' http://localhost:8887/xdac/pg_db/rpc/add_
↪numbers
```

Ответ: 11

Пример вызова функции add_numbers с помощью POST-запроса с параметрами multipart/form-data:

```
curl --form first=12 --form second=34 http://localhost:8887/xdac/pg_db/rpc/add_numbers
```

Ответ: 46

В случае некорректного имени параметра, отсутствия параметра или передачи лишнего параметра в ответ будет получено сообщение об ошибке в формате json. Например:

```
curl -d '{ "second1": 5, "first": 6  }' http://localhost:8887/xdac/pg_db/rpc/add_
↪numbers
```

Ответ:

```
{
  "code": "XDAC005",
  "hint": "Make sure that routine name is correct or try to reload application",
  "message": "Routine add_numbers(first,second1) doesn't exists in app cache",
  "details": ""
}
```

Если у параметров процедуры или функции есть значения по умолчанию, то данные параметры будут считаться необязательными и они могут отсутствовать в запросе. Например, если наша функция add_numbers имеет следующий формат:

```
create function add_numbers(first integer DEFAULT 1, second integer DEFAULT 2)
```

в этом случае функцию можно вызвать без параметров:

```
curl http://localhost:8887/xdac/pg_db/rpc/add_numbers
```

Ответ: 3

Примечание

Каждый вызов процедуры или функции осуществляется в своей транзакции. Наличие оператора `COMMIT` в теле функции приведет к ошибке.

6.3 Правила наименования

Для баз данных PostgeSQL, MS SQL Server все наименования (имена процедур и функций, наименования аргументов) - **регистрозависимые**.

Для вызова функции с именем StRaNgeNamE необходимо вызвать метод следующим образом:

```
curl "http://localhost:8887/xdac/pg_db/rpc/StRaNgeNamE"
```

Вызов в другом формате приведет к ошибке, что такая функция не найдена:

```
{
  "code": "XDAC001",
  "hint": "Make sure that Routine Name is correct or try to reload application",
  "message": "Routine strangename was not found in app cache",
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
  "details": ""
}
```

Для баз данных Oracle и Firebird **все наименования будут автоматически приведены к верхнему регистру**.

6.4 Передача JSON в качестве параметра

Если в качестве параметра функции или процедуры требуется передать json, то в запросе необходимо передать заголовки *Content-Type: application-json* и *Prefer:params=single-object*:

```
curl -H "Content-Type: application/json" -H "Prefer: params=single-object" -d '{"text
↪":"Test message","input":[1,2,3]}' "http://localhost:8887/xdac/pg_db/rpc/jsonInput"
```

Сама процедура или функция должна иметь только один аргумент с типом данных подходящим для обработки json. Например для PostgreSQL:

```
CREATE OR REPLACE FUNCTION api."jsonInput" (
    "myPar" json
)
RETURNS json AS
$body$
begin
return "myPar";
END;
$body$
LANGUAGE 'plpgsql'
```

6.5 Передача бинарного файла в качестве параметра

Если в качестве параметра функции или процедуры требуется бинарный файл, то в запросе необходимо передать заголовок **Content-Type: application/octet-stream**:

```
curl "http://localhost:8887/xdac/pg_db/rpc/upload_binary" \
    -X POST -H "Content-Type: application/octet-stream" \
    --data-binary "@file_name.ext"
```

При таких запросах вызываемая процедура или функция должна иметь один обязательный параметр для обработки бинарных данных.

Например для PostgreSQL:

```
CREATE OR REPLACE PROCEDURE api."upload_binary" (
    bytea
)
AS
$body$
BEGIN
    insert into api.files(blob)
    values ($1);
END;
$body$
LANGUAGE 'plpgsql'
```

Также вы можете передать бинарный файл с помощью multipart/form-data запроса. XDAC произведет поиск процедуры или функции в соответствии с наименованием передаваемых параметров, за исключением параметров, которые содержат файл. Все файлы которые передаются в запросе будут размещены в «служебную» **xdac_files**.

Данная таблица содержит следующие поля:

- **name** - имя параметра в запросе
- **file_name** - имя файла
- **file_mime** - MIME-тип файла
- **file_content** - содержимое файла

В зависимости от типа базы данных таблица xdac_files будет создана разными способами:

- PostgreSQL – временная таблица pgtemp.xdac_files

```
CREATE TEMP TABLE IF NOT EXISTS pg_temp.xdac_files
```

- Oracle – глобальная временная таблица XDAC_FILES

```
CREATE GLOBAL TEMPORARY TABLE XDAC_FILES
```

- MS SQL Server – временная таблица xdac_files

```
CREATE TABLE #xdac_files
```

- Firebird – глобальная временная таблица XDAC_FILES

```
CREATE GLOBAL TEMPORARY TABLE XDAC_FILES
```

Примечание

После завершения текущего запроса (транзакции) информация о загруженных файлах будет удалена.

6.5.1 Пример работы с файлами для базы данных PostgreSQL

Создадим функцию save_document в схеме api:

```
CREATE OR REPLACE FUNCTION api.save_document (
  document varchar,
  document_date varchar
)
RETURNS jsonb LANGUAGE 'plpgsql'
AS
$body$
declare
  l_rec record;
BEGIN
  select f.file_name, f.file_mime, length(f.file_content) file_length
  into l_rec
  from pg_temp.xdac_files f
  where f.name = 'document_scan';
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
  return jsonb_build_object(
          'document',"document",
          'document_date',"document_date",
          'file_name',l_rec.file_name,
          'file_mime',l_rec.file_mime,
          'file_length',l_rec.file_length
    );
END;
$body$;
```

Выполним запрос к серверу приложений, который передает файл my_image.svg:

```
curl --form document=passport --form document_date=10.01.2002 --form document_scan=
↪'@my_image.svg' localhost:8887/xdac/pg_db/rpc/save_document
```

Ответ:

```
{
  "document": "passport",
  "document_date": "10.01.2002",
  "file_length": 9710,
  "file_mime": "image/svg+xml",
  "file_name": "my_image.svg"
}
```

6.6 Работа с заголовками

Все заголовки передаваемые в сервис записываются в параметры сессии.

Следующие параметры доступны для доступа:

- **request.headers –** заголовки запроса
- **request.method –** метод запроса
- **request.path –** путь по которому был вызван сервис

Для того чтобы установить заголовки ответа и код ответа используются параметры **response.headers** и **response.status**.

Заголовок **Content-Type** XDAC проставляет самостоятельно. Любое значение переданное в этот заголовок будет перезаписано.

Хранение данных параметров реализовано в зависимости от базы данных:

- **PostgreSQL – параметры записываются в Grand Unified Configuration (GUC), таким образом работа с параметрами осуществляется посредством вызовов set_config и current_settings.**
  Например:

```
-- Получить все заголовки. Они преобразуются в json
SELECT current_setting('request.headers', true)::json;

-- Метод с помощью которого был вызван сервис
SELECT current_setting('request.method', true); --GET/POST
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
-- Путь по которому был вызван сервис
SELECT current_setting('request.path', true); --/xdac/pg_db/rpc/test

— Чтобы установить заголовки ответа, необходимо передать массив объектов json
SELECT set_config('response.headers','[{"Content-Type":"application/json"},{
↪"Set-Cookie":"foo=bar"}]',true);

-- устанавливает код ответа в 405
SELECT set_config('response.status','405',true);
```

- **Oracle** – параметры записываются в пакет **XDAC_VARIABLES**, для записи используется процедура **SET_CONFIG**, для чтения используется функция **GET_CONFIG**, которые работают с таблицей **CONFIG_TABLE**:

```
CREATE OR REPLACE PACKAGE XDAC_VARIABLES AS
TYPE config_record IS RECORD (
    key_name CLOB,
    value_name CLOB
);
TYPE config_table IS TABLE OF config_record INDEX BY PLS_INTEGER;
g_config config_table;
PROCEDURE SET_CONFIG(p_key CLOB, p_value CLOB);
FUNCTION GET_CONFIG(p_key CLOB) RETURN CLOB;
END XDAC_VARIABLES
```

- **MS SQL Server** – параметры записываются в контекст сессии через **sp_set_session_context**, получить значения можно посредством **SELECT SESSION_CONTEXT**
- **Firebird** – параметры записываются в контекст **USER_TRANSACTION** пользовательской сессии с помощью **rdb$set_context**, для чтения используется **rdb$get_context**.

  Например:

```
SELECT rdb$set_context('USER_TRANSACTION', 'request.headers', '') FROM rdb
↪$database
SELECT rdb$get_context('USER_TRANSACTION', 'request.headers'') FROM rdb
↪$database
```

6.7 Обработка ошибок

В случае возникновения ошибки сервис вернёт её в формате:

```
{
  "code": "",
  "hint": "",
  "message": "",
  "details": ""
}
```

Для базы данных PostgreSQL сервис предоставляет возможность обрабатывать пользовательские ошибки вызванные оператором `raise`

```
raise 'This is postgresql error'
      USING ERRCODE = 'SRV01',
      HINT = 'Try GET method',
      DETAIL = 'This is detail';
```

XDAC преобразует ошибку в следующий вид:

```
{
  "code": "SRV01",
  "hint": "Try GET method",
  "message": "This is postgresql error",
  "details": "This is detail"
}
```

При этом транзакция будет завершена, а все изменения отменены с помощью оператора `ROLLBACK`. Если необходимо передать пользовательскую ошибку в сервис, при этом сохранить результаты выполнения функции, можно использовать установку статуса ответа через GUC.

```
perform set_config('response.status','405',true);
perform set_config('response.headers','[{"Content-Type":"application/json"}]',true);
return json_build_object('message','Only GET method is allowed');
```

6.8 Обновление кэша процедур и функций

Все процедуры и функции в схеме или пакете сканируются при запуске приложения и помещаются в кэш. При изменении сигнатуры процедур или создании новых функций они не будут доступны для вызова через API. Для того, чтобы стали доступны обновленные процедуры или функции необходимо обновить кэш. Этого можно достигнуть следующим образом:

1. Перезагрузить приложение
2. Задать в config.json параметр reloadCacheInterval в секундах, отвечающий за интервал времени для обновления кэша
3. Обратиться к endpoint RELOAD для принудительного обновления кэша:

   ```
   curl http://localhost:8887/reload
   ```

Также для базы данных PostgreSQL доступен механизм обновления кэша с помощью уведомлений `NOTIFY`. Для этого необходимо отправить в канал xdac уведомление о перезагрузке кэша:

```
NOTIFY xdac, 'reload';
```

Чтобы автоматизировать отправку уведомлений в канал можно создать **event trigger на операции ddl**. Пользователь с правами SUPERUSER может создать следующий триггер:

```
CREATE OR REPLACE FUNCTION public.xdac_drop_watch () RETURNS event_trigger
LANGUAGE plpgsql
AS
$body$
DECLARE
  obj record;
BEGIN
 FOR obj IN SELECT * FROM pg_event_trigger_dropped_objects()
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
  LOOP
   IF obj.object_type in ('function','procedure') AND not obj.is_temporary then
     NOTIFY xdac, 'reload';
   end if;
 END LOOP;
END;
$body$;



CREATE OR REPLACE FUNCTION public.xdac_ddl_watch () RETURNS event_trigger
LANGUAGE plpgsql
$body$
DECLARE
  cmd record;
BEGIN
 FOR cmd IN SELECT * FROM pg_event_trigger_ddl_commands()
 LOOP
    IF cmd.command_tag IN ('CREATE FUNCTION', 'ALTER FUNCTION','CREATE PROCEDURE',
↪'ALTER PROCEDURE')
      and cmd.schema_name != 'pg_temp'
   then
     NOTIFY xdac, 'reload';
   end if;
 END LOOP;
END;
$body$;


CREATE EVENT TRIGGER xdac_ddl_watch  ON ddl_command_end  EXECUTE PROCEDURE public.
↪xdac_ddl_watch();
CREATE EVENT TRIGGER xdac_drop_watch ON sql_drop  EXECUTE PROCEDURE public.xdac_drop_
↪watch();
```

ГЛАВА 7

Таблицы и представления - dml

XDAC позволяет проводить операции чтения, записи, обновления и удаления данных в таблицах и представлениях с помощью http-запросов соответствующего типа.

Запросы dml имеют следующий формат:

http://localhost:8887/xdac/datasource_name/dml/schema.table_or_view?operators

Например:

```
curl "http://localhost:8887/xdac/pg_db/dml/api.t_clients?age=in.(35,39)"
```

Примечание

для указания схемы или пакета, содержащего таблицу или представление, необходимо использовать классическую форму записи через символ точки: схема.имя_таблицы

7.1 SELECT

Для чтения данных из таблицы или представления (SELECT FROM) необходимо выполнить **GET-запрос** к эндпоинту источника данных с указанием имени таблицы или представления:

```
curl "http://localhost:8887/xdac/pg_db/dml/t_clients"
```

Сервис возвращает данные в формате NDJSON (Newline Delimited JSON) с использованием HTTP Chunked Transfer Encoding.

Каждая строка ответа содержит один валидный JSON-объект, завершающийся символом новой строки (n).

Данные отправляются по мере их чтения из базы данных.

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/x-ndjson
Transfer-Encoding: chunked

{"age":78,"birthdate":"1986-02-27T00:00:00Z","description":"B","first_name":"АЛЕКСЕЙ",
↪"id":2767,"last_name":"ЕМЕЛЬЯНОВ","salary":41622.90,"second_name":"ЕВГЕНЬЕВИЧ"}
{"age":64,"birthdate":"1965-11-03T00:00:00Z","description":"D","first_name":"ЮЛИЯ","id
↪":2769,"last_name":"ЛОГУНОВА","salary":86158.23,"second_name":"МИХАЙЛОВНА"}
{"age":42,"birthdate":"1997-11-07T00:00:00Z","description":"y","first_name":"АНДРЕЙ",
↪"id":2777,"last_name":"БЕТТ","salary":90502.28,"second_name":"АНАТОЛЬЕВИЧ"}
```

Для выбора конкретных колонок необходимо использовать оператор select с перечислением имен необходимых колонок, например:

```
curl "http://localhost:8887/xdac/pg_db/dml/t_clients?select=age,first_name"
```

Пример ответа:

```
HTTP/1.1 200 OK
Content-Type: application/x-ndjson
Transfer-Encoding: chunked

{"age":78,"first_name":"АЛЕКСЕЙ"}
{"age":64,"first_name":"ЮЛИЯ"}
{"age":42,"first_name":"АНДРЕЙ"}
```

XDAC позволяет фильтровать результирующие строки, добавляя условия по столбцам с помощью специальных операторов.

Например, чтобы вернуть данные о людях младше 18 лет:

```
curl "http://localhost:8887/xdac/pg_db/dml/t_clients?age=lt.18"
```

Также можно задавать несколько условий по столбцам, добавляя дополнительные параметры в строку запроса через символ **&**.

Например, чтобы вернуть людей, которым больше 18 лет и у которых зарплата больше 50000:

```
curl "http://localhost:8887/xdac/pg_db/dml/t_clients?age=gt.18&salary=gt.50000"
```

7.2 DML - операторы

Оператор `eq`

Значение: равно

Аналог PostgreSQL: =

Пример: `id=eq.2770`

Оператор `gt`

Значение: больше

Аналог PostgreSQL: >

Пример: `date_of_birth=gt.1989-01-01`

Оператор `ge`

Значение: больше или равно

Аналог PostgreSQL: >=

Пример: `age=ge.35`

Оператор `lt`

Значение: меньше

Аналог PostgreSQL: <

Пример: `age=lt.35`

Оператор `le`

Значение: меньше или равно

Аналог PostgreSQL: <=

Пример: `age=le.35`

Оператор `neq`

Значение: не равно

Аналог PostgreSQL: <> или `!=`

Пример: `age=neq.35`

Оператор `like`

Значение: оператор LIKE (для избежания URL-кодирования в шаблоне нужно использовать * как псевдоним символа %)

Аналог PostgreSQL: `LIKE`

Пример: `name=like.Us*`

Оператор `ilike`

Значение: оператор ILIKE (регистронезависимый аналог LIKE; * также заменяет %)

Аналог PostgreSQL: `ILIKE`

Пример: `name=like.us*`

Оператор `in`

Значение: значение из списка, например: ?a=in.(1,2,3). Поддерживает запятые в кавычках: ?a=in.(«привет,мир»,»да,вы»)

Аналог PostgreSQL: `IN`

Пример: `age=in.(33,36,37,38)`

Оператор `is`

Значение: проверка на соответствие состоянию: null, not_null, true, false, unknown

Аналог PostgreSQL: `IS`

Пример: `is_active=is.true salary=is.null`

Оператор `not`

Значение: отрицание другого оператора

Аналог PostgreSQL: `NOT`

Пример: `age=not.eq.40`

Оператор `or`

Значение: логическое ИЛИ

Аналог PostgreSQL: `OR`

Пример: `or=(age.lt.18,age.gt.21)`

Оператор `and`

Значение: логическое И

Аналог PostgreSQL: `AND`

Пример: `and=(balance.gt.500,balance.lt.1500)`

7.3 INSERT

Для записи данных в таблицу необходимо выполнить **POST-запрос** к эндпоинту источника данных с указанием имени таблицы и массивом добавляемых данных в JSON.

Например:

```
curl "http://localhost:8887/xdac/pg_db/dml/t_clients" -X POST -H "Content-Type:␣
↪application/json" \
  -d @- << EOF
  [
    {"age":77,"birthdate":"1986-02-27T00:00:00Z","description":"insert by xdac",
↪"first_name":"Имя", last_name":"Фамилия","salary":41622.90,"second_name":"Отчество"}
↪,
    {"age":7,"birthdate":"2026-02-27T00:00:00Z","description":"insert by xdac","first_
↪name":"Имя2", last_name":"Фамилия2","salary":4.90,"second_name":"Отчество2"},
  ]
EOF
```

При успешном выполнении запроса сервис вернет код 201, а иначе - ошибку.

7.4 UPDATE

Для изменения данных в таблице (UPDATE) необходимо выполнить **PATCH или PUT-запрос** к эндпоинту источника данных с указанием имени таблицы, массивом обновляемых данных в JSON и набором операторов.

Например обновление first_name и last_name для строк с id == 1:

```
curl -X PATCH "http://localhost:8887/xdac/pg_db/dml/t_clients?id=eq.1" -H "Content-
↪Type: application/json" \
  -d '{
    "first_name": "John",
    "last_name": "Doe"
  }'
```

При успешном выполнении запроса сервис вернет код 200, а иначе - ошибку.

7.5 DELETE

Для удаления данных в таблице или представлении (DELETE) необходимо выполнить **DELETE-запрос** к эндпоинту источника данных с указанием имени таблицы и набором операторов.

Например удаление строк с id > 2770:

```
curl "http://localhost:8887/xdac/pg_db/dml/t_clients?id=gt.2770" -X DELETE
```

При успешном выполнении запроса сервис вернет код 200, а иначе - ошибку.

Примечание

По умолчанию dml-операции возваращают только код возврата или ошибку, для получения количество затронутых изменениями строк (affected rows) необходимо в запросе указать заголовок: **«Prefer: return=representation»**.