Опыт применения инструмента файлового импорта при миграции на АБС ЦФТ-Банк и СУБД FTData

Для стабильного роста бизнеса и выполнения постоянно меняющихся требований регуляторов банкам необходимо использовать современные технологические платформы. С учетом еще одного фактора — необходимости обеспечения импортонезависимости — чаще всего этого можно добиться только в процессе обновления используемой ИТ-архитектуры.

При этом психологическими стоп-факторами для принятия соответствующего решения являются неприемлемый период недоступности автоматизированной банковской системы и высокие риски возникновения ошибок в процессе миграции.

А значит, к комплексу инструментов миграции и целевой платформе возникает ряд критических требований:

1

Скорость

За ограниченный период времени (обычно процесс перехода проводят в выходные или праздничные дни) необходимо загрузить очень большой объём данных.

2

Гибкость

Часто возникает потребность вносить дополнительные изменения в базу данных даже после тестирования импорта. Необходимо реагировать на новые изменения и загружать обновления в условии ограниченного времени.

3

Анализ ошибок

Инструмент миграции должен обеспечивать быстрый и наглядный анализ наличия ошибок импорта и последующей обработки.

4

Удобство использования и доработки

Инструмент миграции должен работать в тесной интеграции и с учетом особенностей целевых АБС и СУБД, желательно его функционирование в пределах заложенной под целевую АБС инфраструктуры.

Всем этим критериям соответствует инструмент файлового импорта для миграции учетной системы банка на АБС ЦФТ-Банк и СУБД FTData, позволяя сделать процесс переноса данных быстрым, простым, контролируемым и прозрачным.

Архитектурное решение инструмента импорта АБС ЦФТ-Банк

Ранее в АБС ЦФТ-Банк использовался инструмент файлового импорта, совместимый и оптимизированный под применение СУБД Oracle. В целях импортозамещения было принято решение адаптировать существующие опыт и наработки для применения их совместно с СУБД семейства PostgreSQL. В описываемом кейсе в качестве СУБД используется FTData — отечественный форк PostgreSQL, имеющий ряд существенных преимуществ при применении совместно с АБС ЦФТ-Банк.

Миграция с применением инструмента файлового импорта происходит в несколько этапов:

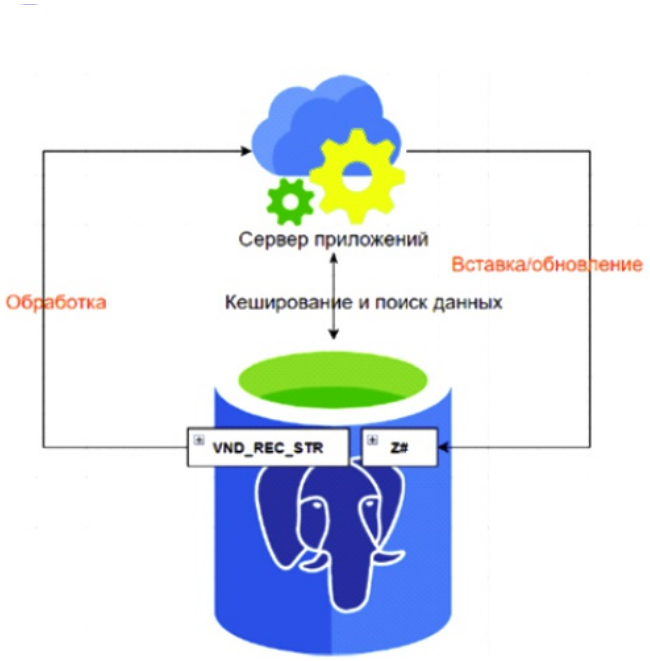
Этап 1
Загрузка данных в базу данных

Исходный файл загружается со всем содержимым в буферную таблицу в VND_REC_STR. Все данные, которые есть в файле, полностью сохраняются. Этот файл фактически живёт в базе данных.

Этап 2 — Обработка данных

Содержимое файла из буферной таблицы раскладывается сразу же в рабочие таблицы АБС ЦФТ-Банк. Такая структура позволяет увидеть, что изначально пришло в файле, увидеть исходные данные и перенести всю обработку в сервера приложений и базу данных.

Загрузка исходного файла во временную таблицу



Обработка данных из буферной таблицы в z#

Применение инструмента импорта

Основные данные инструмента импорта находятся в справочниках:

1. VND Универсальный импорт. Реестр загрузки файлов.
2. VND Универсальный импорт. Форматы файлов.

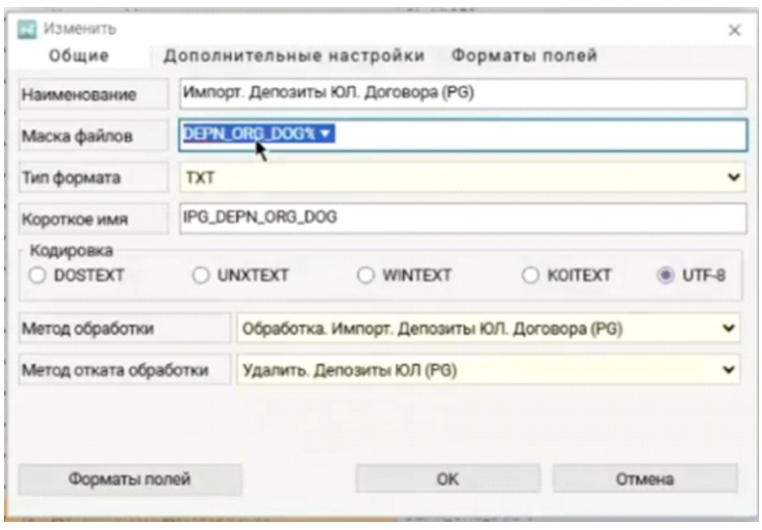
Форматы

Каждому файлу загрузки соответствует формат, который описан, и по которому делается выгрузка.

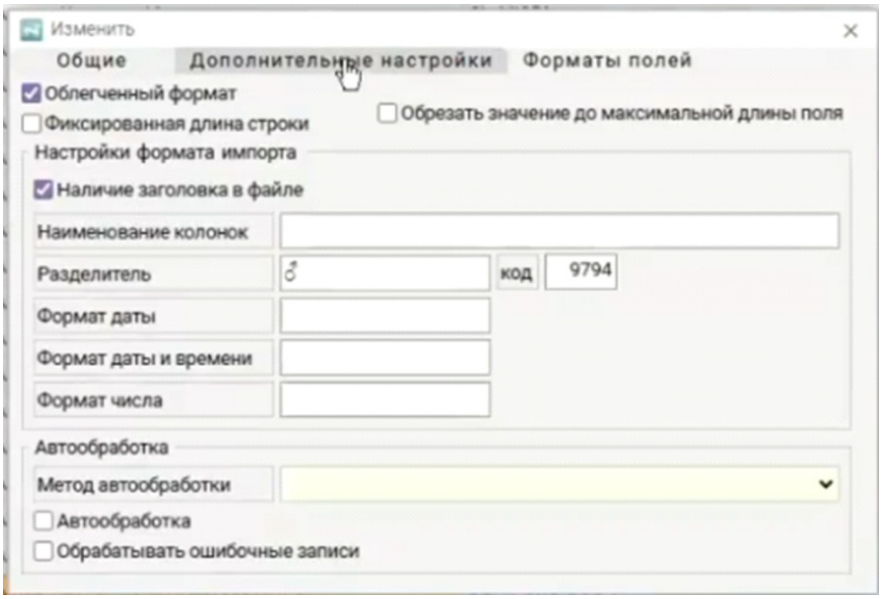
Пример описания файла с депозитами юридических лиц:

Новая АБС				Система-источник				Комментарии и особенности миграции
Поле	Наименование поля	Формат поля		Поле	Наименование поля	Формат поля		
		Длина	Тип данных			Длина	Тип данных	
1	2	3	4	5	6	7	8	9
ICODE*	Внутренний код импорта (ВКИ)	25	Строка					Уникальный код в разрезе продуктов
DATE_BEGINING*	Изначальная дата начала договора (до пролонгаций)	10	Дата					Не может быть больше DATE_BEGIN
DATE_BEGIN	Дата начала действия договора	10	Дата					Если не указана, то берется DATE_BEGINING
DATE_ENDING*	Дата окончания договора	10	Дата					Не заполняется для договоров до востребования
DATE_CLOSE*	Дата закрытия договора	10	Дата					
COM_STATUS	Статус договора	16	Строка Код из справочника «Обобщенные статусы» По договорам, по которым не было зачислений, проставляется статус «Открыт», по тем, по которым есть дата закрытия статус «Закрыт», остальные «работает»					
VID_DOG*	Вид депозита	30	Код из справочника "Виды депозитных договоров"					
DEPART*	Подразделение	30	Код из справочника "Подразделения"					
CLIENT*	Вкладчик	25	Строка					ВКИ клиента, БИК или код SWIFT банка
NUM_DOG*	Номер договора	22	Строка					
FINTOOL*	Валюта	3	Строка					Код ISO или короткое наименование валюты

Каждому такому файлу соответствует **запись с форматами.**
Она включает:

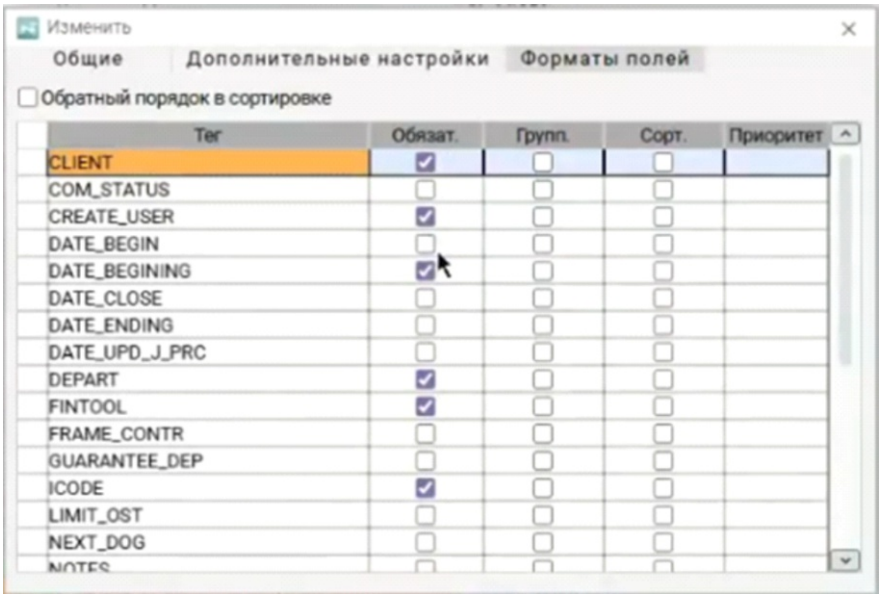


Дополнительные настройки:



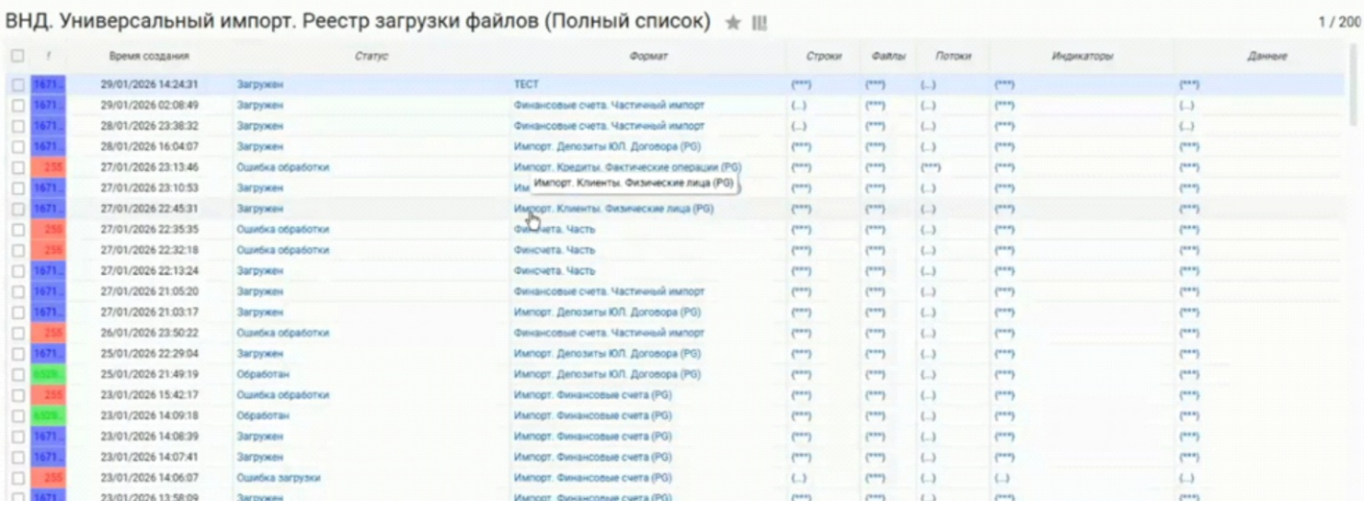
Формат полей

Таблица с форматом полей состоит из тегов. В формате проставлены отметки обязательности полей; заполненность полей будет контролироваться на этапе загрузки. При необходимости есть возможность проставить группировки сортировки и приоритет при сортировке.

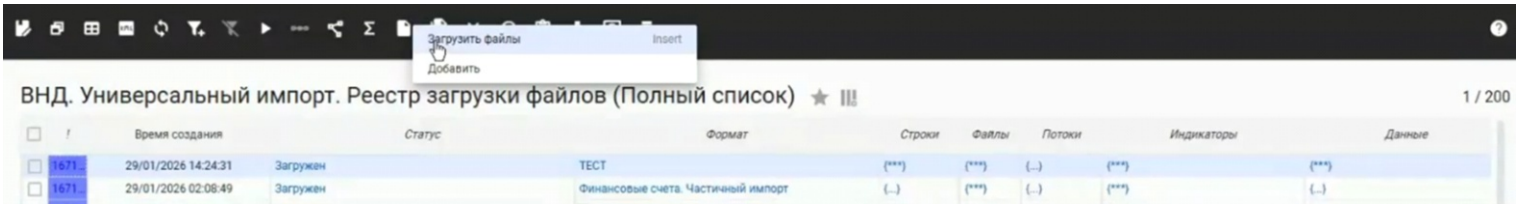


Реестр загрузки файлов

Реестр загрузки файлов — это буферная таблица VND_REC_STR, в которую загружаются все файлы с различными форматами.



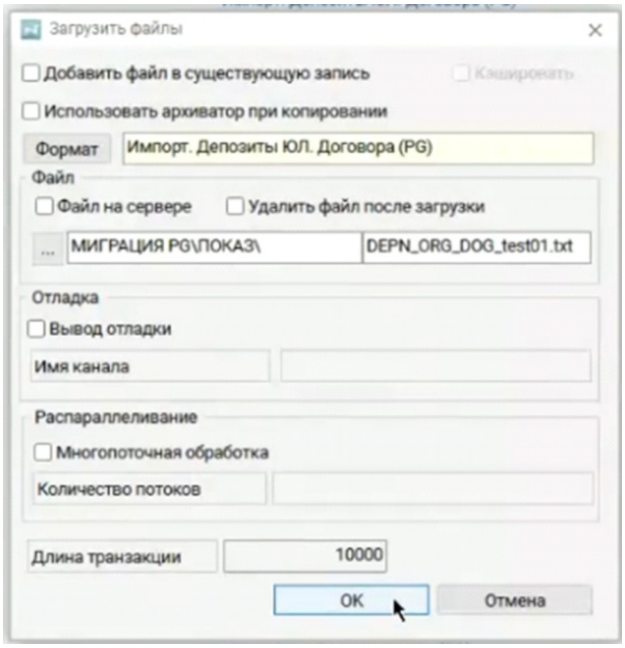
Этот файл загружается через операцию **«Загрузить файлы».**



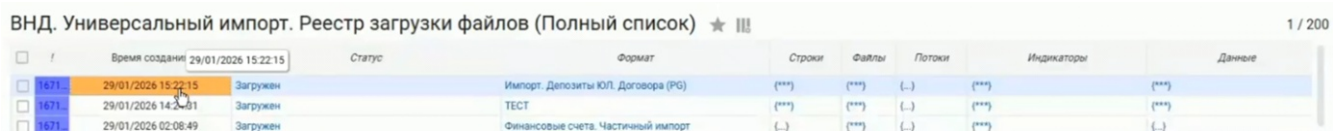
Загрузка файла

Попробуем прогрузить файл «Карточка депозитов юридических лиц».

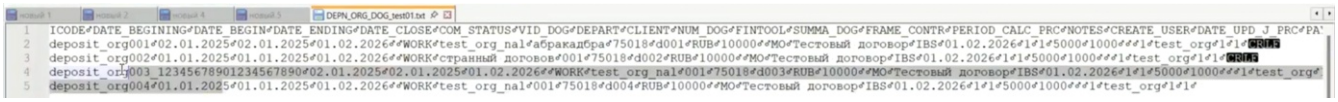
1. Выбираем нужный файл.
2. Формат автоматически определился по маске файла. Загружаем файл.



3. В реестре появляется запись сегодняшним числом. Отображается время создания (по местному часовому поясу), файлу присваивается статус Загружен. Формат определится.



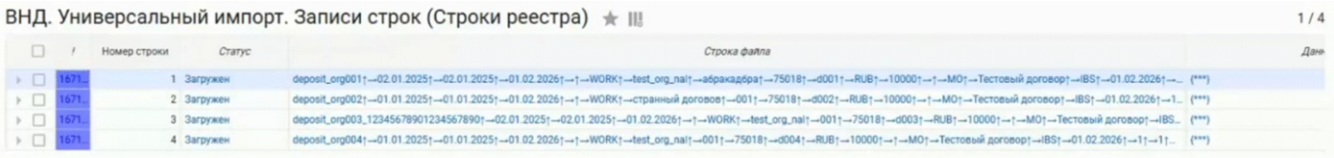
Так выглядит файл с данными, его содержимое:



Первая строка — заголовок (разделён спецсимволом). Далее представлены сами данные (депозит №1–№4)

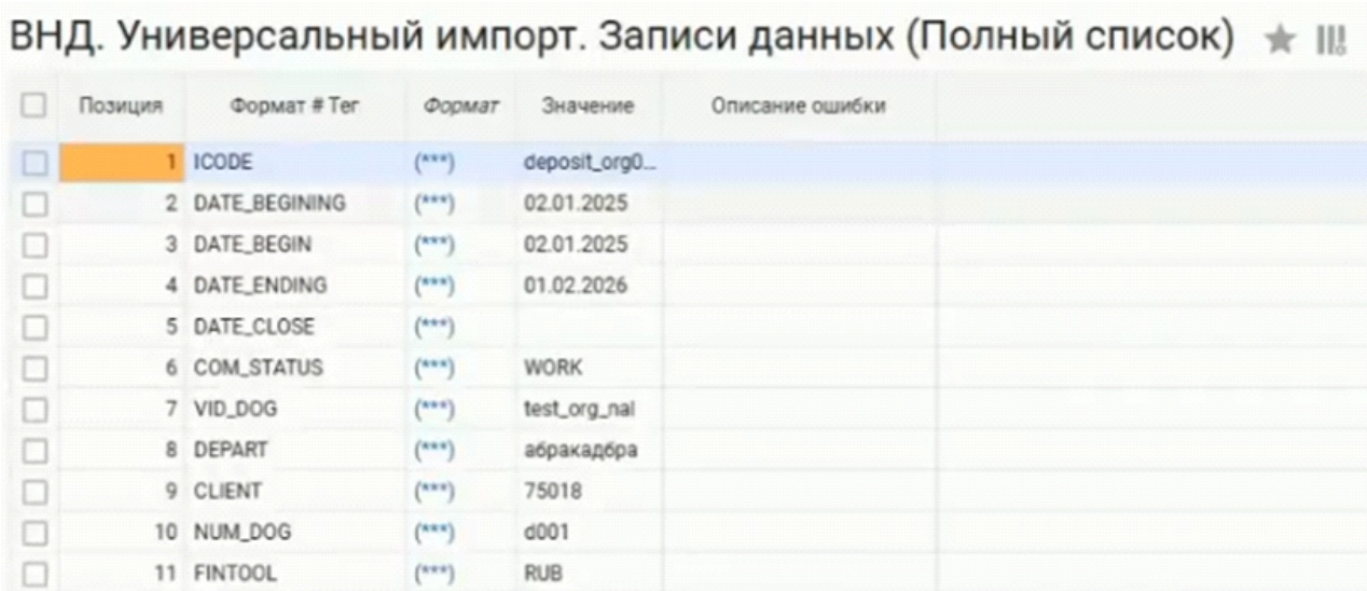
Первое поле — это **внутренний код импорта (ВКИ)**. Служебный реквизит, который используется при импорте в большинстве продуктов. По нему определяется уникальность записи (номера договоров и другие реквизиты не всегда могут быть уникальны).

Справочник выглядит следующим образом:



В справочнике хранится неразобранная строка файла. Служебные символы-стрелки в ней — места, где в данных файла стоит разделитель. По строке можно увидеть, как прошло разделение между всеми реквизитами.

В представлении есть данные:



12	SUMMA_DOG	(***)	10000	
13	FRAME_CONTR	(***)		
14	PERIOD_CALC_PRC	(***)	МО	
15	NOTES	(***)	Тестовый д...	
16	CREATE_USER	(***)	IBS	
17	DATE_UPD_J_PRC	(***)	01.02.2026	
18	PAY_ADD_PRC	(***)	1	
19	PRC_TO_ACCOUNT	(***)	1	
20	NOT_USE_SUMM	(***)	5000	
21	LIMIT_OST	(***)	1000	
22	NEXT_DOG	(***)		
23	PREV_DOG	(***)		
24	STOP_INT_CALC	(***)	1	
25	VID_DOG_PROL	(***)	test_org	
26	WORK_AT_TAKE	(***)	1	
27	GUARANTEE_DEP	(***)	1	

Здесь видно, как конкретная запись разложилась по реквизитам. Это данные внутри буферной таблицы, которые пришли с файлами. Можно увидеть, правильно ли отображается разделитель, что пришло в конкретном реквизите и т.д.

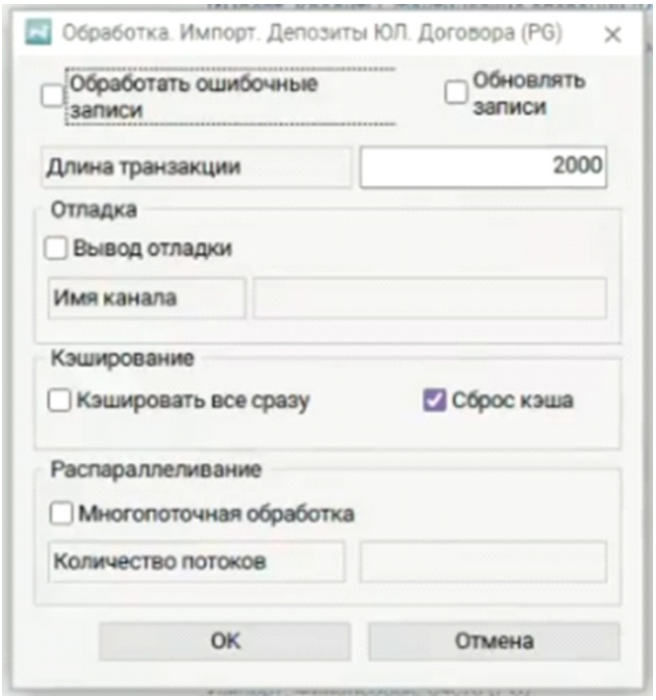
Обработка данных

1. Откроем представление (например, «Депозиты юридических лиц»).

2. Установим фильтр по ВКИ, потому что ВКИ в данном случае специфический.

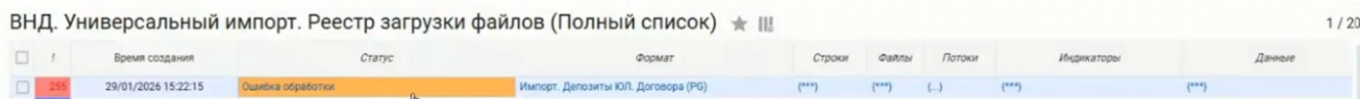
3. Загруженных данных (карточек договора с депозитом) пока нет. Произведём обработку реестра. Операция **Обработать** вызывает работу операции импорта. Она произведёт обработку операции импорта, которая разложит данные из буферной таблицы в зет-решётке. На данной операции можем увидеть опции:

- Обработать ошибочные записи — не выбираем, если все строки в состоянии загружены и ошибок нет.
- Обновлять записи — применимо, если данные уже есть; если в операции импорта уже заложено обновление данных, то данные можно прогрузить поверх.
- Длина транзакции — через сколько записей будет производиться коммит в базу данных.
- Вывод отладки — применимо, если нужно посмотреть, что происходит при обработке.

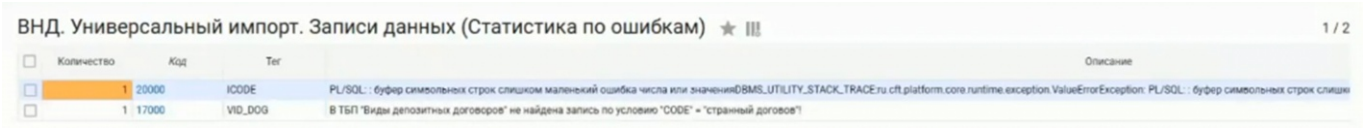


- Кэшировать всё сразу — к этому параметру нужно подходить осторожно. В карточке депозита есть ссылки на подразделения, на виды договоров, на клиента. Чтобы карточка депозита загрузилась, нужно, чтобы все связанные данные уже были в системе. Например, если не найдётся клиент, то и депозит будет загружен, (если данные не найдены, значит запись некорректна, ее нельзя создавать). Все связанные данные, которые используются при загрузке данного формата, можно закэшировать в памяти. После этого поиск данных будет осуществляться непосредственно в оперативной памяти. Это позволяет ускорить загрузку.
- Многопоточная обработка — применимо, если количество данных достаточно большое. В этом случае поднимутся текстовые задания, и каждое из них будет делать обработку. При этом данные всего файла порежутся на фрагменты между этими заданиями, и каждый Job будет обрабатывать только свой фрагмент. Это образует распараллеливание обработки, что увеличивает скорость загрузки файла в систему. В нашем случае ставить не будем, потому что файл небольшой.

4. Нажимаем **ОК**. Статус реестра поменялся на **Ошибку обработки**.

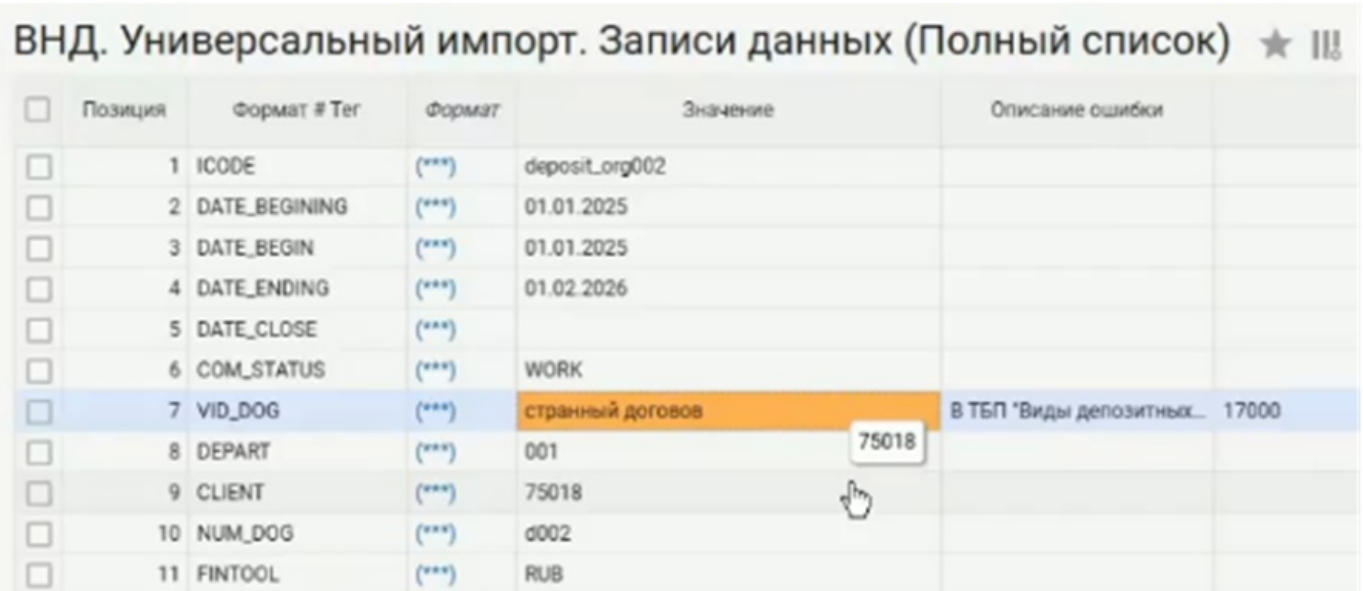


5. Ищем причину ошибки. Перейдем в массив **Статистика по ошибкам** и увидим перечень ошибок.

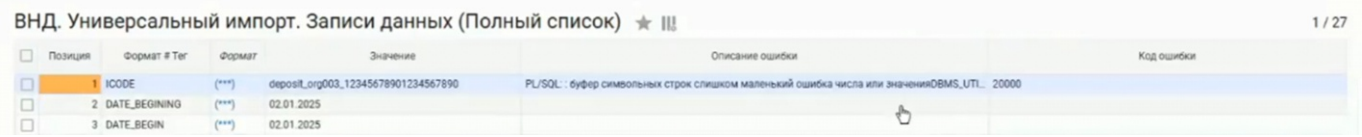


6. Перейдем на строку реестра, где найдена ошибка.

Первая ошибка типичная: в выгрузку пришло какое-то некорректное значение (в справочнике видов договоров по депозитам этого значения нет).



Вторая ошибка: в ВКИ договора попало очень длинное поле:



Для устранения ошибок предусмотрена функция **Экспорт списка ошибок**. Можно сообщить о некорректном значении и предложить выгрузить повторно.

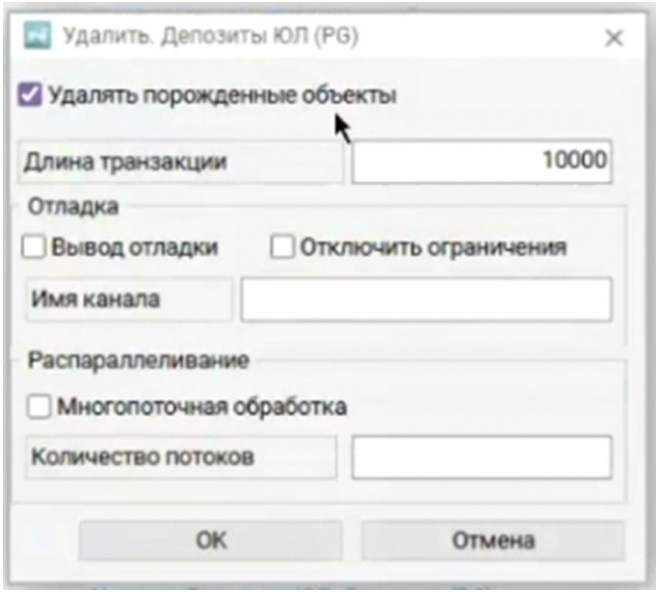
Ограничение нашего инструмента импорта заключается в том, что один файл — это один тип базового понятия (ТБП) либо одна таблица. В данном случае в файле присутствуют только ТБП «Депозиты юридических лиц». Если нужно грузить массив «Счета договора» и остальные массивы, это необходимо делать отдельными форматами, и под это нужен отдельный файл.

Это позволяет быстрее грузить и легче анализировать ошибки. Самые большие проблемы при загрузке фактических операций в том, чтобы всё сходилось, множилось, поэтому каждая отдельная сущность — это отдельный файл и отдельный формат.

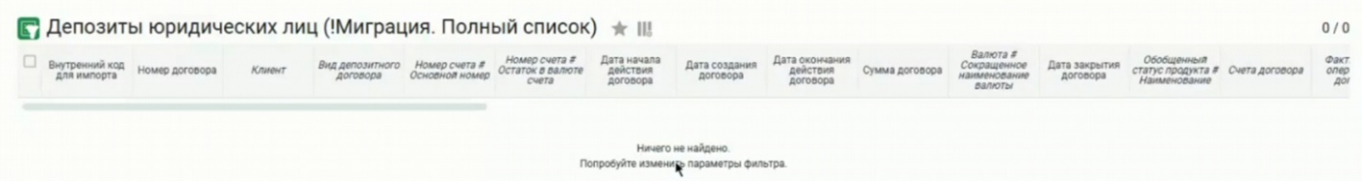
Удаление данных

Поскольку миграция отлаживается итерационным путём, на нашем инструменте импорта формируются не только операции загрузки, но и операции удаления загруженных данных. Эта операция называется **Откат обработки.**

Операция удаления не будет выполнена, если есть связанные объекты.



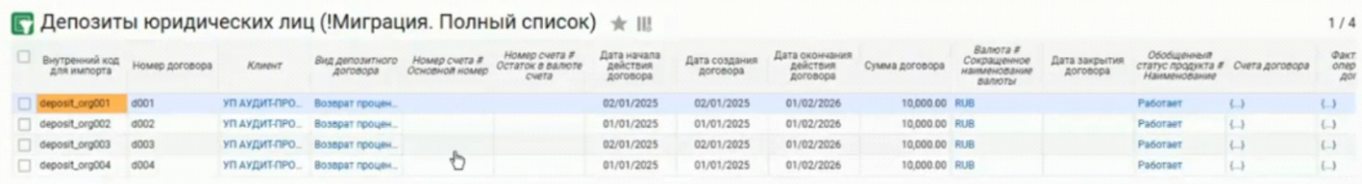
После удаления реестр и строки по нему вернутся в состояние **Загружен.** Если обновить представление, можно увидеть, что из представления данные пропали.



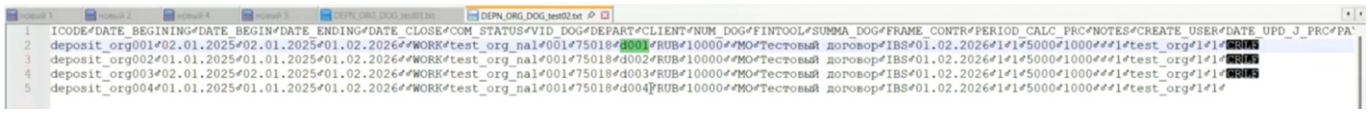
Обновление загруженных данных

В функционале есть возможность обновить загруженные данные. Рассмотрим на примере нового файла без ошибок.

1. Загружаем файл.
2. Запускаем обработку. Файл обработался, появились данные:



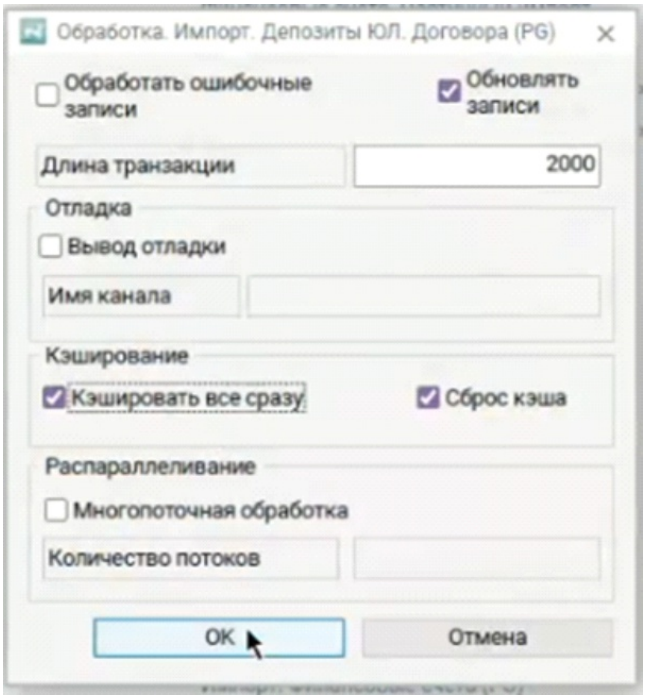
3. Представим, что мы выгрузили данные, но ошиблись при выгрузке номера договора. В таком случае в уже загруженных файлах нужно исправить номера договоров:



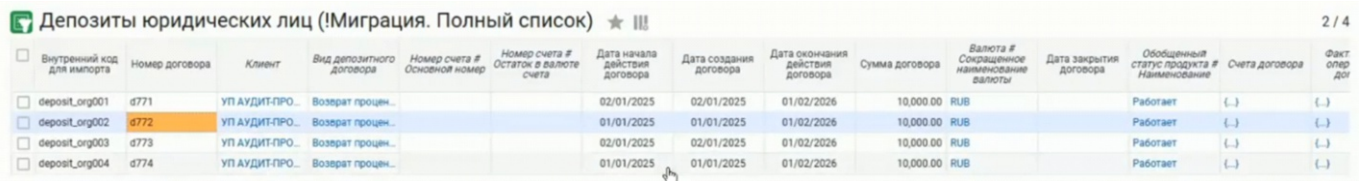
Исправляем номера договоров (например, заменяя «00» на «77») и сохраняем:

Таким образом, мы не откатываем реестр, просто грузим новый файл поверх. Теперь, если перейти в данные, можно увидеть, что номер договора исправлен. При этом в базе данных загружен старый файл с «00» в номере договора.

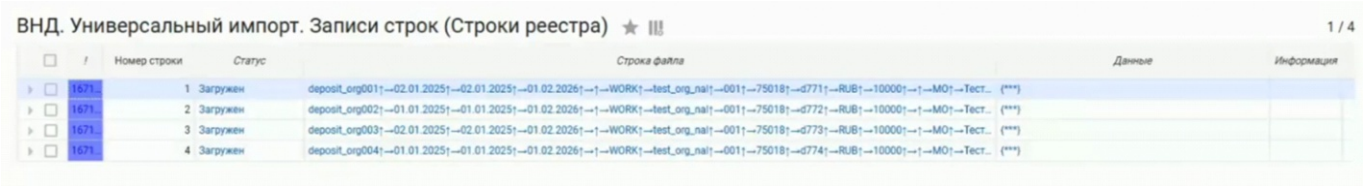
4. Нажимаем **Обработать** и отмечаем **Обновить записи.** Поскольку у старого и нового договора один ВКИ, система считает их за один файл.



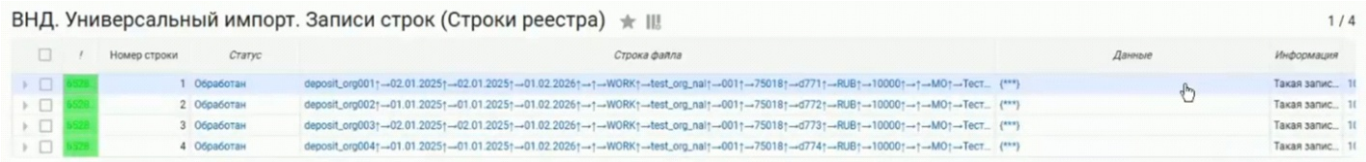
Статус записей сменился на **Обработан**, а в столбце Информация появилась запись — **Запись обновлена**. Номер договора поменялся:

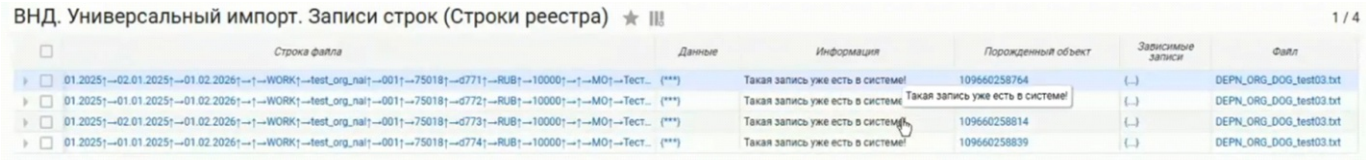


- Если не нужно, чтобы данные обновлялись, можно откатить реестр без удаления. Записи в базе данных останутся, а реестр перейдёт в состояние **Загружен.**



- Если при повторной обработке не ставить отметку **Обновлять записи,** будет указано, что такая запись уже есть в системе





Работа по аналитике загруженных данных, выгрузке списка ошибок и прочее может проводиться человеком, который не разбирается в программировании. Достаточно понимать основные принципы устройства АБС ЦФТ-Банк. Инструмент работает в рамках существующей развёрнутой инфраструктуры АБС ЦФТ-Банк и не требует специфических приложений и функционала.

Порядок работы с инструментом импорта

Операции порождения/обновления объектов в АБС ЦФТ-БАНК

При генерации операции импорта с помощью инструмента импорта нам важно достигнуть следующих целей:

- Загрузить все корректные данные, но при этом получить развернутую информацию об ошибках, которые встретились при загрузке.
- Производительность. Необходимо уменьшить обращение к базе данных, сделать их более точечными.

Макросы в операциях импорта

Все важные параметры для реализации импорта запрятаны в макросы — в них скрывается всё, что находится «под капотом» загрузки. Можно использовать существующие макросы либо генерировать макросы самостоятельно.

Макросы, которые объявляют таблицы кэширования

Таблицы кэширования — это таблицы, к которым мы обращаемся, чтобы найти записи в справочниках во время импорта:

```
public pragma include('[VND_UNIMP_FLOAD]::[LIB2]');
public pragma include('[VND_UNIMP_FLOAD]::[OPT_LIB]');
public pragma include('[VND_UNIMP_FLOAD]::[LIB_CACHE]');

&decl_tbl(CR_RISK_GR_TBL, '::[CR_RISK_GR], (::[PR_CRED] all : cred) all', 'x%collection||"#"||to_char(x.DATE_START, ::[VND_UNIMP_FLOAD].[LIB

lock_obj    boolean;
bExists boolean;

&decl_tbl(USER_TBL, '::[USER]', 'UNI_CODE', '', 'true')
&decl_tbl(CRED_RISK_TBL, '::[CRED_RISK]', 'GROUP_NUM', '', 'true')
&decl_tbl(SERVICE_QUAL_TBL, '::[SERVICE_QUAL]', 'SIGN', '', 'true')

&decl_val_tbl2(PR_CRED_TBL, '::[PR_CRED]', 'x.[INTERNAL_CODE]', '', 'GR_RISK_HIST', 'V_CACHE')

&declare_rt2(OBJ_INS, CR_RISK_GR)
&declare_rt2(OBJ_UPD, CR_RISK_GR)

IPG_PR_CRED_RISK (
    @name('Обработать ошибочные записи') P_PROC_ERROR in [BOOLEAN] default null
) is

validate is
begin
    if P_MESSAGE = 'DEFAULT' then
        lock_obj := true;
        V_CACHE := true;

        V_COMMIT_PART := 2000;
        V_DROP_CACHE := true;
        P_PROC_ERROR := this.[STATE] = &ulib.stErrorProc;
        P_INFO := 'OK,%VAR%.V_DEBUG,%VAR%.V_MULTI_THREAD';
    end if;
end;
```

В данном примере мы выполняем импорт ТБП группы кредитного риска — данная операция предназначена для того, чтобы заполнить массив **История групп риска**, продукт Кредит.

В процессе импорта этих данных потребуется информация о пользователях, о справочнике группы кредитного риска, качестве обслуживания долга и так далее. Поэтому необходимо объявить таблицы кеширования при помощи макросов либо **decl_tbl**, либо **decl_val_tbl.**

Макрос **decl_tbl** — это, по сути, словарь, пара ключ-значение.

Макрос **decl_val_tbl** — это макрос, который позволяет закэшировать больше реквизитов. Представляет собой не просто одно значение по ключу, а набор значений.

Макрос CACHE

Здесь происходит заполнение таблиц кэширования:

```
&CR_RISK_GR_TBL_$CACHE("Принадлежность к группе кредитного риска")
&PR_CRED_TBL_$CACHE("Кредиты")
&CRED_RISK_TBL_$CACHE("Группы кредитного риска")
&USER_TBL_$CACHE("Пользователи")
&SERVICE_QUAL_TBL_$CACHE("Качество обслуживания долга")
```

Цикл, внутри которого мы бежим по реестру, считываем строки из реестра и построчно выполняем обработку:

```
begin
    idx_rec := &InitRec(x.f_ref, &ulib.stProcessed);

    idx_ins := null;
    idx_upd := null;

    --ICODE
    tmp_str := &GetStr('ICODE');
    if not tmp_str is NULL then
        if &PR_CRED_TBL_$EXISTS(tmp_str) then
            oRec%collection     := &PR_CRED_TBL_$GET(tmp_str, 'GR_RISK_HIST');
        else
            &SetErrRec(idx_rec,  'В ТБП "'||::[PR_CRED]%classname||'" не найдена запись по условию "INTERNAL_CODE" = "'||tmp_str
        end if;
    end if;

    oRec.[DATE_START] := &GetDate('BEG_DATE');

    --oRec.[DATE_END] := nvl(&GetDate('END_DATE'), constant.date_max);

    oRec.[FIX] := &GetBool('FIX');

    oRec.[PRC_RESERV] := &GetNum('RES_RATE');

    tmp_str := &GetStr('RISK_GROUP');
    if not tmp_str is NULL then
        if &CRED_RISK_TBL_$EXISTS(tmp_str) then
            oRec.[RISK_GROUP]       := &CRED_RISK_TBL(tmp_str);
        else
            &SetErrRec(idx_rec,  'В ТБП "'||::[CRED_RISK]%classname||'" не найдена запись по условию "GROUP_NUM" = "'||tmp_str||
        end if;
```

Мы вычитываем из временной таблицы данные по тегам, которые есть в формате. Например, мы сохраняем дату / логический признак / процентную ставку и так далее. Необходимо заполнить переменную структуру (это переменная типа, который импортируется) значениями, которые мы хотим видеть в базе данных в АБС ЦФТ-Банк.

Для полей, которые являются ссылками, обратимся к кэш-таблицам. Для того чтобы искать в них данные, используется тоже соответствующий макрос. В данном случае необходимо искать запись в справочнике групп риска.

Все макросы реализованы через динамический PL+, они генерируют необходимый код, в зависимости от того, какое ТБП импортируется, и данный код исполняется. Соответственно, разработчику об этом думать не нужно.

Перекладывание готовых экземпляров ТБП, в данном случае CR_RISK_GR, во временную таблицу для вставки и непосредственно во вставку базы данных выполняется при помощи специального макроса.

```
declare
    oRec [CR_RISK_GR];
begin
    idx_rec := &InitRec(x.f_ref, &ulib.stProcessed);

    idx_ins := null;
    idx_upd := null;

    --ICODE
    tmp_str := &GetStr('ICODE');
    if not tmp_str is NULL then
        if &PR_CRED_TBL_$EXISTS(tmp_str) then
            oRec%collection     := &PR_CRED_TBL_$GET(tmp_str, 'GR_RISK_HIST');
        else
            &SetErrRec(idx_rec,  'В ТБП "'||::[PR_CRED]%classname||'" не найдена запись по условию "INTERNAL_CODE" = "'||tmp_str
        end if;
    end if;

    oRec.[DATE_START] := &GetDate('BEG_DATE');

    --oRec.[DATE_END] := nvl(&GetDate('END_DATE'), constant.date_max);

    oRec.[FIX] := &GetBool('FIX');

    oRec.[PRC_RESERV] := &GetNum('RES_RATE');

    tmp_str := &GetStr('RISK_GROUP');
    if not tmp_str is NULL then
        if &CRED_RISK_TBL_$EXISTS(tmp_str) then
            oRec.[RISK_GROUP]       := &CRED_RISK_TBL(tmp_str);
        else
            &SetErrRec(idx_rec,  'В ТБП "'||::[CRED_RISK]%classname||'" не найдена запись по условию "GROUP_NUM" = "'||tmp_str||
        end if;
    end if;

    oRec.[CALC_RISK_GROUP] := oRec.[RISK_GROUP];

    tmp_str := &GetNum('SERVICE_QUAL');
    if not tmp_str is NULL then
        if &SERVICE_QUAL_TBL_$EXISTS(tmp_str) then
```

В конце итерации этого цикла необходимо переложить готовый экземпляр во временную таблицу — это таблица в памяти.

```
    oRec.[IGNORED] := &GetBool('IGNORED');

    tmp_str := oRec%collection||'#'||to_char(oRec.[DATE_START], &ulib.DFORMAT);

    if not &ItErrRec(idx_rec) then
        --#ifdef Java
            ::[RUNTIME].[PLP2].REGISTER_VARIABLE('oRec', oRec);
        --#endif
        if &CR_RISK_GR_TBL_$EXISTS(tmp_str) then
            &SetObjRec(idx_rec, &CR_RISK_GR_TBL(tmp_str))

            idx_upd := &OBJ_UPD_$CNT + 1;

            if V_UPD then
                oRec%id  := &GetObjRec(idx_rec);

                OBJ_UPD%init(oRec, true, idx_upd);

                if not &ItErrRec(idx_rec) then
                    &SetInfoRec(idx_rec, 'Запись обновлена!')
                end if;
            else
                &SetInfoRec(idx_rec, 'Такая запись уже есть в системе!')
            end if;
        else
            idx_ins := &OBJ_INS_$CNT + 1;

            &OBJ_INS_$INIT(oRec)

            oRec%id := &cache.getId();
            &SetObjRec(idx_rec, oRec%id)

            OBJ_INS%init(oRec, true, idx_ins);

        end if;
    end if;
exception when others then
    &SetErrRec(idx_rec, utils.error_stack)
end;
```

Если вы дошли до этого этапа, это означает, что все бизнес-проверки пройдены: найдены все ссылки, проверены все обязательные поля.

Для проверки обязательности полей не нужно писать код. Обязательность полей объявляется на уровне формата. Если какое-то из полей будет не заполнено, об этом будет сигнализировано пользователю.

Макросы INS и UPD

По прошествии определенного количества строк (в данном случае 2 000 строк) выполняется вставка или обновление. Для этого есть отдельная процедура, которая представляет из себя два макроса, **INS** и **UPD**, в которых спрятана работа по пакетной вставке или обновлению данных:

```
execute is
  idx_ins     integer;
  idx_rec     integer;

  tmp_str     varchar2(4000);

  idx_upd     integer;

procedure commit_proc is
begin
    &OBJ_INS_$INS
    &OBJ_UPD_$UPD('FIX,PRC_RESERV,CALC_RISK_GROUP,RISK_GROUP,USER_MODIF,SERVICE_QUAL,IGNORED')
    &SaveRecs
    &d('Обработано - '||idx_rec||' записей за '||&GetTimeStr('Обработка')||'. Всего обработано - '||&all_cnt||' записей, из них с ошибками -
end;
```

Генерируется динамический код, который в случае, например, с депозитами юрлиц делает вставку в три таблицы, которая необходима.

Так генерируется практически готовая операция, которую можно доработать при необходимости.

Генерация идентификаторов

Для генерации идентификаторов, которые нужны при загрузке, необходимо знать ID порождённой записи, потому что он сохраняется в реестр и потом его оперируем при откате. Это выполняется при помощи обращения к **сиквенсу Sec ID.**

Для повышения эффективности этих обращений, чтобы не обращаться каждый раз к сиквенсу за следующим ID, используется отдельный сиквенс для миграции. Есть специальная служебная операция, которая выделяет пул идентификаторов, которые будут использоваться непосредственно для миграции из числа тех ID, которые могут быть сгенерированы сиквенсом Sec ID.

При импорте кэшируется обращение к данному сиквенсу, чтобы не обращаться каждый раз за каждым идентификатором, тем самым уменьшаем количество тех.обращений.

Отключение триггеров при удалении

В случаях, когда на данные ТБП потенциально может быть большое количество ссылок, есть возможность ускорить откат объектов при помощи отключения ограничений. Это актуально, поскольку из таблиц, на которые есть множество внешних ключей, записи довольно долго удаляются за счёт того, что внутренняя реализация внешних ключей сделана через специальные системные триггеры PostgreSQL.

Снятие ограничения реализовано через следующую команду:

```
SET session_replication_role = 'replica'
```

В PostgreSQL триггеры имеют режим срабатывания, и по умолчанию триггеры находятся в режиме **origin**. Данная служебная команда предназначена для того, чтобы в системах логической репликации не срабатывали повторно триггеры, которые не должны этого делать.

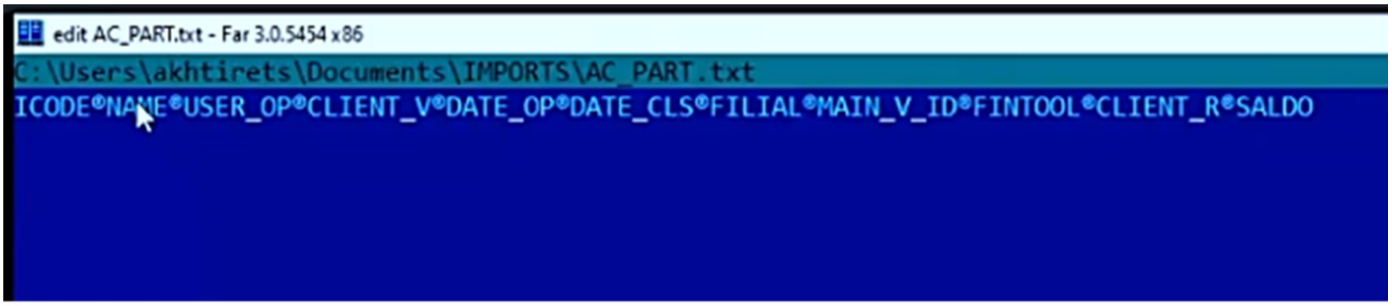
Данная команда срабатывает на сессию. Если включить эту команду, то в течение сессии будут срабатывать только те триггеры, которые находятся в режиме **replica**. Все триггеры, которые не находятся в режиме **replica**, перестают работать.

```
                order by rec.[POS] desc);
end if;
if P_DEL_OBJ then
    &StartTimeCnt('Удаление')
    &d('Удаление порожденных объектов...')
    v_cur_del.fetch(RECS%id, RECS.[STATE], RECS.[OBJ_REF], RECS.[INFO], OBJ_DEL%id);
    K := RECS%id.count;
    v_cur_del.close;
    ALL_CNT := ALL_CNT + K;

    ::[VND_UNIMP_FLOAD].[OPT_LIB].disable_trigger := V_DISABLE_TRG;
    &OBJ_DEL_$DELETE(ID)
    ::[VND_UNIMP_FLOAD].[OPT_LIB].disable_trigger := null;

    &RECS_$UPD('STATE,OBJ_REF,INFO')

    &FixTimeCnt('Удаление')
    &d('Обработано - '||K||' записей за '||&GetTimeStr('Удаление')||'. Всего обработано - '||ALL_CNT||' записей!')
    if not &ulib.cur_Thread is NULL then
        update u(u.[PROC_RECS] = u.[PROC_RECS] + K) in ::[VND_THREADS] all where u = &ulib.cur_Thread;
    end if;
```

Это действие применимо только на сессию и только на протяжении процедуры отката экземпляров. Это позволяет существенно ускорить удаление таких объектов, как клиенты, счета и прочее.

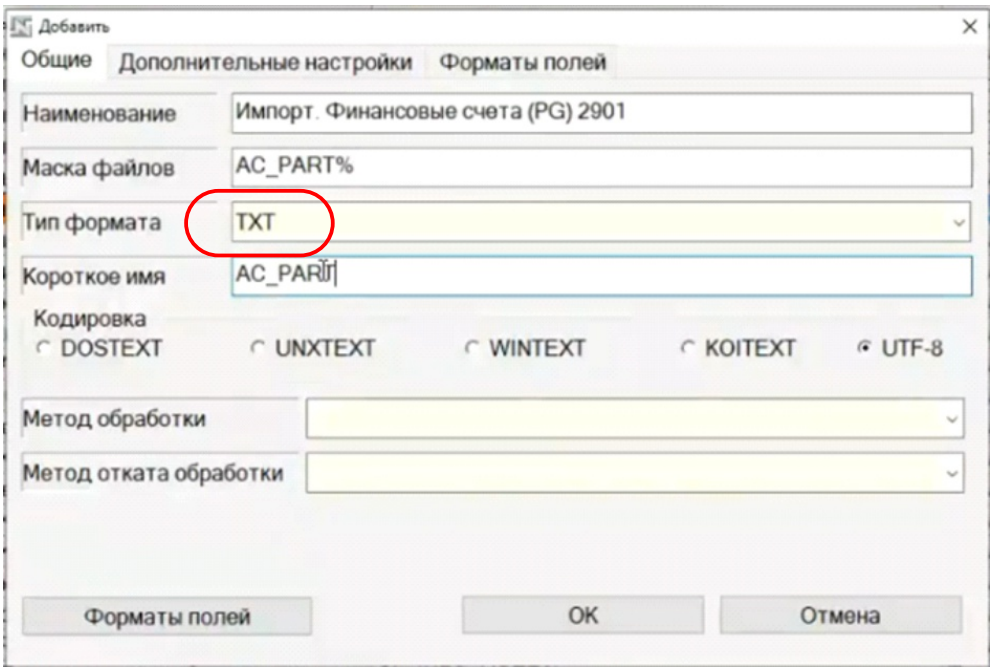
Разработка форматов импорта: от согласования формата импорта до получения операции

Подготовим файл AC_Part.txt, который состоит из одной строчки (набор реквизитов финансовых счетов).

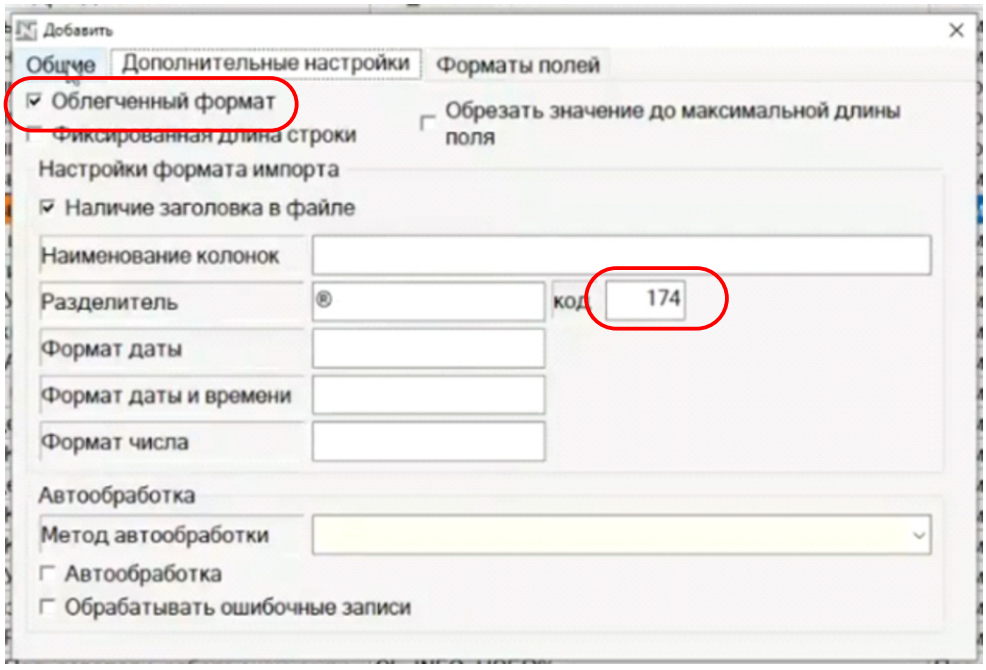


Чтобы загрузить файл, нужно завести заголовок формата.

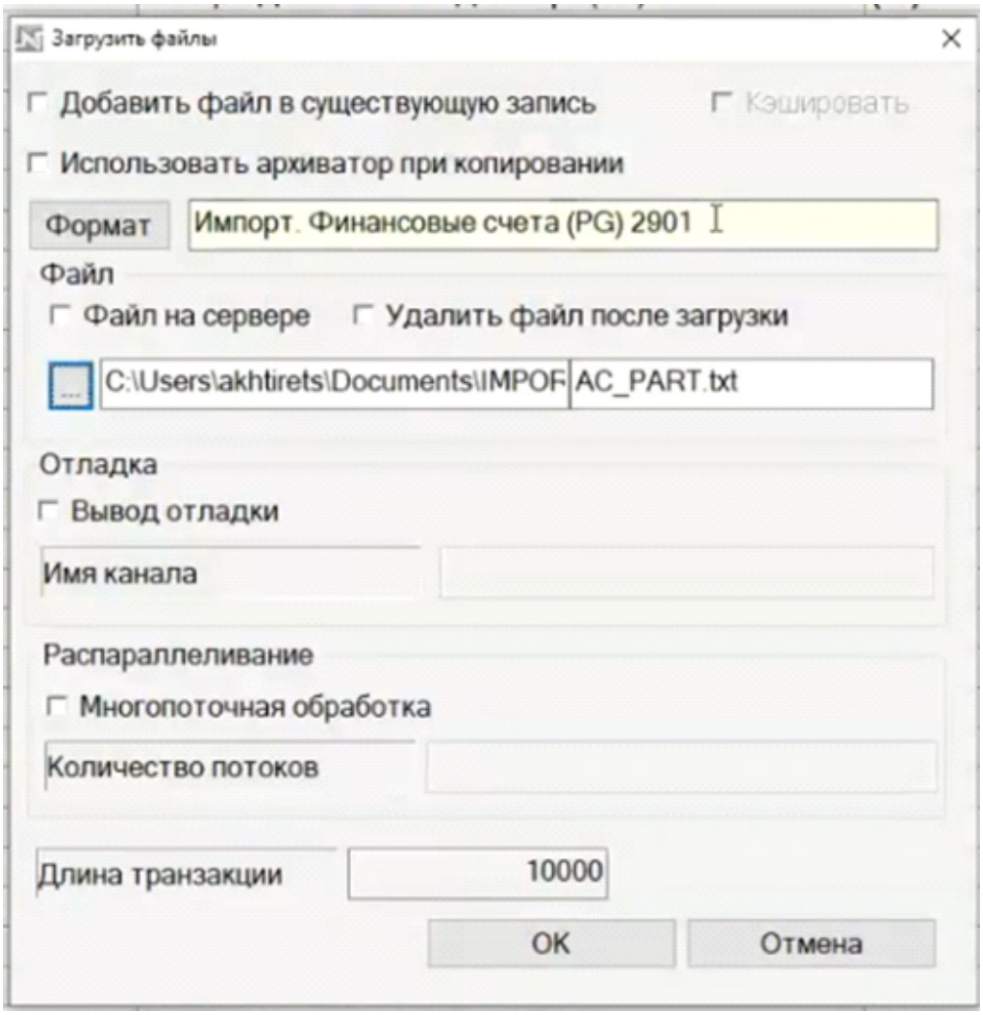
1. В справочнике заходим в режим Форматы файлов.
2. Добавляем формат:



3. В Дополнительных настройках отмечаем **Облегченный формат** и добавляем код разделительного знака:

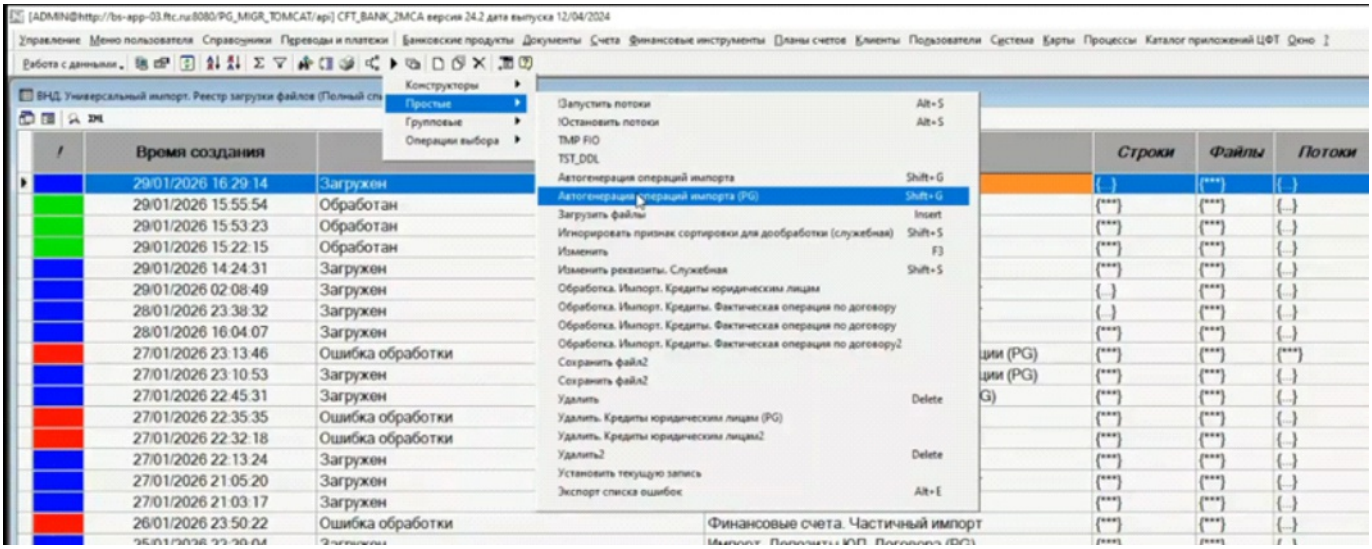


4. Переходим в режим Реестра загрузки файлов и загружаем файл. Формат определился.

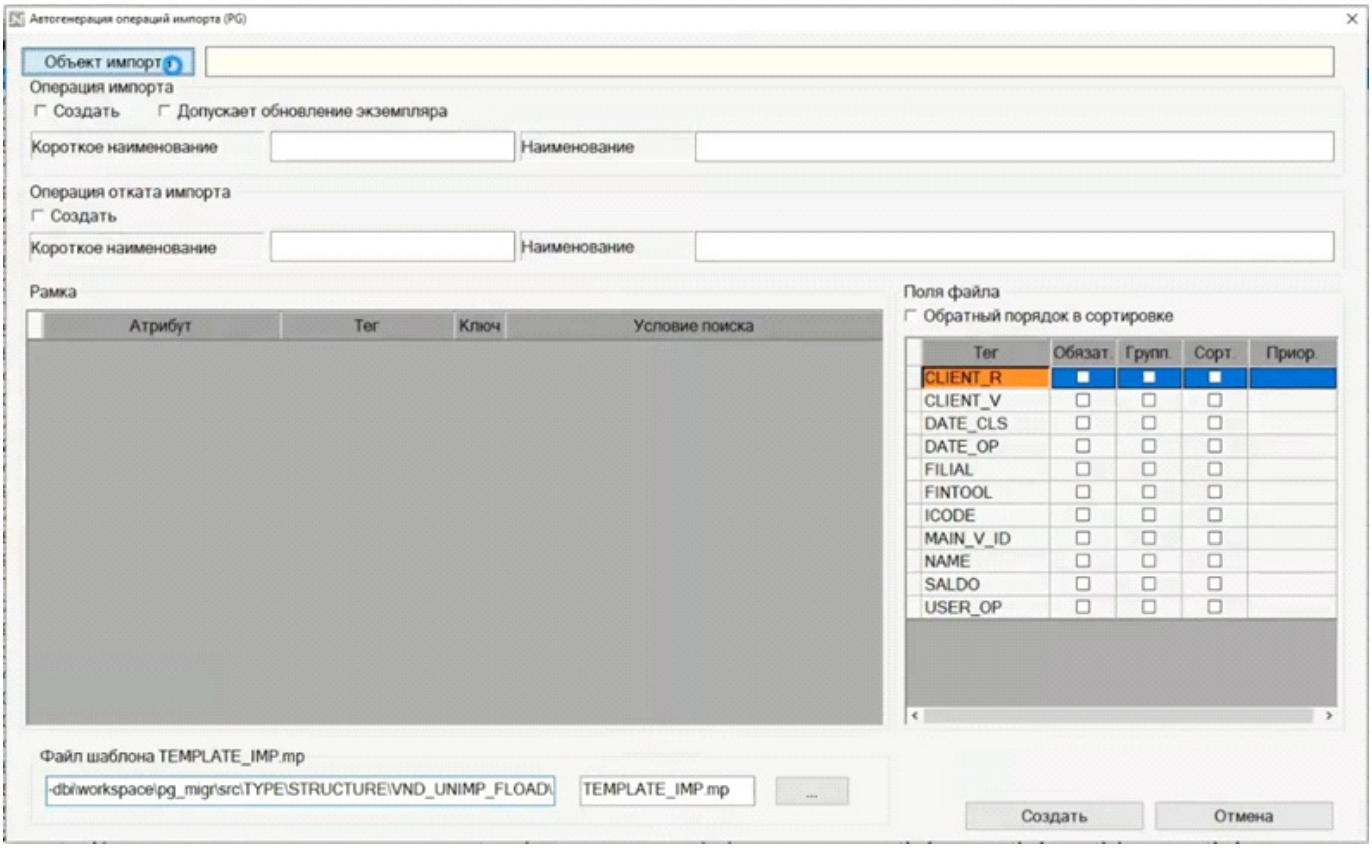


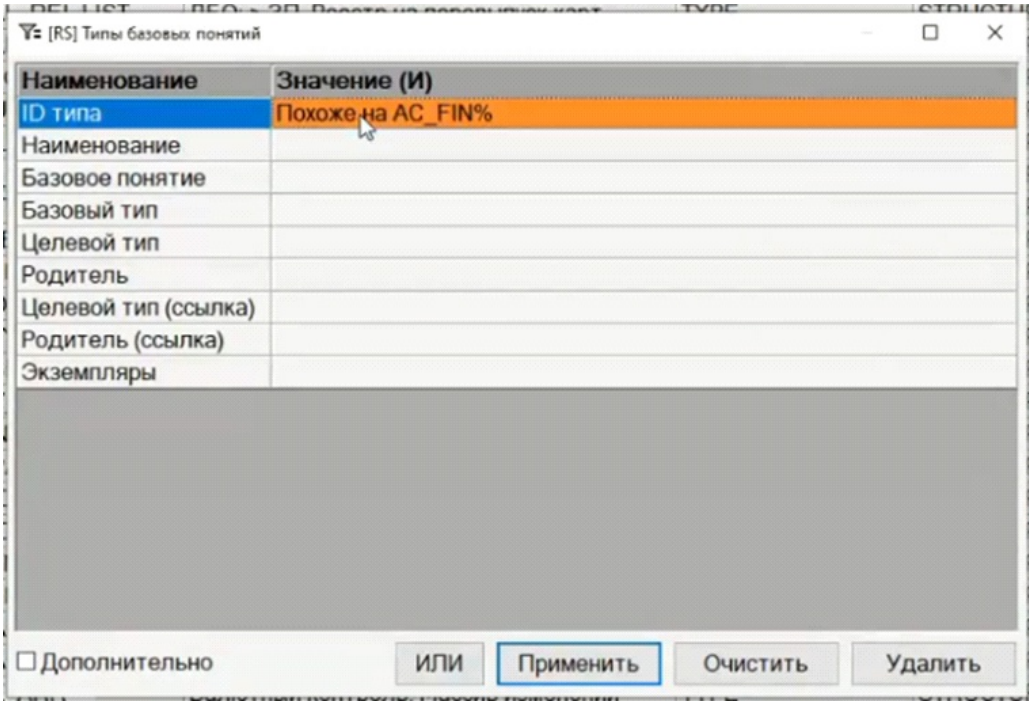
В массиве Форматы полей появились все реквизиты, которые перечислены в файле.

5. Генерируем операцию обработки. Выбираем операцию **Автогенерация операции импорта (PG):**

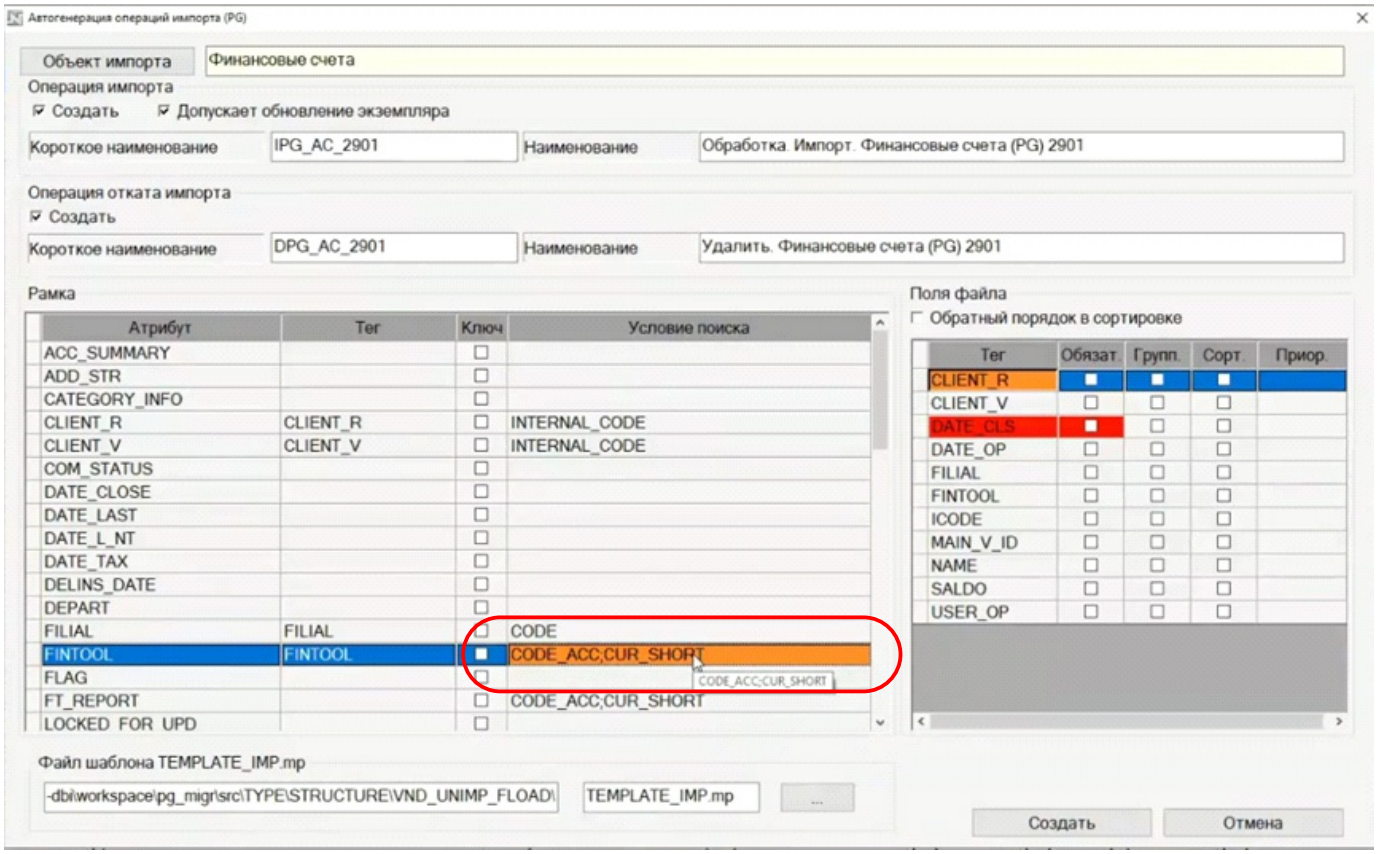


6. Указываем объект импорта. В нашем случае это финансовые счета. Соответственно, идентификатор типа **AC_FIN.**





7. Выбираем **Создать**, указываем, что данная операция допускает обновление экземпляра. Заполняем поля в гриде:

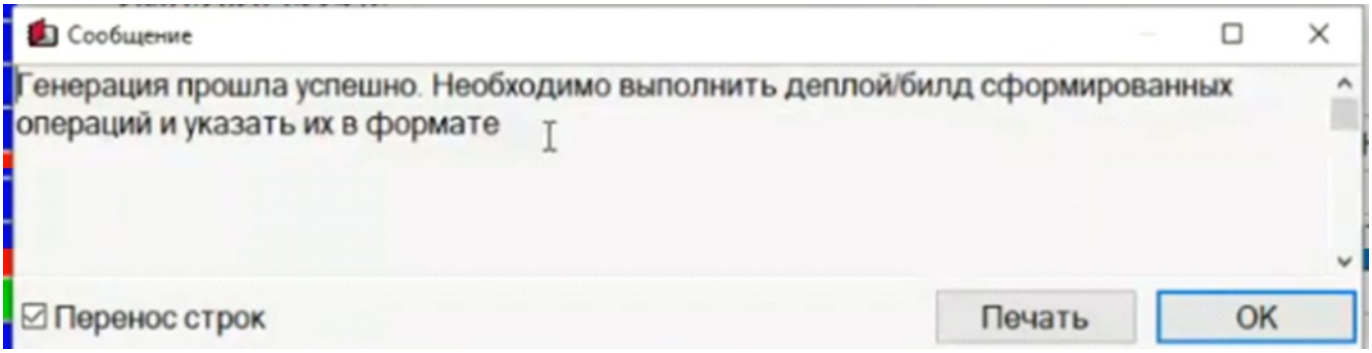


Теперь указано два реквизита. При загрузке в этом справочнике по переданному в файле значению будут искаться записи как по реквизиту кода **ACC**, так и по реквизиту **CUR_SHORT**.

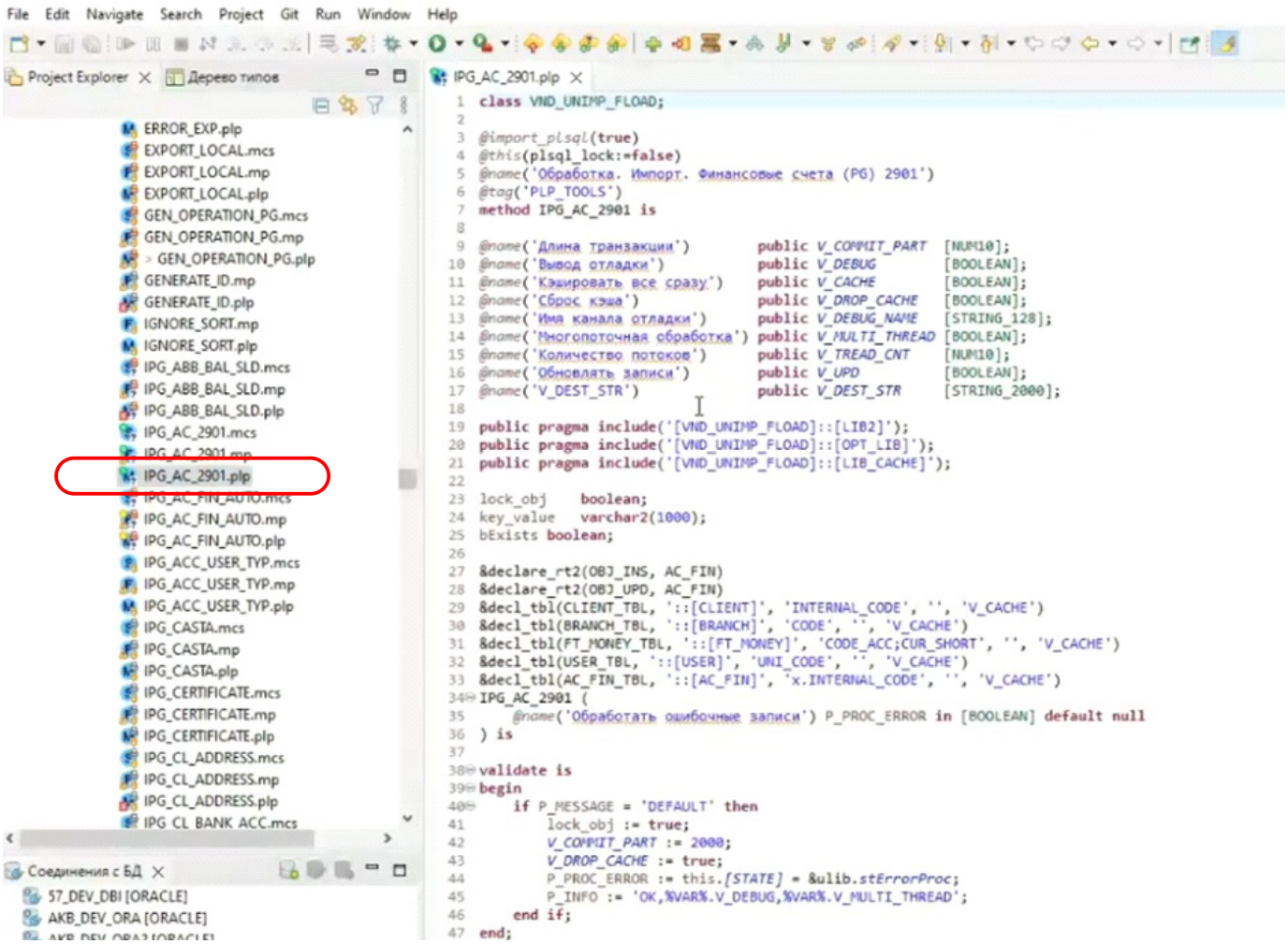
8. В следующем гриде для реквизитов файла есть обязательные поля, группировка по потокам, сортировка, приоритет. Непосредственно на саму операцию обработки заполнение этих полей не влияет, но здесь можно настроить формат. Если поле помечается как обязательное, то при загрузке в формате операция обработки увидит эту пометку, и, если реквизит **MAN_V_D** не заполнен, запись будет отбракована.

9. Файл шаблона используется как образец при автогенерации. Если этого файла нет в проекте, его необходимо импортировать со схемы. Путь указывает, в каком проекте будет создана операция.

10. Нажимаем **Создать**. Получаем сообщение, что генерация прошла успешно:



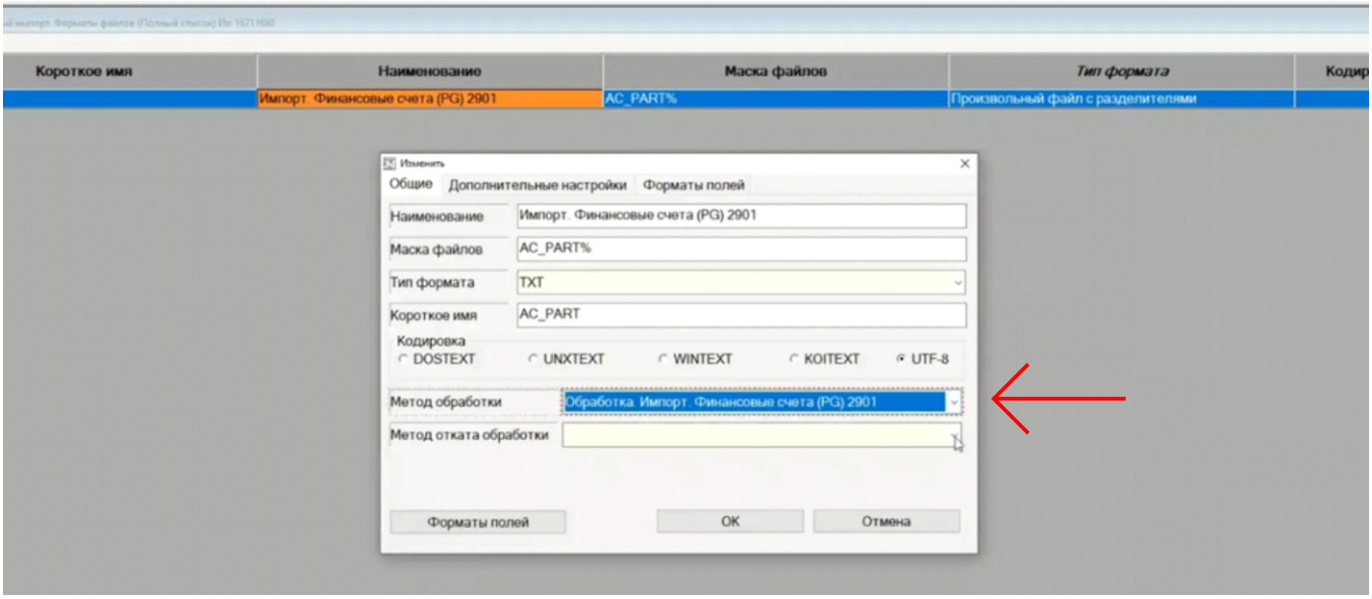
Чтобы проверить, действительно ли генерация прошла успешно, нужно зайти в A2. Там можно увидеть, что создана операция **IPG_AC_2901.plp.**



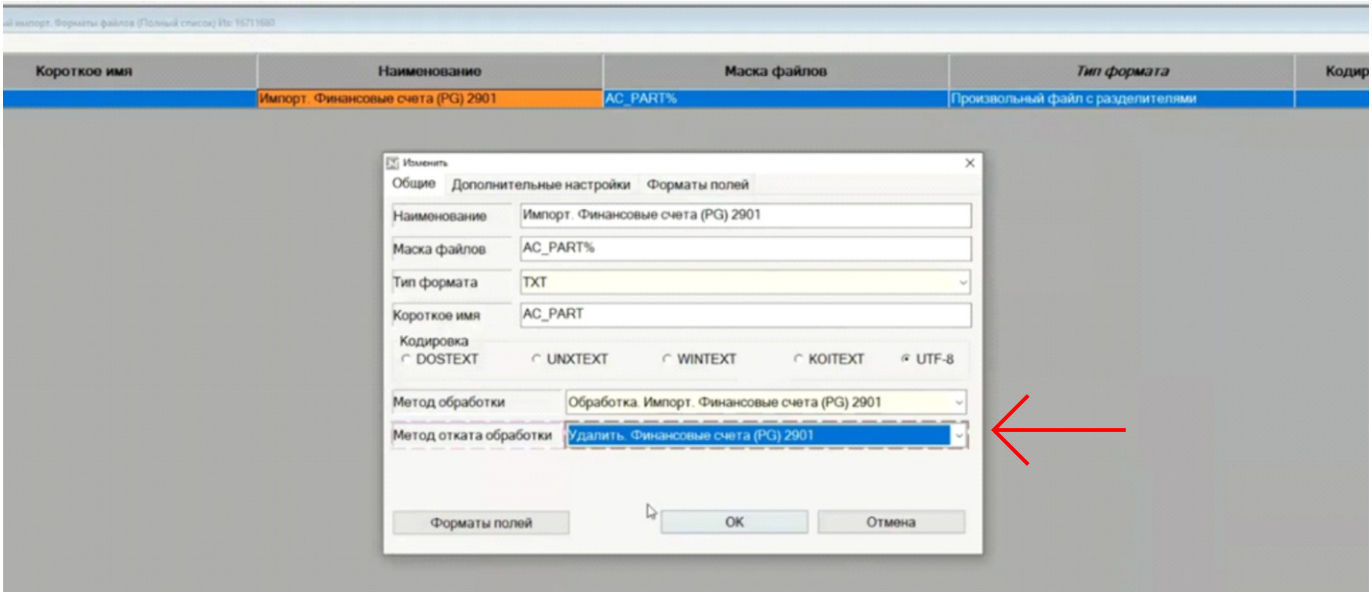
Появились справочники, которые будут закэшированы, клиенты, подразделения, валюты, пользователи и сами финансовые счета, которые будут импортироваться.

Операция обработки и операция удаления созданы и готовы к многопоточности.

11. Через навигатор, подключенный к локальному серверу подключений, переходим в формат, вызываем его на коррекцию. В методе обработки, указываем нашу операцию обработки:



И операцию удаления:



Мы готовы получить файл с данными, который можно грузить и обрабатывать.

Результаты нагрузочного тестирования инструмента файлового импорта

В результате нагрузочного тестирования получили следующие результаты:

N	Формат	Размер файла (МБ)	Количество записей в файле	Действие	Количество потоков	Особые параметры запуска	Время работы
1	Импорт. Клиенты. Физические лица (PG)	99,4 МБ	351,168	Загрузка			18 с
			(загружено без ошибок 303,712)	Обработка	8	Кэшировать сразу = Да	2 мин 3,74 с
				Обновление	8	Кэшировать сразу = Да	2 мин 18,64 с
						Обновлять записи = Да	
				Удаление	8	Без отключения триггеров	27 мин 19,87 с
				Удаление	8	С отключением триггеров	2 мин 40 с
2	Импорт. Документы. Платежные документы (PG)	6,369 Гб	10,862,122	Загрузка		Архивация и выкладка на ФИО	14 мин 38,49 с
		одним файлом, либо разбит на шесть	(загружено без ошибок 10,402,300)			9 минут	
				Обработка	10	Кэшировать сразу = Да	1 ч 53 мин
				Удаление	10	Без отключения триггеров	3 ч 44 мин
				Удаление	10	С отключением триггеров	16 мин 56,5 с

*Оборудование, на котором проводилось тестирование:

- Сервер БД Postgres: 4 физических ядра x86 частотой не меньше 2 ГГц, 32 Гб оперативной памяти
- Дисковая система: 500 Гб с возможностью расширения под системное ПО, под базу данных в зависимости от её объема
- ПО БД: PostgreSQL FT Data 16.6.2.2-1, ЦФТ-Банк с версией ПЯ 25.2
- Сервер приложений: 12 физических ядер x86 частотой не меньше 2ГГц, 80 Гб оперативной памяти, 500 Гб под системное ПО

Были загружены две основные сущности: карточки клиентов («Клиенты. Физические лица») и платёжные документы.

1 Загрузка карточек клиентов

Количество записей в файле:
350 000.

Объем файла:
100 МБ.

Итог:

- загрузка заняла секунды.
- На удаление 350 000 записей ушло 30 минут без отключения триггеров и **3 минуты с отключением триггеров.**

2 Загрузка платёжных документов

Количество записей в файле:
более 10 миллионов.

Объем:
более 6 ГБ

Итог:

- Загрузка вместе с обработкой заняла около 2 часов.
- Удаление заняло около 4 часов без отключения триггеров и **16 минут с отключением триггеров.**

Интеграция инструмента с СУБД FTData

Использование СУБД FTData позволяет адаптировать существующие инструменты импорта под отечественную СУБД, обеспечивая совместимость и стабильность работы с существующей платформой. Реализованные в СУБД FTData относительно PostgreSQL изменения минимизируют сложности при миграции на АБС ЦФТ-банк и ее последующей эксплуатации.

Высокая скорость загрузки данных

Совместное применение инструмента файлового импорта АБС ЦФТ-банк и СУБД FTData, включая использование буферных таблиц и минимизацию обращений к базе данных, позволяет переносить большие объемы данных в ходе миграции в короткие сроки.

Удобство анализа ошибок

Инструмент файлового импорта позволяет выявить и локализовать ошибки импорта при миграции для их последующей обработки.

Простота разработки и поддержки

Разработка инструмента ведётся на PL+, что позволяет разработчикам АБС ЦФТ-Банк легко понимать и модифицировать инструмент без необходимости разворачивать дополнительную инфраструктуру.

Эффективная обработка данных

СУБД FTData поддерживает структуру и функции, позволяющие инструменту файлового импорта разложить данные из буферной в рабочие таблицы, провести анализ и обработку ошибок с впечатляющей скоростью.