**Как Case Platform хранит информацию**

Начнем с описания того, как Case Platform "физически" хранит данные. Case Platform использует реляционные базы данных для хранения системной информации. В качестве систем управления базами данных (СУБД) могут выступать: **Oracle, PostgreSQL и Microsoft SQL Server**.

Системная информация, в свою очередь, хранится в двух разных базах данных (БД):

* первая – БД для локальных данных (localstorage) – предназначена для хранения информации из созданных приложений, т.е. из типов и процессов;
* вторая – БД для платформенных метаданных (platform) – предназначена для хранения настроек созданных приложений (например, форм, процесс, пользовательских функций и т.д.), параметров Platform, модели безопасности (например, подразделений, пользователей, должностей т .д.) и других сервисных метаданных.

Типовое название базы данных для локальных данных — *fislocal*, а для платформенных метаданных — *fisplatform*.

2 отдельные базы данных используются для **PostgreSQL** и **Microsoft SQL Server**. Для Oracle же используется **1 база данных**, но в ней создаются **2 отдельные схемы-пользователя**. Типовые названия схем-пользователей для Oracle — LOCALSTORAGE и FISPLATFORM.

Практически каждая таблица в базе локальных данных соответствует типу или процессу, созданному в приложении. Изменения в физической структуре базы данных (например, создание новой таблицы для нового типа, добавление столбца в таблицу для нового атрибута типа и т.д.) происходят в рамках инициализации приложения.

Со списком изменений физической структуры можно ознакомиться на вкладке «Изменения структуры БД», в свойствах приложения.

**Подключение к базе данных**

Часто при разработке и поддержке приложений возникает потребность “прямой” работы с БД, например: для отладки SQL-запросов, анализа и изменения данных из типов, мониторинга работы БД и т.д. в связи с этим требуется использовать отдельное программное обеспечение (ПО) для работы с БД.

В рамках данного урока будет рассмотрено подключение к трем разным СУБД, поддерживаемым Case Platform.

**Указанные данные на скриншотах являются только примерами, а сама инструкция по подключению демонстрирует порядок, но не призывает к действию.**

**1. Установка ПО**

Для начала необходимо установить программное обеспечение для подключения к БД.

**1.1. PostgreSQL**

Для работы с базой данных **PostgreSQL** необходимо загрузить и установить бесплатный клиент **pgAdmin**:  
<https://www.pgadmin.org/download/>

**1.2. Oracle**

Для работы с базой данных **Oracle Database**необходимо загрузить и установить **Oracle Instant Client. Пакет Basic Package**(набор библиотек, который позволяет подключиться к БД Oracle):   
<https://www.oracle.com/ru/database/technologies/instant-client/downloads.html>

После его установки необходимо скачать и установить бесплатный клиент **DBeaver Community Edition**:   
<https://dbeaver.io/download/>

**1.3. Microsoft SQL**

Для работы с базой данных **Microsoft SQL Server** необходимо загрузить и установить бесплатный клиент **Microsoft SQL Server Management Studio**:  
<https://docs.microsoft.com/ru-ru/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>

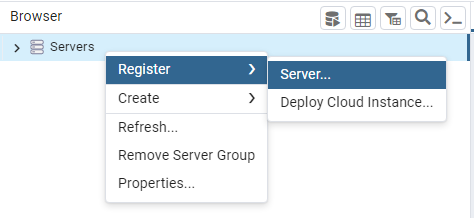
**Примечание**: Для подключения к базам данных Postgres и MS SQL можно воспользоваться также DBeaver. Специализированные клиенты для этих баз данных имеют более широкий функционал для работы с СУБД: графический план запросов, мониторинг работы БД, построение графиков, просмотр логов и т.д. Для базовой работы с СУБД: исполнение запросов, просмотр данных и т.д. можно воспользоваться клиентом DBeaver.

**2. Подключение к PostgreSQL**

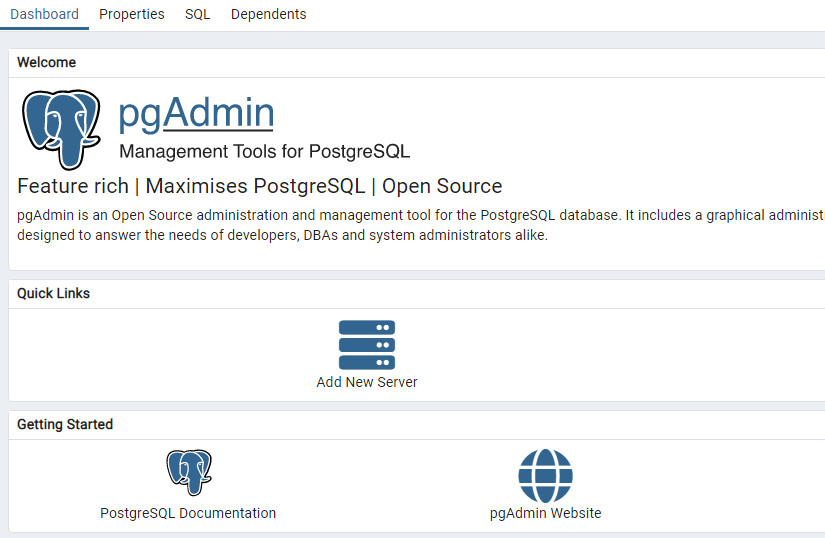
Для подключения к PostgreSQL необходимо:

1. Открыть приложение pgAdmin. **Важно:** при первом открытии приложения нужно обязательно установить мастер-пароль.

2. Нажать правой кнопкой мыши на раздел Servers, выбрать подпункт **Register → Server.**

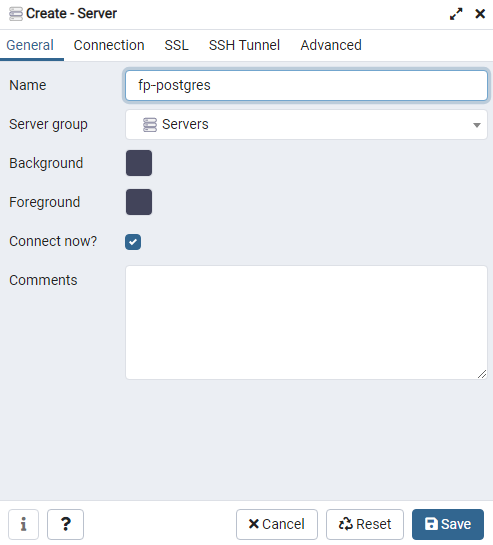


Или нажать на **Add New Server** в основной панели.



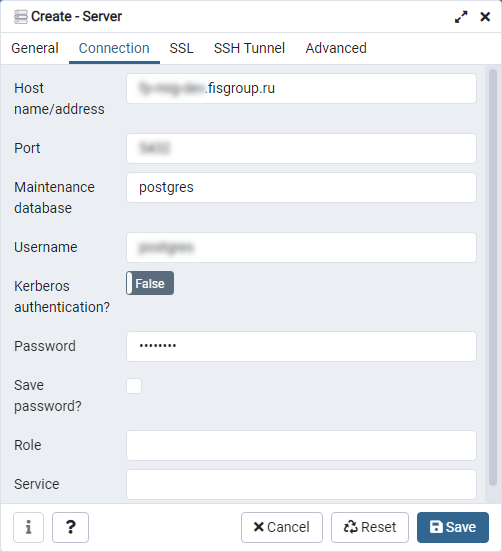
После чего будет открыто окно для настройки подключения к серверу БД.

3. На вкладке **General**:



В поле **Name** указать **любое название подключения**, обычно указывается адрес стенда.

4. На вкладке **Connection**:



* В поле **Host name / address** указать адрес сервера, к которому выполняется подключение, а в поле **Port** — порт.
* В полях **Username** и **Password** ввести логин и пароль от учетной записи, под которой будет проводится подключение.

Затем необходимо создать подключение, нажав на кнопку **Save**.

В левой панели должно появится подключение с указанным именем, при раскрытии его pgAdmin выполнит подключение к БД.

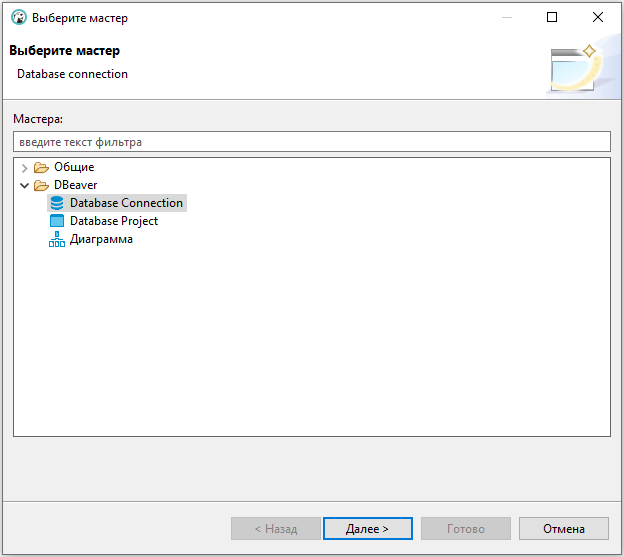
**3. Подключение к Oracle**

Для подключения к Oracle необходимо:

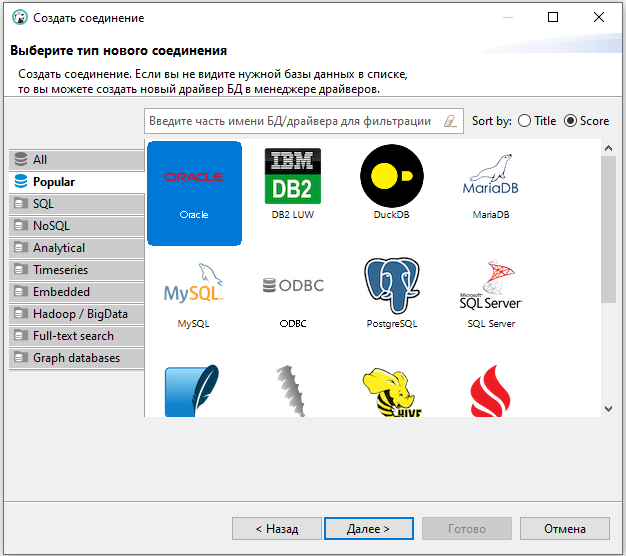
1. Открыть приложение **DBeaver**.

2. В верхней панели выбрать пункт **Файл → Новый**.

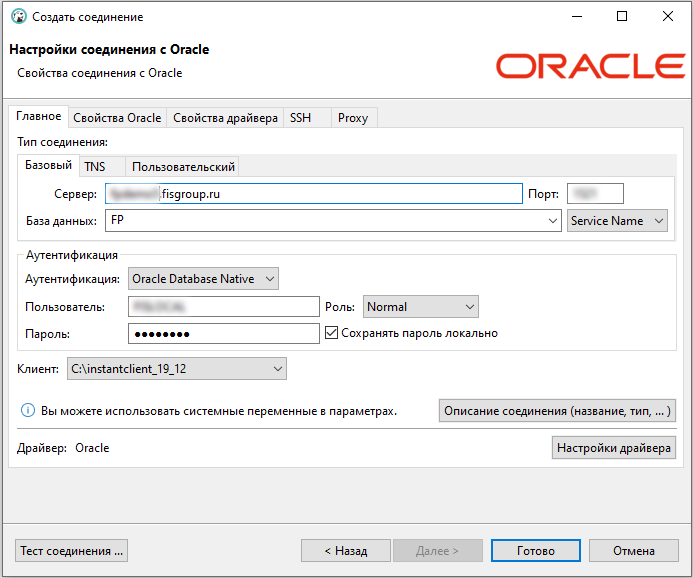
3. В появившемся всплывающем окне выбрать мастер **Database Connection** и нажать на **Далее**.



4. На шаге **Выберите тип соединения** необходимо выбрать **Oracle**и нажать на **Далее**.



5. На следующем шаге необходимо настроить соединение:



* В поле **Сервер** указать адрес базы данных.
* В поле **Порт** ввести порт для подключения.
* В поле **Клиент** проверить, что указан правильный путь до Oracle Instant Client.
* В поле **База данных** указать название базы данных.
* В поле **Пользователь** ввести логин учетной записи, под которой будет проводится подключение.
* В поле **Пароль** ввести пароль от учетной записи.

Далее нажать на кнопку **Готово**.

**Внимание:** От указанного пользователя зависит, к какому именно типу БД (БД локальных данных или БД платформенных метаданных) будет выполняться подключение.

**Дополнительно:** Можно проверить соединение, нажав на кнопку **Тест соединения**. Если во всплывающем окне будет указано, что было успешно соединено с БД, то данные введены корректно.

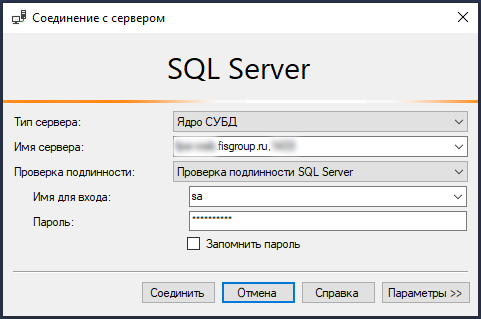
В левой панели должно появится подключение с указанным именем, при раскрытии его DBeaver выполнит подключение к БД.

**4. Подключение к Microsoft SQL Server**

Для подключения к Microsoft SQL Server необходимо:

1. Открыть приложение **Microsoft SQL Server Management Studio**.

2. В появившемся всплывающем окне:



* Указать в **Имя сервера** адрес БД, а потом через запятую указать порт для подключения к БД.
* В выпадающем списке **Проверка для подлинности** выбрать **Проверка подлинности SQL Serve**r.
* В **Имя для входа** ввести логин от учетной записи, а в **Пароль** ввести пароль от учетной записи.

После чего нажать на кнопку **Соединить**, чтобы подключиться к БД. В левой панели должно появится подключение к указанному серверу.

**Дополнительная информация**

Краткое руководство по работе с запросами в MS SQL Management Studio: <https://docs.microsoft.com/ru-ru/sql/ssms/quickstarts/ssms-connect-query-sql-server?view=sql-server-ver15>

Краткое руководство по работе с запросами в pgAdmin: <https://www.pgadmin.org/docs/pgadmin4/development/query_tool.html>

Краткое руководство по работе с запросами в DBeaver: <https://dbeaver.com/docs/wiki/SQL-Editor/>

**Физическое проектирование модели данных**

В данном уроке рассмотрено создание приложения на платформе. **По желанию** можно повторить все шаги по уроку. В рамках этого курса, чуть позже, обязательным для создания будет похожее приложение, но с другими настройками.

В рамках данного урока рассмотрим основы **физического проектирования модели данных** в Case Platform на примере небольшого приложения. Для создания модели данных будем использовать две категории объектов — [перечисление](https://doc.caseplatform.tech/#0.0.0:0:0:user_guide/classifier_builder.html:0:ru) и [тип](https://doc.caseplatform.tech/#0.0.0:0:0:user_guide/type_builder.html:0:ru), поэтому рекомендуется изучить их описание в документации перед прохождением курса.

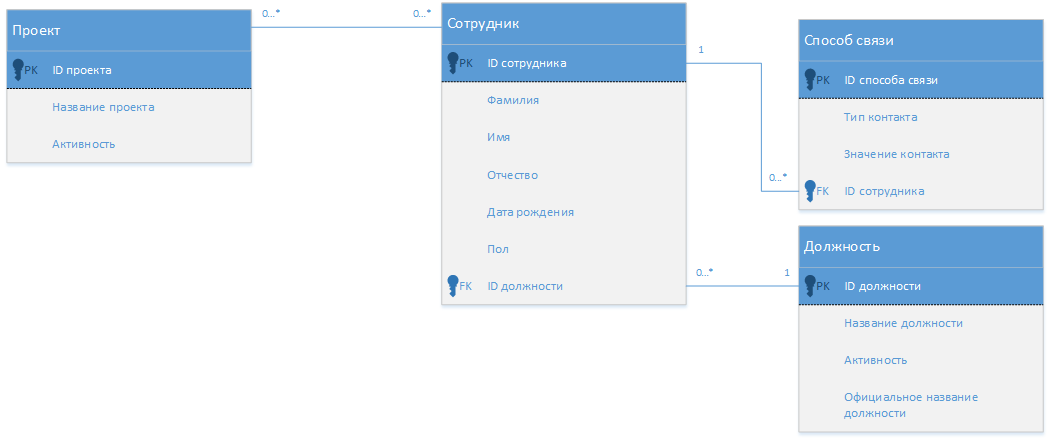
## ****Ключевые моменты:****

* Перечисления рекомендуется использовать для статических и простых справочных сущностей;
* Справочные типы — для справочных сущностей с более комплексным атрибутивным составом;
* Для **мягкого удаления** записей рекомендуется использовать логические атрибуты;
* Коллекции или отдельные связующие типы необходимо применять только при наличии связи “многие-ко-многим” между типами.

## ****1. Краткое описание приложения****

В рамках данного урока мы настроим **систему учета кадров в IT-компании**: система должна будет фиксировать список сотрудников, работающих в компании, их должность, а также проекты, в которых они работают.

На рисунке ниже представлена схема данных разрабатываемого приложения.

Схема данных системы учета кадров в IT-компании

## ****2. Создание приложения****

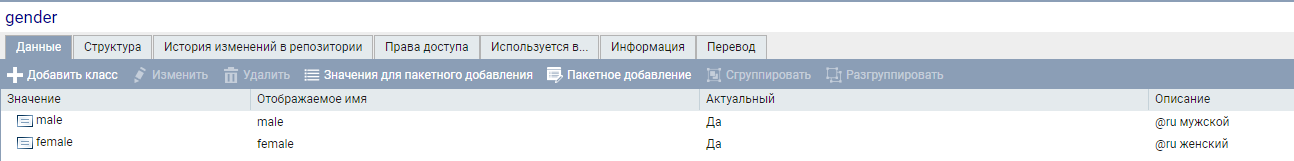
Создадим приложение под названием **HR\_demo**, и в модуле **Main** (Основной) создадим 2 пакета — **data** и **dictionaries**. Пакет **data** будет предназначен для хранения оперативных данных, а пакет **dictionaries**— для хранения справочных данных.

Начнем с создания сущностей для справочных данных.

### ****2.1. Настройка перечислений****

Перечисления рекомендуется использовать для создания статических справочных сущностей, т.е. тех сущностей, чьи значения не изменяются в процессе работы приложения. Эквивалентом перечислений во множестве языков программирования являются Enumeration (enum) в Java, C#, TypeScript и т.д.

В рамках разрабатываемого приложения мы создадим перечисления для пола сотрудника и типа контакта у способа связи. В пакете **dictionaries** для пола сотрудника мы создадим перечисление **gender**, в который будут добавлены 2 записи – мужской и женский: male, female. Созданное перечисление представлено на рисунке ниже

Перечисление gender для хранения пола сотрудника

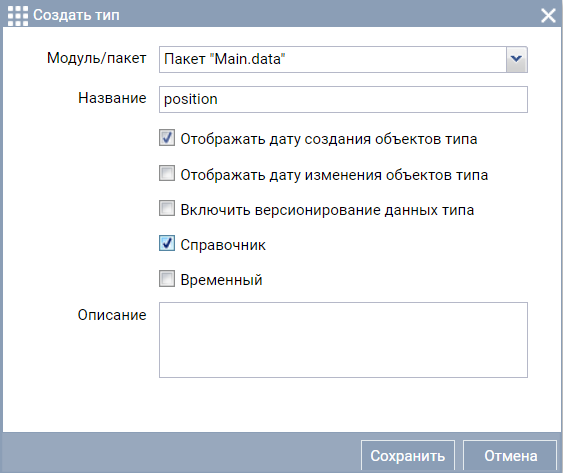
Для типа контакта мы также создадим перечисление **contact\_type**, в которое будут добавлены две записи – телефон и электронная почта: phone и email.

### ****2.2. Настройка справочных типов****

Типы также можно использовать в качестве хранения справочных сущностей. Их рекомендуется использовать в случае если:

* справочная сущность обладает более комплексной структурой — например, сущность имеет более трех "классических" атрибутов, которые есть в перечислениях: значение, отображаемое имя и актуальность;
* справочная сущность является динамической, т.е. её значения могут быть изменены в процессе работы приложения (однако при этом стоит учесть особенность создания типа в Case Platform).

В рамках разрабатываемого приложения справочный тип будет создан для сущности **должность сотрудника** (position), т.к. она обладает более комплексной структурой, чем способно отразить перечисление. При создании справочного типа необходимо указать, что он является справочником, установив флаг **Справочник** во всплывающем окне создания типа, как указано на рисунке ниже:



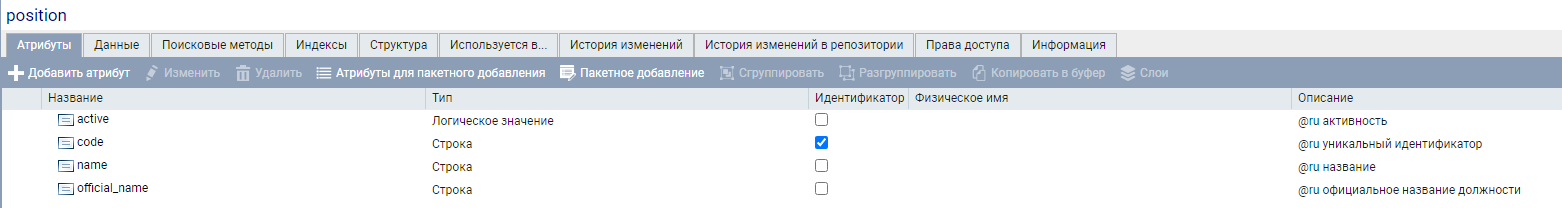
В этот тип добавим следующие атрибуты:

* признак активности должности – **active** – данный признак нам необходим для **мягкого удаления** записи, этот концепт проектирования будет рассмотрен ниже;
* название должности – **name**;
* уникальный идентификатор должности – **code**;
* официальное название должности – **official\_name**.

Отдельный атрибут для хранения ID **во все типы** добавлять не нужно, т.к. для каждого типа автоматически создается столбец “Номер объекта”, который будет выступать первичным ключом таблицы. Подробно этот столбец будет рассмотрен в следующих уроках на примере SQL-запросов.

**Мягкое удаление** записи (**soft deletion**) — паттерн проектирования модели данных, при котором в сущность добавляется признак, отвечающий за используемость записи — например, актуальность или активность. В случае если у записи не отмечен этот признак, то в приложении она не будет отображаться и использоваться. Этот флаг позволяет нам не удалять запись из таблицы (например, при помощи UDML-функции [**УДАЛИТЬ**() / **DELETE**()](https://doc.caseplatform.tech/#0.0.0:0:0:reference_guide/udml_functions/system_functions/DELETE.html:0:ru)), тем самым упрощая ее восстановление при необходимости. Подробнее про него можно прочитать в статье “[To Delete or to Soft Delete, That Is the Question!](https://dzone.com/articles/to-delete-or-to-soft-delete-that-is-the-question)”.

У атрибута **code** мы также установим флаг под названием **Атрибут является идентификатором**. Добавленные атрибуты представлены ниже:

Структура типа для должности

#### 

#### **Для чего в справочных типах нужен признак Справочник и атрибут с флагом Является идентификатором?**

Установка флагов **Справочник** в типах и **Идентификатор** в атрибутах необходима для повышения удобства и оперативности установки обновлений.

В рамках доработки приложения часто возникает необходимость изменить значения справочников (например, добавить новую запись, изменить значение атрибута у другой записи и т.д.). Чтобы автоматически переносить изменения этих справочников, в Case Platform реализована функциональность экспорта и импорта данных. При экспорте данных Case Platform может выгрузить данные всех типов, которые отмечены как справочники. А при импорте справочников Case Platform может произвести слияние данных (merge) на основе значения атрибута, который отмечен как идентификатор.

**Внимание: если у типа установлен флаг Справочник, но при этом нет атрибута, отмеченного как идентификатор, Case Platform не сможет корректно произвести слияние и импорт данных не выполнится.**

Кроме того, отдельный атрибут для хранения уникального идентификатора записи необходим для корректной идентификации записи в бизнес-логике приложения. Например, в рамках данного приложения в зависимости от должности сотрудника будет определяться видимость полей в карточке сотрудника.

Если в приложении для реализации бизнес-логики будет использован только ID объекта (т.е. результат функции [**ИДОБЪЕКТА**() / **OBJECTID**()](https://doc.caseplatform.tech/#0.0.0:0:0:reference_guide/udml_functions/system_functions/OBJECTID.html:0:ru)), то появляется вероятность возникновения ошибки, связанной с некорректной идентификацией записи. Это связано с тем, что ID записям выдаются автоматически при помощи последовательностей в БД (sequence), и при переносе данных из приложения на разработческой среде в приложения в продуктивных и тестовых средах этот ID записи может быть другим.

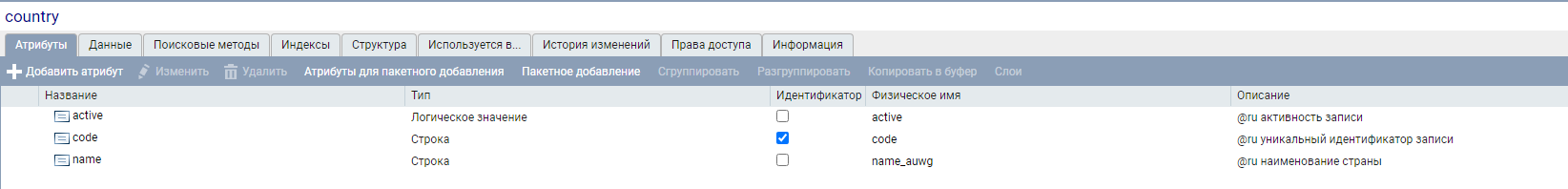
**Примечание:** если записи в типе полностью заполняются пользователем приложения (например, через специально созданную форму пользователь ведет справочник схем кредитования), то устанавливать флаг “Справочник” у такого типа не рекомендуется, т.к. в разных средах записи в справочниках могут быть совершенно разными, и при переносе данных записи из целевой среды могут быть утеряны.

**Важно #1:** не рекомендуется использовать ссылки на **несправочные (обычные) типы** в справочных типах. В этом случае при экспорте / импорте справочных данных значения этих ссылочных атрибутов не будут перенесены, и они останутся пустыми.

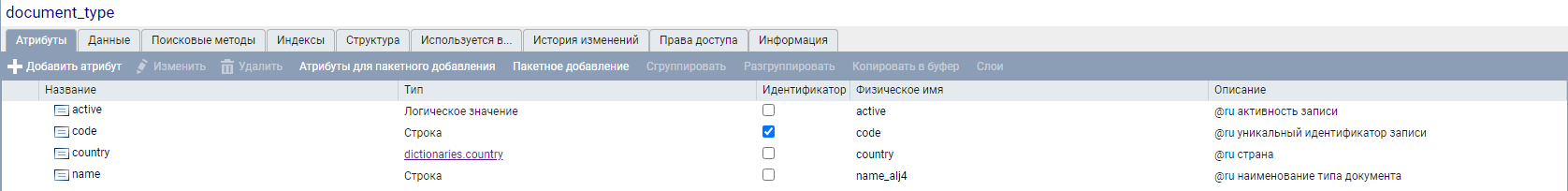
**Важно #2:** если в справочном типе используются ссылки на другие справочные типы, то при экспорте в список выгружаемых данных необходимо включить не только целевой справочник, но и справочники, на которые он ссылается.

**Эксперимент.** Рассмотрим этот пункт подробнее на примере: в отдельном приложении мы создадим модуль **dictionaries.** В этом модуле мы создадим 2 справочных типа - **справочник стран** (country) и **справочник типов документов** (document\_type).

Структура типа **country** представлена ниже:

Структура типа справочника стран

Структура типа **document\_type** представлена ниже, как можно увидеть в этом типе задействована ссылка на справочник стран:

Структура типа справочника типов документов

Далее мы заполним данными этими типы. В **country**добавим 2 записи:

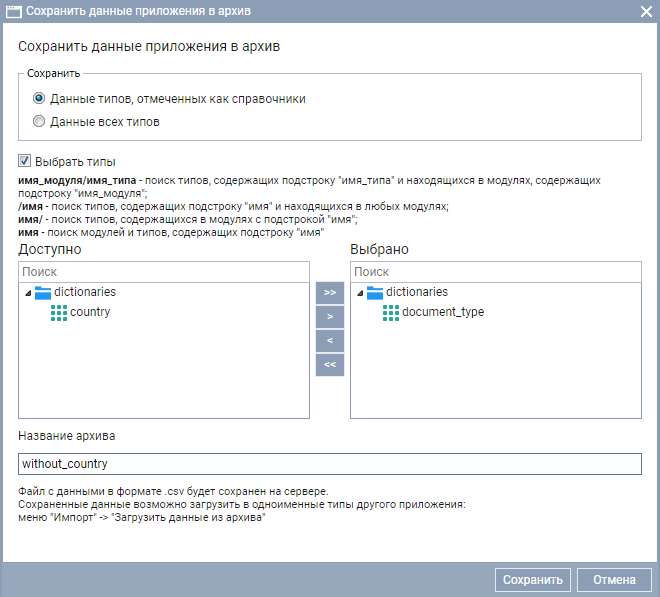
| **name** | **code** | **active** |
| --- | --- | --- |
| Country 1 | 1 | Да |
| Country 2 | 2 | Да |

В **document\_type**также добавим 2 записи:

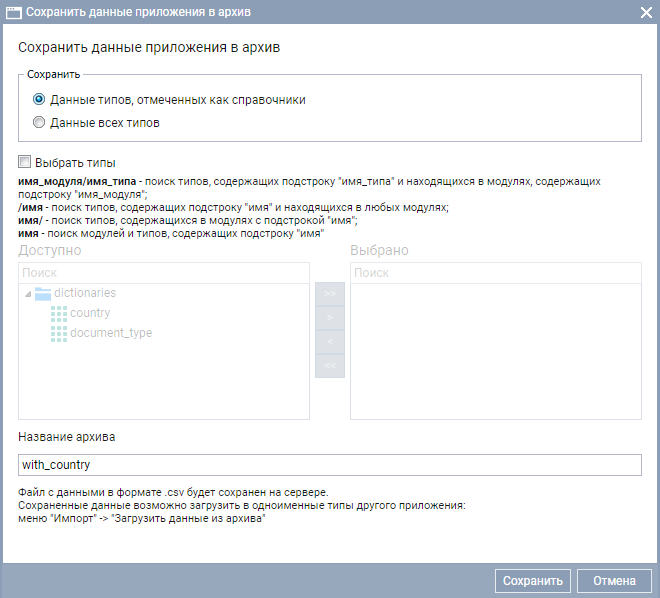
|  |  |  |  |
| --- | --- | --- | --- |
| **name** | **code** | **country** | **active** |
| Document Type 1 | 1 | Country 1 | Да |
| Document Type 2 | 2 | Country 2 | Да |

После чего мы проведем экспорт справочника **document\_type**, выбрав в верхней панели пункт меню **Экспорт** → **Сохранить данные в архив**.

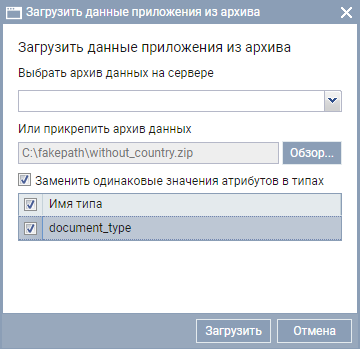
Сначала мы проведем экспорт данных без справочного типа **country**:



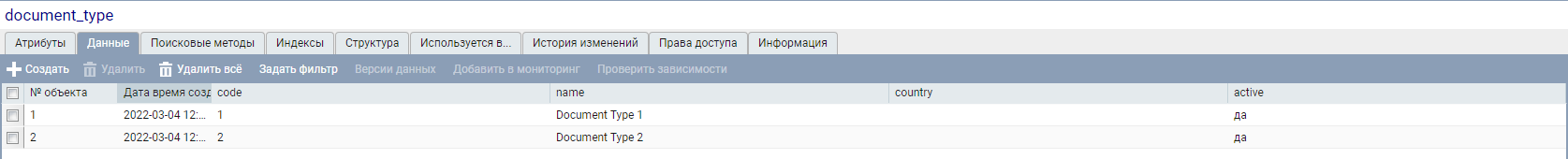
А затем проведем экспорт всех справочников:



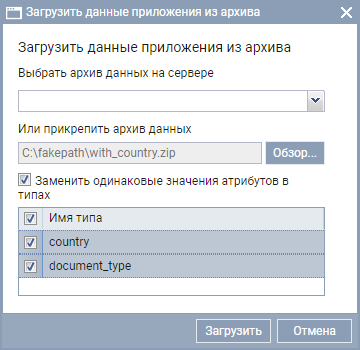
Далее мы проведем импорт данных, выбрав в верхней панели пункт меню **Импорт** → **Загрузить данные из архива**. Сначала мы импортируем первый вариант выгрузки данных без справочника стран в соответствии с параметрами указанными ниже:



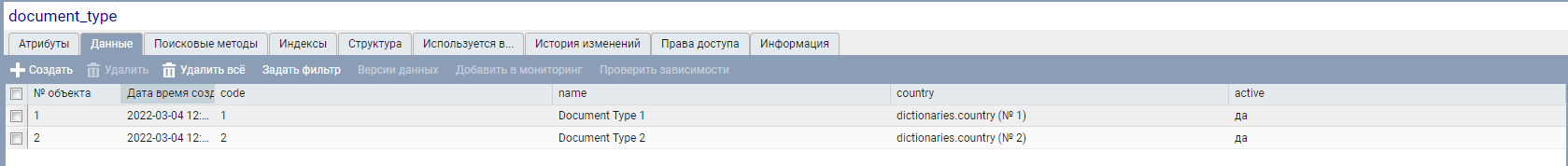
После импорта данных мы можем убедиться, что в справочнике типов документов атрибут, ссылающийся на справочник стран, стал пустым:



Далее мы проведем повторно импорт данных, но в этот раз мы используем вариант выгрузки данных со справочником стран:



После импорта этого варианта выгрузки данных мы можем убедиться, что в этот раз атрибут, ссылающийся на справочник стран, теперь корректно заполнен:

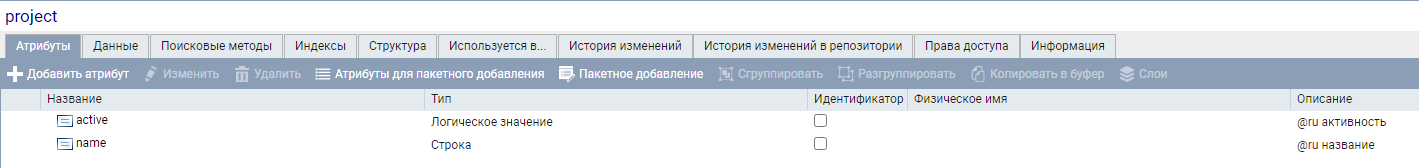


### 

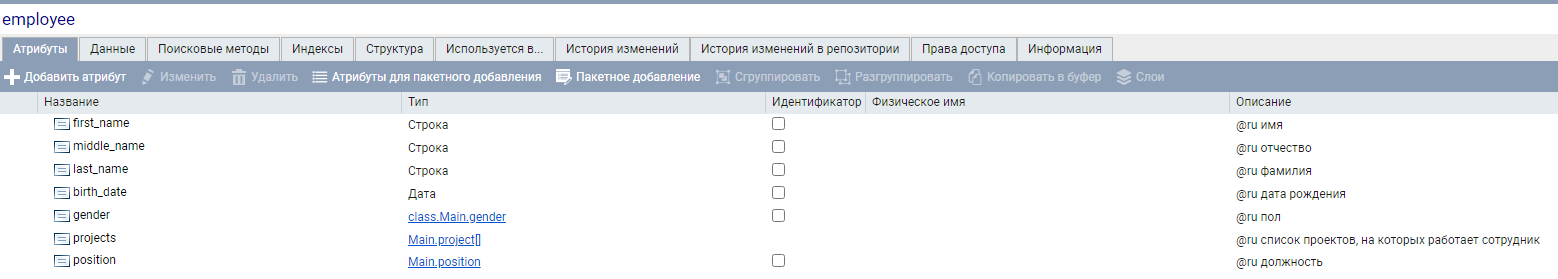
### ****2.3. Настройка оперативных типов****

Для хранения оперативных сущностей необходимо использовать несправочные типы. В рамках разрабатываемого приложения мы создадим их для следующих сущностей: **сотрудник, проект и способ связи.** Типы мы будем создавать в пакете под названием **data**.

Мы начнем с типа для хранения проектов – **project**, в который добавим 2 атрибута – это **наименование проекта** и **признак активности проекта**, который также нужен для мягкого удаления записи.

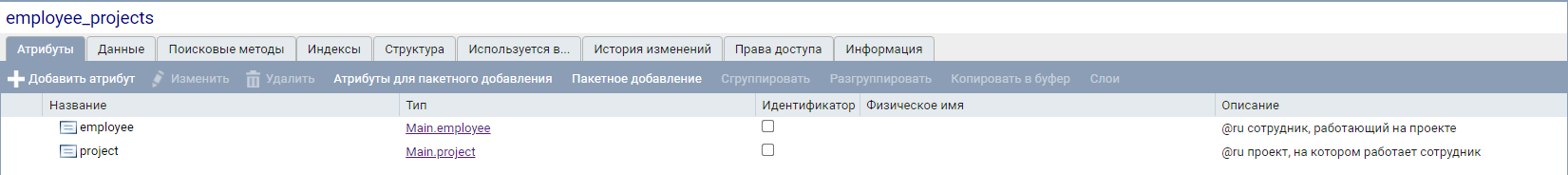
Структура типа для хранения проектов

Затем создадим тип для хранения сотрудников – **employee**, в который мы добавим атрибуты для хранения ФИО, даты рождения, пола, должности сотрудника, а также список проектов, в которых он работает.

Структура типа для хранения сотрудников

В связи с тем, что сотрудник может участвовать в нескольких проектах одновременно и что в одном проекте могут работать несколько сотрудников (т.е. связь “многие-ко-многим”), мы укажем у атрибута **projects**флаг **Атрибут является коллекцией**.

Альтернативным вариантом реализации связи “многие-ко-многим” является **создание нового связующего типа**, который будет хранить 2 атрибута, один из которых будет ссылаться на **employee**, а другой на **project**.

Пример связующего типа

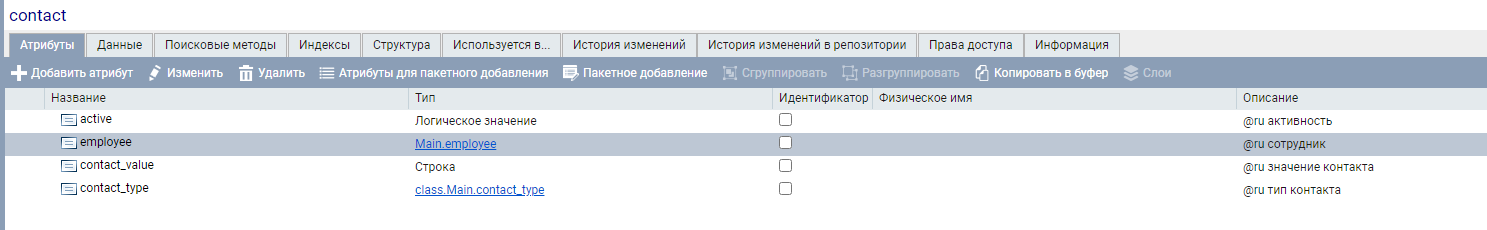
Эти 2 варианта имеют следующие отличительные черты:

| **Расхождение** | **Коллекционный атрибут** | **Связующий тип** |
| --- | --- | --- |
| Получение записей | Для получения записей можно обратиться к коллекционному атрибуту | Для получения записей необходимо создать отдельный поисковый метод |
| Удаление связи | Для удаления связи Case Platform физически удаляет связь из таблицы | Для удаления связи можно физически удалить запись из типа либо добавить новый атрибут для мягкого удаления связи |
| Возможности расширения | Отсутствуют | Можно добавлять неограниченное количество новых атрибутов |

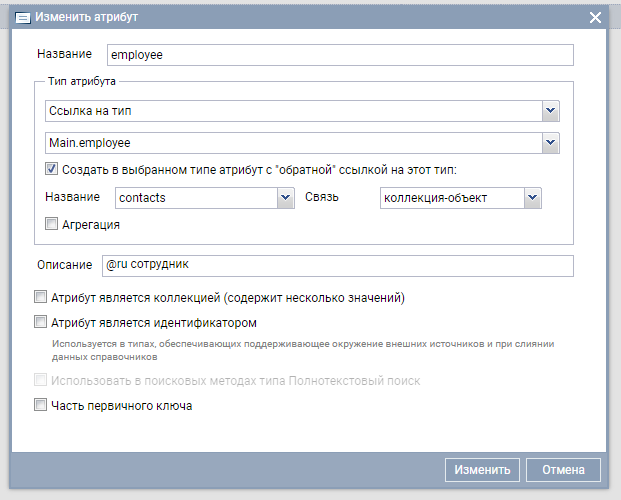
В рамках данного урока мы воспользуемся **коллекционным** атрибутом **projects** в типе **employee,** т.к. нам не нужны расширенные возможности связующего типа.

Особенности атрибута типа “коллекция” будут подробнее рассмотрены в следующих уроках.

И в заключение создадим последний тип для хранения способов связи с сотрудником – **contact**, в который добавим атрибуты: значение контакта, тип контакта, активность контакта и ссылку на сотрудника, т.к. связь между контактом сотрудника – “один-ко-многим”.

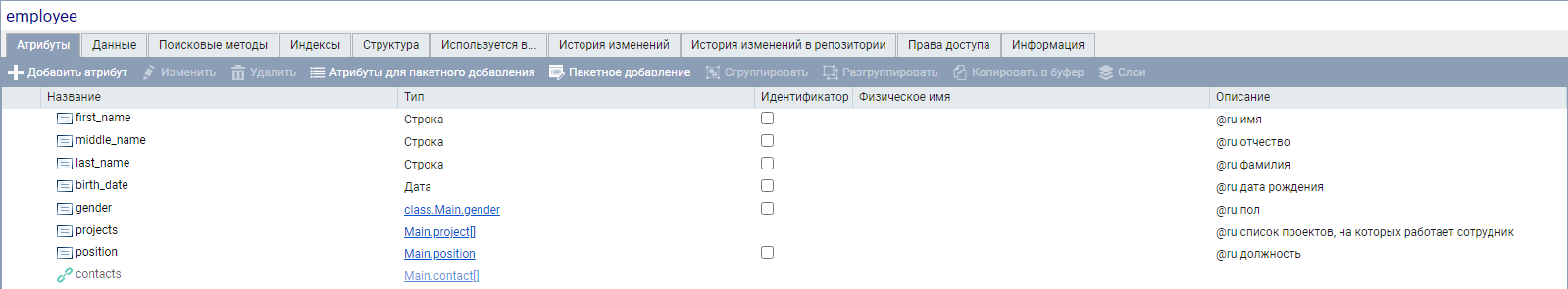
Структура типа "contact"

При добавлении ссылки на сотрудника мы также установим флаг **Создать в выбранном типе атрибут с "обратной" ссылкой на этот тип**, чтобы в тип **employee** был добавлен ассоциативный атрибут. Создание ассоциативного атрибута позволит упростить поиск и просмотр контактов в Конструкторе, связанных с этим сотрудником.



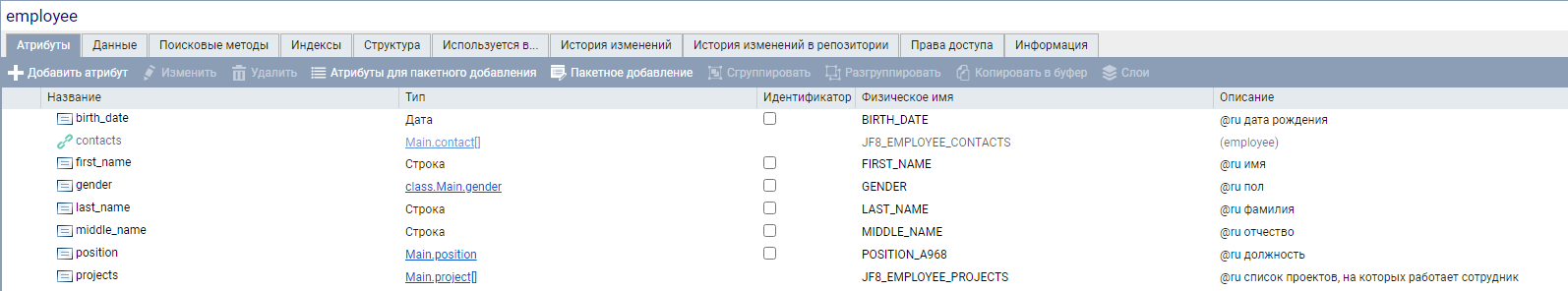
**Внимание:** данный атрибут нужен только для упрощения поиска данных в Конструкторе в процессе отладки приложения.  Использовать ассоциативные атрибуты непосредственно в работе приложения, т.е. обращаться к ним в коде, не рекомендуется: при обращении к ассоциативному атрибуту генерируется SQL-запрос для получения данных следующего формата: **select \* from table where column = :1 order by ID**. По этой причине в работе приложения лучше задействовать поисковые методы, т.к. их можно настроить самостоятельно.

После добавления этого атрибута мы можем увидеть, что в типе **employee** появился соответствующий атрибут.

Структура типа employee

Следующий шаг — **перезапуск приложения**, который необходим для синхронизации логической модели данных с физической моделью данных. В рамках перезапуска Case Platform создаст в базе локальных данных новые таблицы и присвоит всем типам и их атрибутам физические имена, т.е. названия таблиц / столбцов, отвечающих за хранение данных.

После успешного перезапуска и обновления страницы можно увидеть, что столбец **Физическое имя** в типе теперь заполнен данными.

Структура типа employee с выданными физическими именами

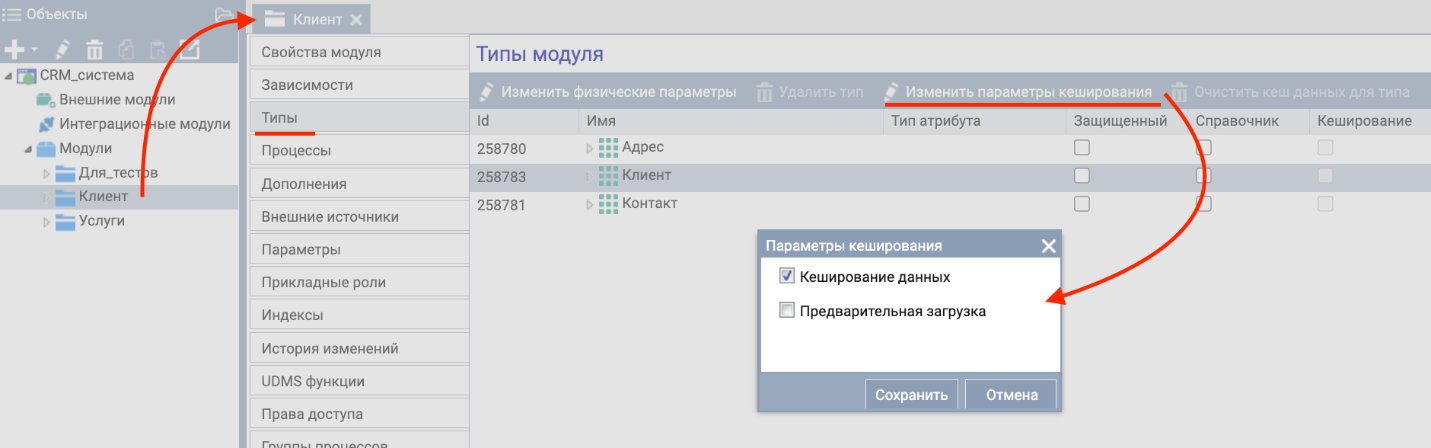
**Настройки типа — кеширование**

В целях повышения оперативности работы приложения рекомендуется у **справочных** типов включать кеширование.

При включенном кешировании записи типа загружаются в оперативную память, и поисковые методы Конструктора и Case Platform для получения этих записей не обращаются к базе данных, а получают нужную информацию из кеша.

Для включения кеширования необходимо:

1. Перейти во вкладку **Типы** у модуля.
2. Выбрать нужный Тип.
3. Нажать на кнопку **Изменить параметры кеширования**.
4. Установить флаг **Кеширование данных**.
5. Перезапустить приложение.



Флаг **Предварительная загрузка** отвечает за повторное заполнение кеша после очистки. Данный флаг рекомендуется ставить в случае, если записи из типа могут часто изменяться (например, это справочник, который редактируется пользователями в приложении). Кеш будет всегда повторно заполняться после перезагрузки приложения.

❗️ Кеш общий на все потоки и разбит на части по UDM-типам. Конкурентный доступ обеспечивается за счет блокировок по каждому из типов.

❗️ Если в рамках потока №1 произошло изменение/добавление/удаление записи в справочнике, а после этого в рамках потока №2 было обращение к этому справочнику, то изменения из потока №1 будут видны потоку №2. Это происходит за счет автоматического изменения кеша после каждой успешной транзакции в БД, если в рамках этой транзакции были внесены изменения в кешируемый тип-справочник.

Более того, если в рамках транзакции добавлено, изменено или удалено 42 записи, то по успешному окончанию транзакции только эти 42 записи будут добавлены/изменены/удалены из имеющихся 200 записей в типе-справочнике (42 и 200 это рандомные числа для примера).

❗️ Для работы с кешем данных типа есть функция [CLEARDATACACHE()](https://doc.caseplatform.tech/#0:0:0:reference_guide/udml_functions/system_functions/CLEARDATACACHE.html:0:en), которая очищает кеш данных типа. Если у указанного типа включено свойство **Предварительная загрузка**, то после очистки кеша, функция снова заполнит кеш данными.

**Физическое хранение данных приложения в БД**

Продолжим рассматривать приложение **HR\_demo**, созданное в предыдущих уроках. В рамках теоретической части подключаться к БД и выполнять запросы самостоятельно не требуется.

В рамках данного урока мы подробно рассмотрим то, как платформа физически хранит данные в БД. Эта информация нам пригодится при написании SQL поисковых методов, а также поможет в дальнейшем при проектировании модели данных приложений.

**1. Заполнение данных приложения**

Ранее созданное приложение мы наполним информацией. Базовые шаги по добавлению, редактированию и удалению записей в типах рассмотрены в [документации](https://doc.caseplatform.tech/#0.0.0:0:0:user_guide/type_builder/type_data/type_objects_operations.html:0:ru).

* Начнем с заполнения **справочника должностей**:

| **Название** | **Код** | **Официальное название должности** | **Активность** |
| --- | --- | --- | --- |
| Разработчик | developer | Программист | Да |
| Тестировщик | qa | Специалист по тестированию в области ИТ | Да |
| Менеджер проектов | pm | Управляющий проектом | Да |

* Затем создадим **проекты**:

| **Название** | **Активность** |
| --- | --- |
| Проект 1 | Да |
| Проект 2 | Да |

* После чего создадим **сотрудников**:

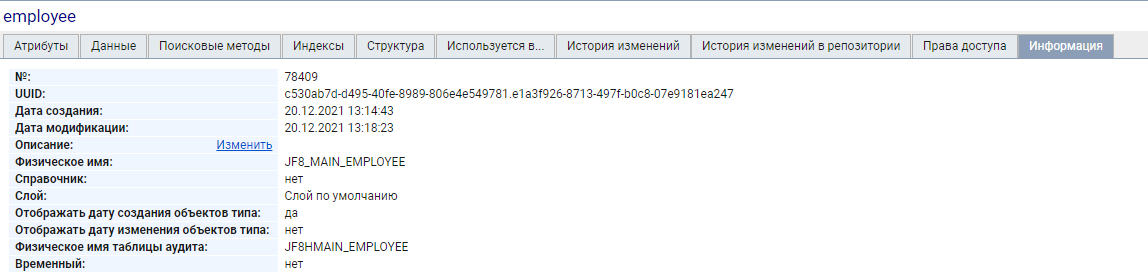
| **Имя** | **Отчество** | **Фамилия** | **Дата рождения** | **Пол** | **Проекты** | **Должность** |
| --- | --- | --- | --- | --- | --- | --- |
| Иван | Иванович | Иванов | 1990-01-01 | Мужской | Проект 1, Проект 2 | Разработчик |
| Ольга | Ивановна | Иванова | 1990-02-01 | Женский | Проект 2 | Разработчик |

* И затем создадим **контакты**:

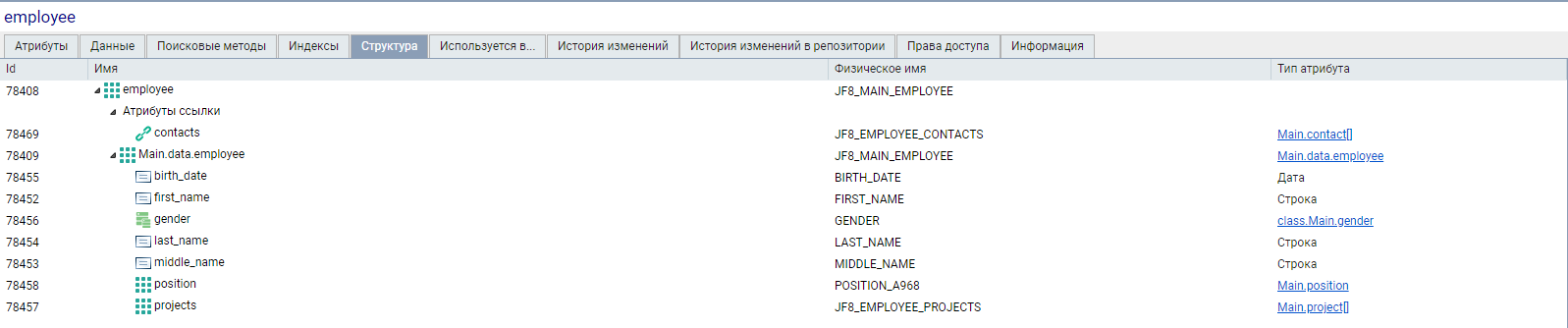
| **Активность** | **Тип контакта** | **Значение контакта** | **Сотрудник** |
| --- | --- | --- | --- |
| Да | Телефон | +7-900-123-45-67 | Иван Иванович Иванов |
| Да | Электронная почта | test@mail.com | Ольга Ивановна Иванова |

**2. Просмотр данных из таблицы при помощи клиента работы с БД**

Далее подготовим запрос, который получит данные из типа **employee**. Чтобы узнать, к какой именно таблице необходимо обратиться, откроем вкладку **Информация**. На данной вкладке нас интересует свойство под названием **Физическое имя**, которое хранит название соответствующей таблицы в базе данных.

Вкладка Информация в типе

Для получения более подробной информации о столбцах таблицы необходимо переключиться на вкладку **Структура**, либо на вкладку **Атрибуты**. В столбце **Физическое имя** будут представлены соответствующее атрибуту название столбца в таблице.

Вкладка Структура в типе

После чего подключимся к БД локальных данных, подготовим и выполним запрос на получение всех столбцов из таблицы для employee:

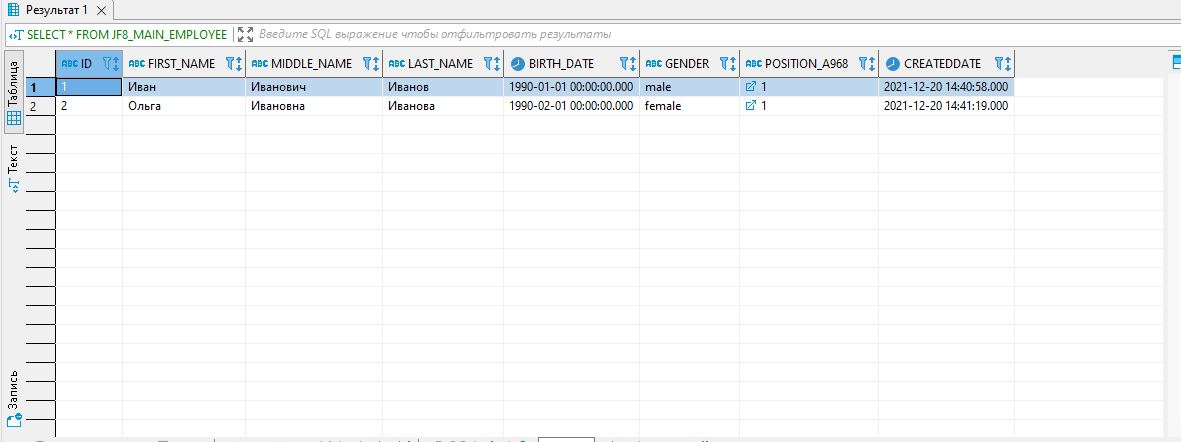
SELECT

\*

FROM

JF8\_MAIN\_EMPLOYEE

После его выполнения можно увидеть, что в списке выходных столбцов присутствуют физические имена атрибутов из вкладки **Структура** — *first\_name, middle\_name, last\_name, birth\_date, gender и position.*

Результаты запроса

Кроме столбцов, соответствующих созданным пользователем атрибутам Типа, Case Platform автоматически создает следующие столбцы:

* **ID** – уникальный идентификатор записи, соответствующий значению поля № объекта при просмотре данных Типа;
* **createddate** – дата и время создания записи. Столбец будет создан, если у Типа установлен флаг **Отображать дату создания объектов типа**;
* **modificationdate** – дата и время обновления записи. Столбец будет создан, если у Типа установлен флаг **Отображать дату изменения объектов типа**.

**3. Особенности хранения локальных данных**

Выделим основные особенности хранения данных в локальной БД:

1. ID (номер объекта, первичный ключ) – это **строковый**, а не числовой тип данных;
2. Длина строковыхатрибутов, не имеющих свойства **Длинный текст**, не может быть больше 2000 символов;
3. В отличие от типов и коллекционных атрибутов, Перечисления не хранятся в локальной базе данных как отдельные таблицы — вместо этого они хранятся в таблицах **classifierdef** и **classifieritem** в базе метаданных Case Platform (fisplatform);
4. Атрибуты, ссылающиеся на Перечисления, хранятся в таблицах как строковые значения классов, поэтому для получения их значений не нужно использовать присоединение таблиц;
5. Атрибуты, ссылающиеся на типы, хранятся в таблицах как **строковые внешние ключи записей этих Типов**;
6. Коллекционные атрибуты хранятся как **отдельные таблицы**;
7. Типы данных столбцов могут отличаться в зависимости от используемой базы данных.

Рассмотрим подробнее **пункт 5.** Для связи между таблицами используются **внешние ключи**: в **дочерней** **таблице** (т.е. в таблице, которая содержит внешний ключ) хранится **строковый атрибут**, который связан с **первичным ключом родительской таблицы** (т.е. таблицы, на которую ссылается этот атрибут).

На результатах запроса, приложенных ранее, можно увидеть, что сотрудник с ID = 1 ссылается на должность с ID = 1. Чтобы посмотреть более подробную информацию о должности сотрудника можно использовать JOIN, в условии которого проверяется связь между столбцом **position** в **employee** и столбцом **id** в таблице **position**.

SELECT

e.id,

e.FIRST\_NAME ,

e.MIDDLE\_NAME ,

e.LAST\_NAME ,

p.NAME\_AC17

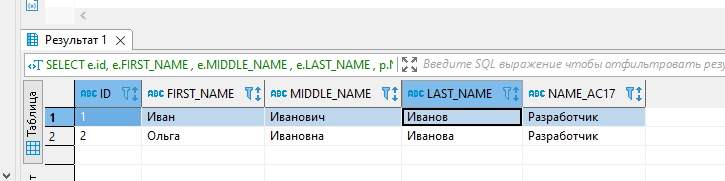
FROM

JF8\_MAIN\_EMPLOYEE e

INNER JOIN JF8\_MAIN\_POSITION p ON

p.id = e.POSITION\_A968

Результаты запроса представлены ниже:

Результаты запроса

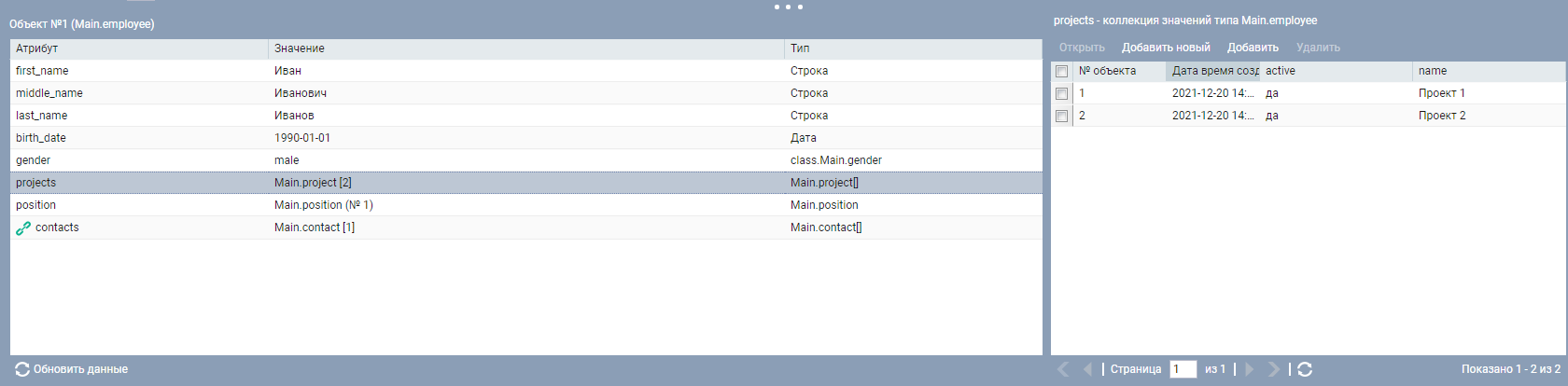
Соответственно, для обеспечения консистентности данных в таблицах используется ограничения внешнего ключа: из таблицы нельзя удалять записи, если на неё есть ссылка; нельзя создавать записи в дочерней таблице, которые ссылаются на несуществующую запись и т.д.

Рассмотрим подробнее **пункт 6.** Для реализации коллекционных атрибутов Case Platform создает сопоставительные таблицы, каждая из которых состоит из трех столбцов:

* ID записи сопоставительной таблицы, **физическое имя** —**ID**;
* ID записи из Типа, в которую добавлен коллекционный атрибут, **физическое имя** —**host\_mtype**;
* ID записи, на которую ссылается коллекционный атрибут, или же просто значение (если это коллекция простого типа, тип данных будет отличаться в зависимости от типа простых данных). Если это коллекционный атрибут ссылается на Тип, то **физическое имя** — **value\_mtype**, иначе — **value**.

Точное название таблицы, которое хранит коллекцию, можно также увидеть во вкладке **Структура**, у коллекционного атрибута физическое имя будет отличаться от имен простых атрибутов — оно похоже на физическое имя типа.

Приведем пример — в разработанном приложении указано, что сотрудник Иванов работает в 2ух проектах: Проект 1 и Проект 2.

Значения коллекционного атрибута

Чтобы получить названия проектов, в которых работает сотрудник, к таблице **employee** присоединим сопоставительную таблицу при помощи столбца **host\_mtype**, после чего к этой сопоставительной таблице присоединим таблицу **projects** при помощи столбца **value\_mtype**.

SELECT

e.id,

e.FIRST\_NAME ,

e.MIDDLE\_NAME ,

e.LAST\_NAME ,

p.NAME\_AMZP

FROM

JF8\_MAIN\_EMPLOYEE e

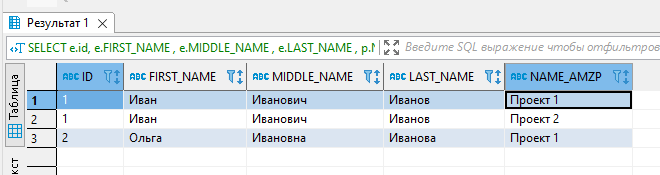
INNER JOIN JF8\_EMPLOYEE\_PROJECTS ep ON

e.id = ep.host\_mtype

INNER JOIN JF8\_MAIN\_PROJECT p ON

ep.value\_mtype = p.id

Результаты запроса представлены ниже:

Результаты запроса

Как было упомянуто в **пункте 7**, физические типы столбцовотличаются в зависимости от используемой СУБД, поэтому эту особенность необходимо учитывать при проектировании модели данных и написании запросов. Точные соответствия типов данных Case Platform и типов данных в СУБД представлены в таблице ниже:

| **Тип данных в Platform** | **Тип в Oracle** | **Тип в PostgreSQL** | **Тип в MS SQL** |
| --- | --- | --- | --- |
| Строка | varchar2(2000) | character varying (2000) | nvarchar(2000) |
| Строка (Длинный текст) | CLOB | text | text |
| Целое число | bigint | bigint | bigint |
| Дробное число | float | double precision | float |
| Период | bigint | bigint | bigint |
| Длительность | bigint | bigint | bigint |
| Дата | date | date | date |
| Дата и время | date (если версия Oracle 8) или  timestamp (если версия Oracle >= 9) | timestamp without timezone | datetime2(7) |
| Двоичные данные | blob | bytea | varbinary(max) |
| Логическое | number(1,0) | boolean | bit |

**Автоматическая или ручная выдача физических имен?**

**1. Различия**

В Case Platform реализована возможность ручной выдачи физических имен, которая позволяет:

* указать префикс, с которого будут начинаться физические имена таблиц;
* при создании типа указать его физическое имя;
* при создании атрибутов типа указывать их физические имена;
* указывать физические имена индексов.

Перед созданием приложения и проектированием модели данных необходимо ответить на вопрос — требуется ли ручная выдача физических имен?

Ручная выдача физических имен обычно нужна в следующих случаях:

* данные из приложения будут напрямую выгружаться для составления отчетов внешней системой, или внешняя система использует их в своей бизнес-логике;
* применение процедур БД в бизнес-логике приложения.

В этих случаях автоматическая выдача физических имен крайне **не рекомендуется**, т.к. есть большая вероятность того, что физические имена типов могут отличаться на разных стендах.

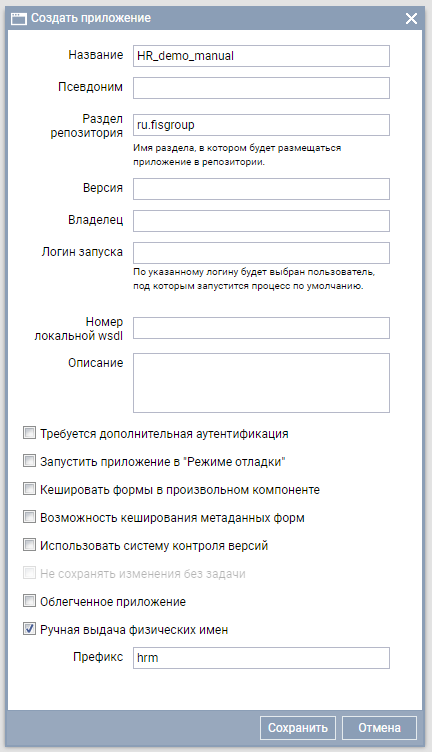
Кроме того, ручная выдача физических имен повышает оперативность написания SQL поисковых методов: если SQL-запрос написан и протестирован в отдельном клиенте работы с СУБД, то при автоматической выдаче физических имен этот SQL-запрос нужно будет перевести на UDM-имена.

**2. Проектирование модели данных с ручной выдачей физических имен**

Создание данного приложения **обязательно** к выполнению. Подробное описание того, что нужно сделать будет в следующем (практическом) уроке.

Создадим аналогичное приложение для учета кадров. Укажем у него названием **HR\_demo\_manual** и установим флаг **Ручная выдача физических имен**. В поле **Префикс** укажем строку из трех символов в нижнем регистре, которая должна отвечать следующим требованиям:

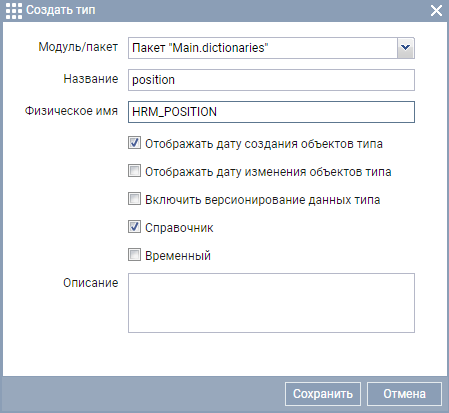
* начинаться с латинской буквы;
* содержать только латинские буквы и цифры



В модуле **Main** мы также создадим 2 пакета – **data**и **dictionaries**. В пакет **dictionaries** мы добавим те же 2 перечисления для **пола** (gender) и **типа контакта** (contact\_type). После чего добавим справочник для **должностей** (position). При включенной ручной выдаче физических имен во всплывающем окне добавления типа появляется новое текстовое поле – **Физическое имя**, в которое необходимо ввести имя таблицы, которая будет создана для этого типа.

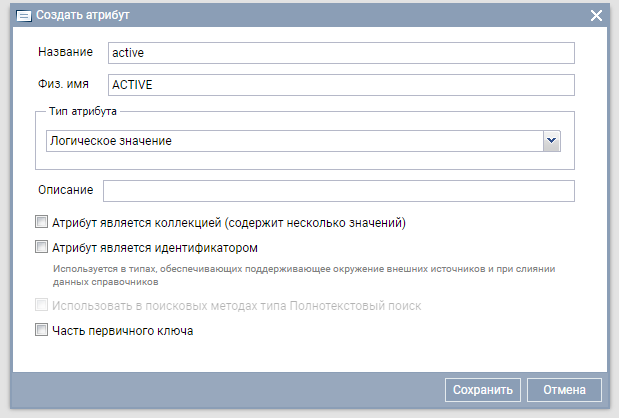
После ввода физического имени Case Platform автоматически добавляет к нему префикс и переводит его в верхний регистр. При указании физического имени необходимо также учитывать ограничения:

* имя может содержать только латинские буквы, цифры и знак подчёркивания,
* длина ограничена:
  + 31 символ с учетом префикса для MS SQL;
  + 30 символов с учетом префикса для Oracle или PostgreSQL.

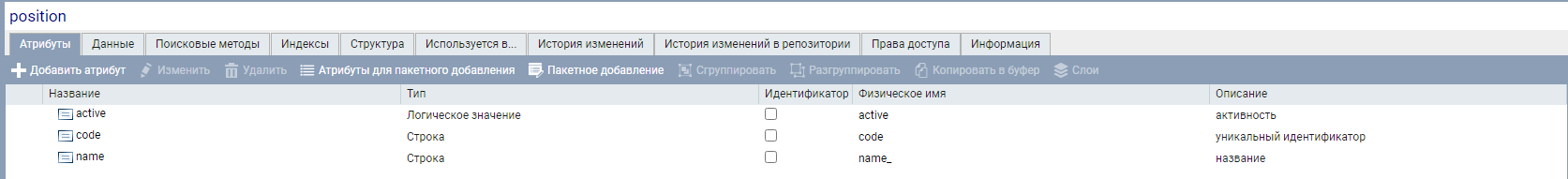


После создания типа добавим ему атрибуты. Во всплывающем окне добавления атрибута также появилось поле **Физическое имя**. При указании физического имени необходимо учитывать следующие ограничения:

* имя атрибута может содержать только латинские буквы, цифры и знак подчёркивания;
* длина ограничена:
  + для MS SQL имя атрибута не может быть длиннее 128 символов;
  + для Oracle — 30 символов;
  + для PostgreSQL — 63 символов.
* нельзя использовать зарезервированные SQL-слова (к материалу приложен список зарезервированных слов [**reserved\_words\_sql.txt**]).



Добавленные атрибуты представлены ниже:

Созданный тип для должности

Для **коллекционных атрибутов** необходимо вначале указать **префикс**, т.к. для них создается отдельная таблица, при этом необходимо учитывать ограничения наименований таблиц.

После создания остальных оперативных типов и инициализации приложения можно убедиться, что таблицы были корректно созданы в БД.

**[reserved\_sql\_words.txt](https://caseplatform.unicraft.org/api/v2/resource/40fd68e7-0839-46e7-816a-30bcab2372ad/any/0519f737-6c21-4051-b9b3-7a2deffe1af3.txt/eyJhbGciOiJFUzUxMiIsInR5cCI6IkpXVCJ9.eyJzaWQiOiIxOWNjYzExNS00ZjIwLTQyMmQtYmJmMi1hOTAyNmEzZmY0N2UiLCJpYXQiOjE3MDM0MzE1OTYsImV4cCI6MTczNDE4OTk5Nn0.AGaZ4Qm0L_q8-Ujx4JmgOKoPA38e9SD7fSWzUAbgnMI1A8mIJTwpnd0O9yd6pXe0guInMumaWrMiNE5Ta7JWRZcBAGdTlRt6YGTMYeeNbtaMnWyXUXDIb8_6pdePGxi8rpv_4rjVOzCT5Ys7HlJk_FdctxkbKMx_3rqQuGC7Cn75moN2" \t "_blank)**

**Общие особенности конструкторных и SQL поисковых методов**

**Поисковые методы** – один из основных инструментов работы с данными из типов. Мы уделим особое внимание двум типам поисковых методов – **конструкторный** (он же Конструктор запросов) и **SQL**. Оба этих метода в процессе своей работы могут генерировать SQL-запрос, отправлять его на исполнение в СУБД и возвращать результаты этого запроса. Но при этом у каждого из этих методов есть свои особенности и тонкости в настройке, которые мы далее рассмотрим.

**Примечание**: если тип является кешируемым, то конструкторный поисковый метод не проводит генерацию SQL-запроса и обращение к СУБД. Платформа проводит поиск записей на основе данных из кеша.

### 

### ****Общие особенности поисковых методов****

При создании поисковых методов необходимо учесть следующие особенности:

**1.** Параметры, у которых тип **Ссылка на объект**, при использовании в поисковом преобразовываются в их идентификаторы, т.е. [**ИДОБЪЕКТА**()/**OBJECTID**()](https://doc.caseplatform.tech/#0.0.0:0:0:reference_guide/udml_functions/system_functions/OBJECTID.html:0:ru).

**2.** Параметры, у которых тип **Перечисление**, при использовании в поисковом преобразовываются в их значения, а не наименования.

**3.** Параметры, которые были заполнены коллекцией, при использовании в поисковом преобразовываются в список значений через запятую. Например, если была подана коллекция записей из типа, то она превращается в перечисление идентификаторов.

**4.** При использовании UDML-функции [**КОЛИЧЕСТВО**()/**COUNT**()](https://doc.caseplatform.tech/#0.0.0:0:0:reference_guide/udml_functions/aggregate_functions/COUNT.html:0:ru) для **SQL поискового метода** платформа автоматически преобразовывает запрос в **select count(\*) from (запрос)**, таким образом в оперативную память не будут выгружаться записи из типа.

**5.** При записи результатов вызова пользовательского поискового метода в **js-переменную** платформа всегда будет записывать коллекцию, даже если в полученных результатах находится только 1 объект.

Рассмотрим подробнее эту особенность поисковых методов на примере. В рамках разработанного приложения для учета кадров создадим процесс, состоящий из трех блоков: блок старта процесса, скрипт и блок завершения процесса. В скрипте процесса мы вызовем поисковый метод для получения должности “Менеджер проекта” по его коду:

Log4JS.log('Вызов поискового метода для получения должности "Менеджер проектов" по коду:' , 'search\_method\_test.log');

var position = udm:(Main:position:by\_code('pm'));

Log4JS.log('Полученные результаты' , 'search\_method\_test.log');

Log4JS.log(position, 'search\_method\_test.log');

Log4JS.log('Являются ли полученные результаты коллекцией: ' + Array.isArray(position), 'search\_method\_test.log');

Затем мы запустим процесс и в созданном логе *search\_method\_test.log* мы можем увидеть, что полученные результаты вызова действительно **являются коллекцией**:

2022-02-04T09:59:53.221+0700 Полученные результаты

2022-02-04T09:59:53.222+0700 Коллекция [

          MObj type='Main.position' id='3'{

                    official\_name:

                    code:pm

                    active:true

                    name:Менеджер проектов

          }

]

2022-02-04T09:59:53.223+0700 Являются ли полученные результаты коллекцией: true

**Если в работе приложения ожидается, что в результате вызова поискового метода будет получен только один объект, то рекомендуется добавлять обращение к первому элементу массива, чтобы избежать возможных ошибок, например:**

var position = udm:(Main:position:by\_code('pm')[0]);

Но если поисковый метод вернул только 1 объект и запись результата проводится в неколлекционную UDML-переменную (например, атрибут процесса или переменная пользовательской функции), то платформа проводит автоматическую обработку и возвращает полученный элемент из коллекции без необходимости явного обращения к его первому элементу.

В ранее созданный процесс мы добавим **новый атрибут position**, проинициализируем процесс, и в блоке скрипта укажем следующий код:

Log4JS.log('Вызов поискового метода для получения должности "Менеджер проектов" по коду:');

udm:(position := Main:position:by\_code('pm'));

Log4JS.log('Полученные результаты' , 'search\_method\_test.log');

Log4JS.log(udm:(position), 'search\_method\_test.log');

Log4JS.log('Являются ли полученные результаты коллекцией: ' + Array.isArray(udm:(position)), 'search\_method\_test.log');

После запуска процесса в логе мы можем увидеть, что платформа записала **только 1 объект**:

2022-02-04T10:13:42.602+0700 Полученные результаты

2022-02-04T10:13:42.602+0700 MObj type='Main.position' id='3'{

          official\_name:

          code:pm

          active:true

          name:Менеджер проектов

}

2022-02-04T10:13:42.603+0700 Являются ли полученные результаты коллекцией: false

**6.** Поисковые методы возвращают только **записи из типа, в котором они были созданы**, даже если указанный тип в операторе FROM отличается от типа, в котором создан поисковый.

Приведем пример: в типе **position** мы создадим новый поисковый метод, который получает должность из типа **employee** по указанной фамилии. При его настройке мы явно укажем, что атрибут **position** будет иметь алиас (alias) **id**, чтобы платформа смогла корректно загрузить данные.

select

udm:(Main:employee.position) id

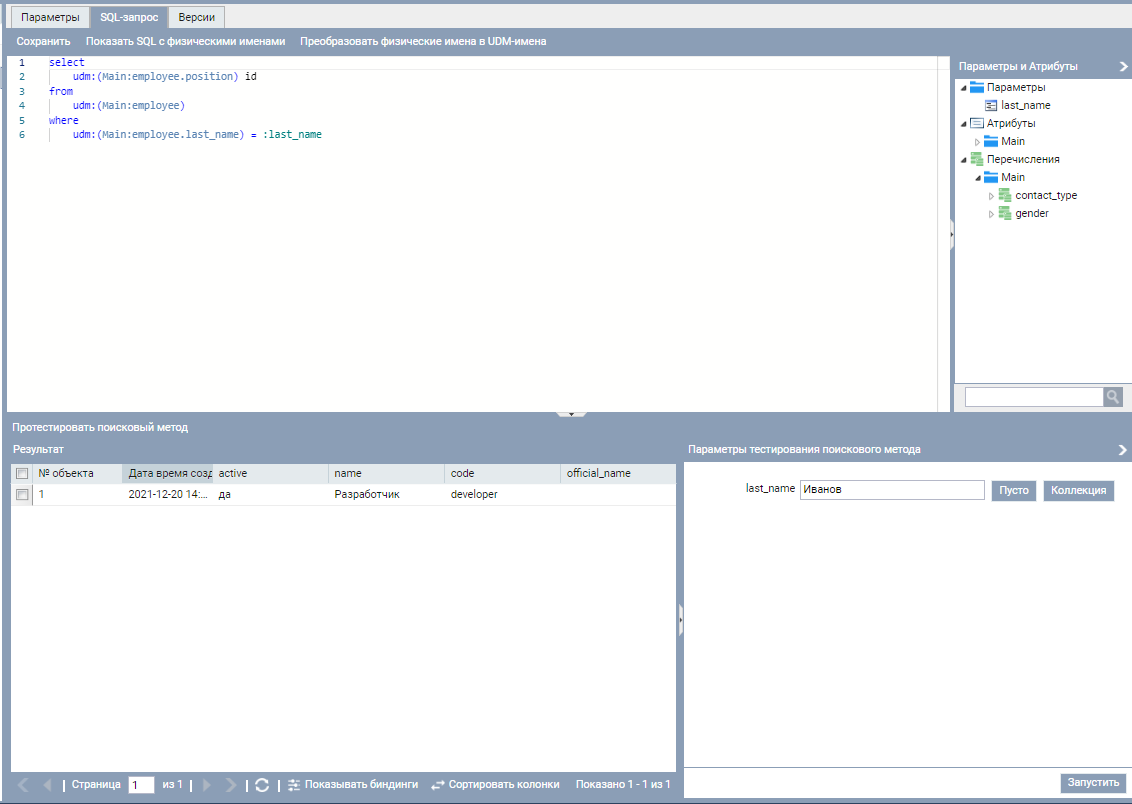
from

udm:(Main:employee)

where

udm:(Main:employee.last\_name) = :last\_name

При его тестировании мы укажем фамилию сотрудника, который занимает должность “Разработчик”:



Несмотря на то, что в качестве исходной таблицы был указан тип **employee,** платформа успешно вернула нужную должность.

**Кратко про транзакции**

При написании поисковых методов, а также разработке самого приложения необходимо учитывать транзакционность приложения. В основе работы с данными лежат **распределенные транзакции.**

**Транзакция** — это набор связанных задач (например, создание записи в таблице, обновление столбца у записи из таблицы и т.д.), который, помимо всего прочего, либо завершается успешно (фиксация / commit) или с ошибкой (отмена / rollback) как единое целое. У транзакции обычно выделяют свойства по аббревиатуре **ACID**:

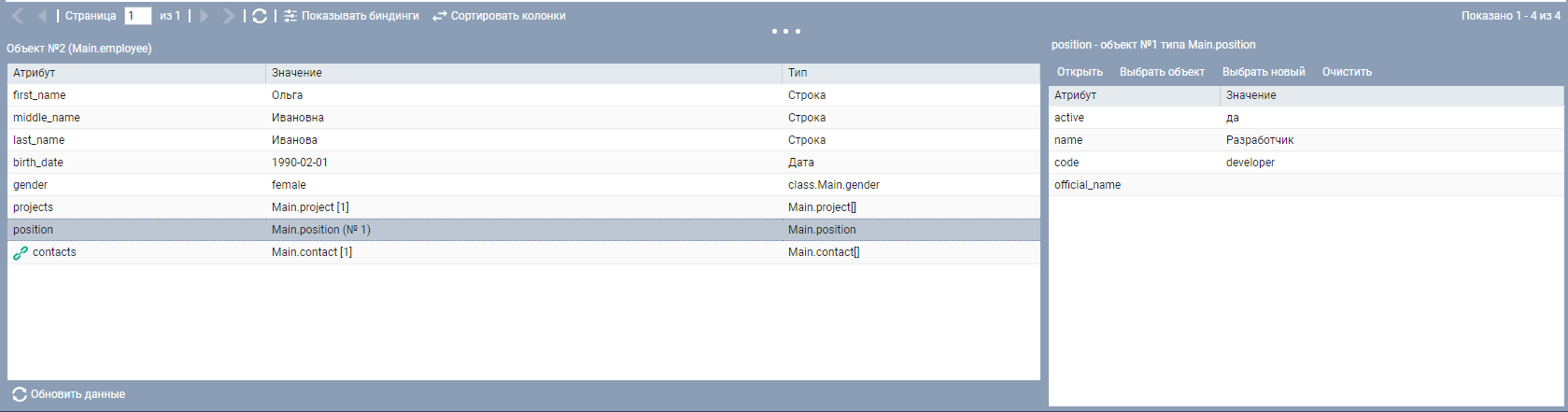
1. **A**tomicity (атомарность) — транзакция либо полностью применила все свои изменения, либо полностью откатилась (отменилась).
2. **C**onsitency (согласованность) — в конце транзакции система находится в согласованном состоянии.
3. **I**solation (изолированность) — параллельные транзакции не влияют друг на друга (это обеспечивает, например, phantom reads и non-repeatable reads), выполняются как будто последовательно.
4. **D**urability (надёжность) — завершённые (commited) транзакции сохраняются даже в случае сбоев и перезапуска системы.

**Распределенная транзакция** — это транзакция, затрагивающая несколько ресурсов. Для фиксации распределенной транзакции все участники должны гарантировать, что любое изменение данных будет постоянным. Изменения должны сохраняться даже в случае фатального сбоя системы или других непредвиденных событий. Если хоть один из участников не сможет предоставить такую гарантию, вся транзакция завершится с ошибкой и будет выполнен откат любых изменений данных внутри области транзакции.

В Case Platform **изменения записей в типе** происходят только при **фиксировании** транзакции. Фиксирование транзакции происходит в следующих случаях:

* при использовании функции [**ФИКСИРОВАТЬ**()/**COMMIT**()](https://doc.caseplatform.tech/#0.0.0:0:0:reference_guide/udml_functions/system_functions/COMMIT.html:0:ru);
* при использовании блока типа “Таймер” в процессе;
* при переходе на блок типа “Форма” в процессе;
* при переходе на асинхронный блок в процессе;
* при завершении процесса;
* при завершении сценарии сервера.

Рассмотрим работу транзакции на примере. В разработанной системе учета кадров мы откроем тип **employee**. У сотрудника Ивановой была установлена должность (атрибут **position**) — “Разработчик”:

Запись в типе до изменения

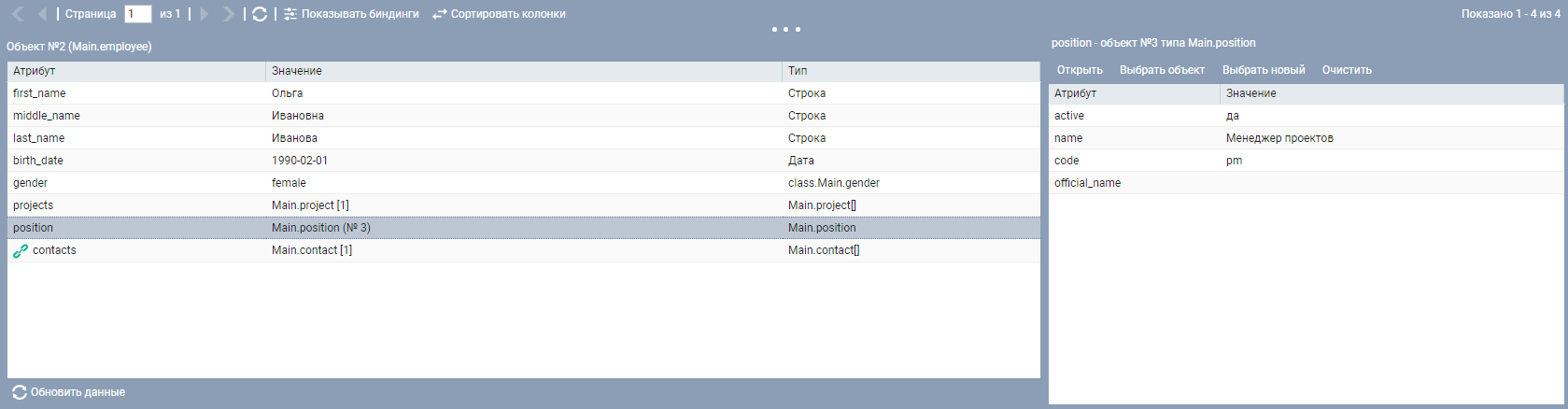
В этом приложении мы создадим новый процесс, который будет состоять только из трех блоков: блок старта процесса, скрипт и блок завершения процесса. Скрипт изменит должность у этого сотрудника с “Разработчик” на “Менеджер проекта”, но перед фиксированием транзакции (т.е. завершением процесса) мы установим небольшую паузу, в 10 секунд, при помощи Java-метода **sleep** (<https://docs.oracle.com/javase/tutorial/essential/concurrency/sleep.html>). Скрипт смены должности представлен ниже:

var employee = udm:(Main:employee:byId('2'));

employee.position = udm:(Main:position:by\_code('pm')[0]);

Packages.java.lang.Thread.sleep(10000);

Далее мы сохраним процесс и запустим его. После запуска повторно откроем тип **employee** иувидим, что у сотрудника Ивановой до сих пор установлена должность “Разработчик”. Но по истечении 10 секунд и обновлении данных в типе мы увидим, что запись в типе была обновлена, значение атрибута **position** было сменено на “Менеджер проекта”:

Запись в типе после изменения

**Особенности конструкторного поискового метода**

Рассмотрим основные особенности конструкторных поисковых методов.

**1. Как конструкторный поисковый метод преобразовывается к SQL**

При вызове поискового метода платформа преобразовывает настройки конструкторного поискового метода в SQL-запрос вида:

SELECT distinct(a0.ID)  from table\_name  as a0  where ((a0.column\_name=?))

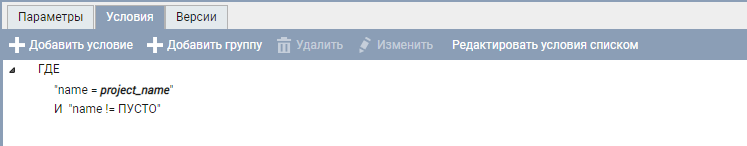
где:

* table\_name – это физическое название типа, в котором вызывается поисковый;
* условия, идущие после where - это предикаты поискового метода;
* column\_name – это физическое название столбца, которое указано в условии;
* ? – это значение параметра.

Case Platformвсегда указывает **id** в качестве выходного столбца и использует оператор**distinct**, чтобы получить только уникальные значения идентификаторов.

В случае если параметр поискового метода опционален и его значение не было указано, то Case Platform не будет генерировать условие и пропустит его. Но если параметр поискового метода не опционален, то Case Platform сгенерирует условие вида: **column\_name = null***.*

**Примечание**: в случае, если параметр поискового метода может быть пустым и параметр не опционален, стоит добавить проверку, что значение параметра не пустое. Это можно сделать перед его вызовом при помощи UDML-функции НЕ [**ЕНД**()](https://doc.caseplatform.tech/#0.0.0:0:0:reference_guide/udml_functions/properties_and_values_checking/IFNULL.html:0:ru) / NOT [**IFNULL**()](https://doc.caseplatform.tech/#0.0.0:0:0:reference_guide/udml_functions/properties_and_values_checking/IFNULL.html:0:ru), либо добавить дополнительное условие в поисковый метод, что проверяемый атрибут не пустой. Например:

Условие поискового метода, учитывающее непустоту атрибута

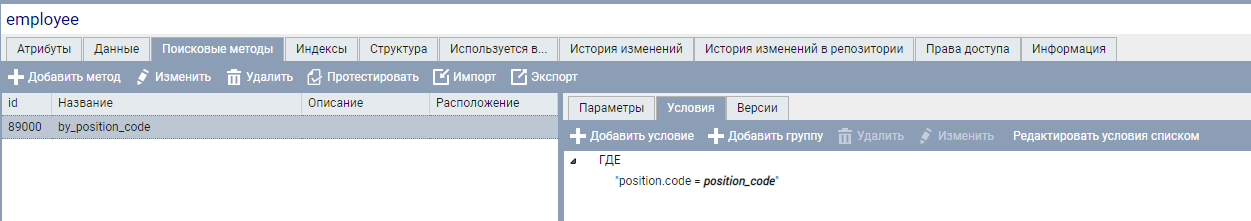
**2. Поиск записей с учетом изменений в памяти**

После исполнения SQL-запроса в базе данных и получения результатов, Case Platform для конструкторных поисковых методов дополнительно проверяет изменения, которые были проведены до **фиксирования транзакции** в базе данных. В случае, если в памяти были обнаружены новые или измененные записи, то Case Platform проводит дополнительную проверку этих записей на предмет соответствия условиям поискового метода.

Если запись не соответствует условиям, то она исключается из выборки, и наоборот, если запись теперь соответствует им, то она включается в выборку.

**Пример**

В рамках разработанного приложения для учета кадров нам необходимо получить всех сотрудников с должностью "Разработчик". Для этого мы создадим и настроим конструкторный поисковый, который ищет записи в типе **employee**по значению атрибута – **position.code**.

Условия поискового метода by position code

А в отдельном процессе мы создадим скрипт, который будет проводить поиск сотрудников и в атрибут **Result** записывать ID найденной записи и фамилию сотрудника. Скрипт приведен ниже:

var value\_for\_result = udm:(

text(

Main:employee:by\_position\_code('developer').each(employee|

'ИД записи:' + objectid(employee) + ', Фамилия: ' + employee.last\_name),

false,

'; '

)

);

udm:(Result := js:string(value\_for\_result));

Атрибут **Result** после запуска процесса будет иметь следующее значение:

ИД записи:1, Фамилия: Иванов; ИД записи:2, Фамилия: Иванова; ИД записи:4, Фамилия: Сергеев

В этом же скрипте настроим смену должности у сотрудника Ивановой без фиксирования транзакции – с “Разработчика” до “Менеджера проектов”:

var employee  = udm:(Main:employee:byId('2'));

employee.position = udm:(Main:position:by\_code('pm')[0]);

var value\_for\_result = udm:(

text(

Main:employee:by\_position\_code('developer').each(employee|

'ИД записи:' + objectid(employee) + ', Фамилия: ' + employee.last\_name),

false,

'; '

)

);

udm:(Result := js:string(value\_for\_result));

Значение атрибута **Result**:

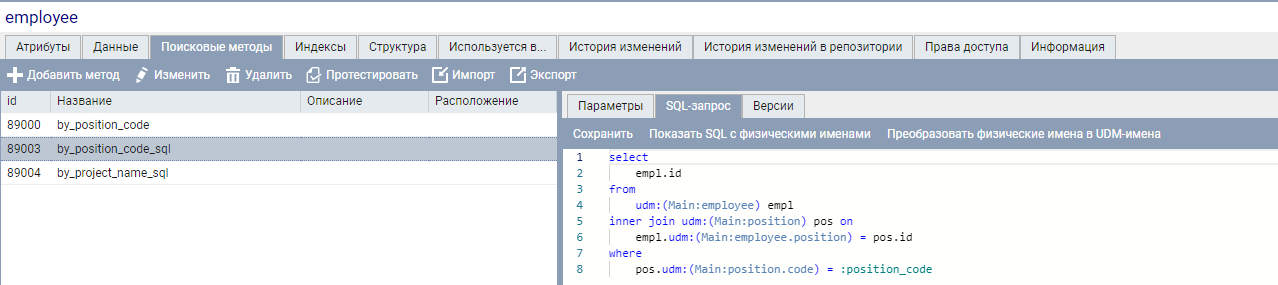
ИД записи:1, Фамилия: Иванов; ИД записи:4, Фамилия: Сергеев

Таким образом поисковый проверил изменения в памяти.

**Внимание**: у SQL-запросов такой дополнительной проверки не происходит, он ориентируется только на значения из базы данных.

**Пример:**

В рамках разработанного приложения мы создадим похожий SQL поисковый метод для получения сотрудников по должности:

SQL поисковый метод для поиска по должности

Сотруднику Ивановой мы вернем предыдущую должность – “Разработчик”, а в скрипте заменим вызов поискового на вызов нового SQL поискового метода:

var employee  = udm:(Main:employee:byId('2'));

employee.position = udm:(Main:position:by\_code('pm')[0]);

var value\_for\_result = udm:(

text(

Main:employee:by\_position\_code\_sql('developer').each(employee|

'ИД записи:' + objectid(employee) + ', Фамилия: ' + employee.last\_name),

false,

'; '

)

);

udm:(Result := js:string(value\_for\_result));

После его запуска мы увидим, что изменения в памяти **не были учтены** и в атрибут **Result** также была записана сотрудник Иванова:

ИД записи:1, Фамилия: Иванов; ИД записи:4, Фамилия: Сергеев; ИД записи:2, Фамилия: Иванова

**3. Дополнительные материалы**

<https://youtu.be/hQgvkzoeAVU> – видео, в котором подробно рассказано про особенности конструкторных поисковых методов.

**Особенности SQL поискового метода**

Рассмотрим основные особенности SQL поисковых методов и их настройки.

**1. Выходные столбцы**

Для получения записи из типа Case Platform необходим столбец с названием **id** (регистр не учитывается). Использование оператора \* или добавление других столбцов в select не рекомендуется, т.к. Case Platform при вызове поискового не использует остальные столбцы, в связи с чем он излишен и может негативно повлиять на производительность приложения.

Приведем пример в рамках разработанного приложения. Создадим поисковый в типе **position**, который получает записи по коду должности:

select

id

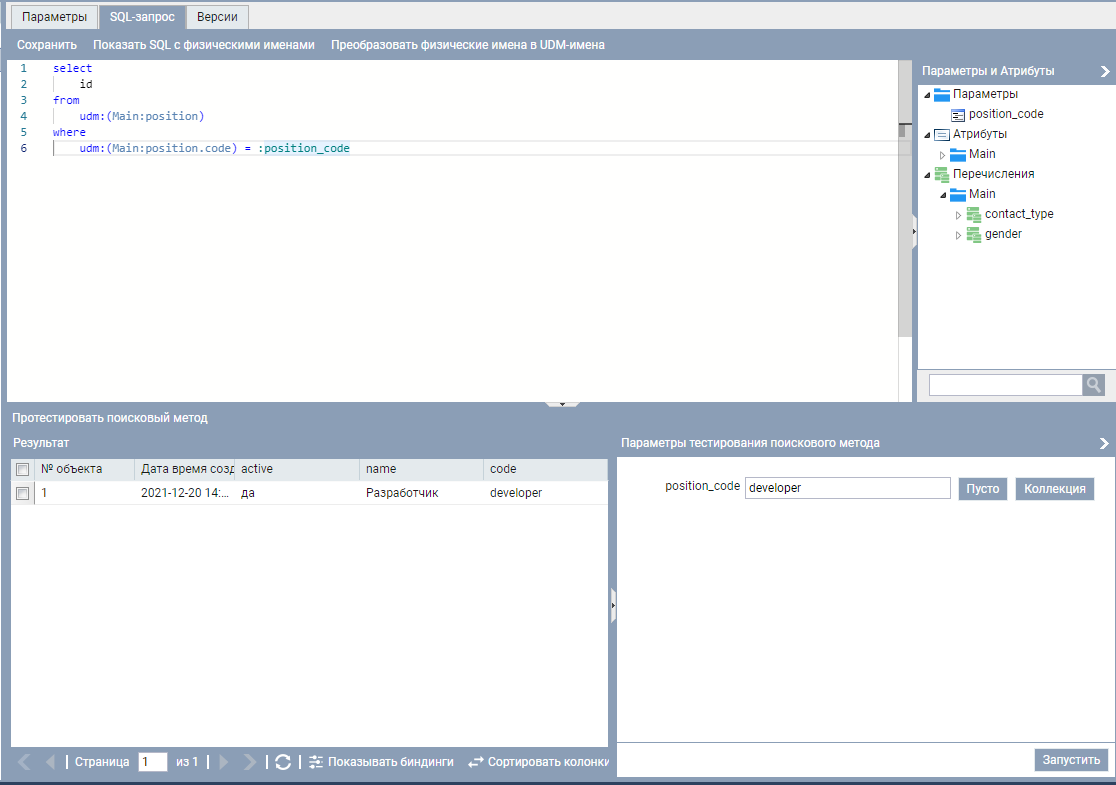
from

udm:(Main:position)

where

udm:(Main:position.code) = :position\_code

При его тестировании мы получаем нужную запись по переданному коду:

Использование столбца id в SQL поисковых методах

**2. Физические имена или UDM-имена?**

При написании SQL поисковых методов можно использовать как **физические имена** типов и атрибутов, так и их **логические имена**, т.е. UDM-имена. При выполнении поискового метода Case Platform автоматически заменит UDM-имена на физические, и после чего отправит запрос на исполнение в базу данных.

Применение физических или UDM-имен в SQL поисковых методах зависит от **способа выдачи физических имен в приложении:**

* Если используется автоматическая выдача физических имен, то нужно использовать только UDM-имена, так как есть большая вероятность того, что физические имена типов могут отличаться на разных стендах.
* Если используется ручная выдача физических имен, то можно использовать как UDM-имена, так и физические имена, т.к. они будут одинаковыми на разных стендах.

**3. Работа с отдельными типами в SQL**

Основы физического хранения типов были рассмотрены в уроках ранее, в рамках данного пункта кратко рассмотрим использование ссылок на другие типы и коллекционные атрибуты.

Для связи между таблицами применяются внешние ключи, **в дочерней** таблице (т.е. в таблице, которая содержит внешний ключ) хранится **строковый атрибут**, который связан с **первичным ключом родительской таблицы** (т.е. таблицы, на которую ссылается). И чтобы получить значение из родительской таблицы необходимо её «присоединить» при помощи JOIN.

Рассмотрим данную связь на примере: создадим SQL поисковый метод, который ищет сотрудников по коду должности:

select

empl.id

from

udm:(Main:employee) empl

inner join udm:(Main:position) pos on

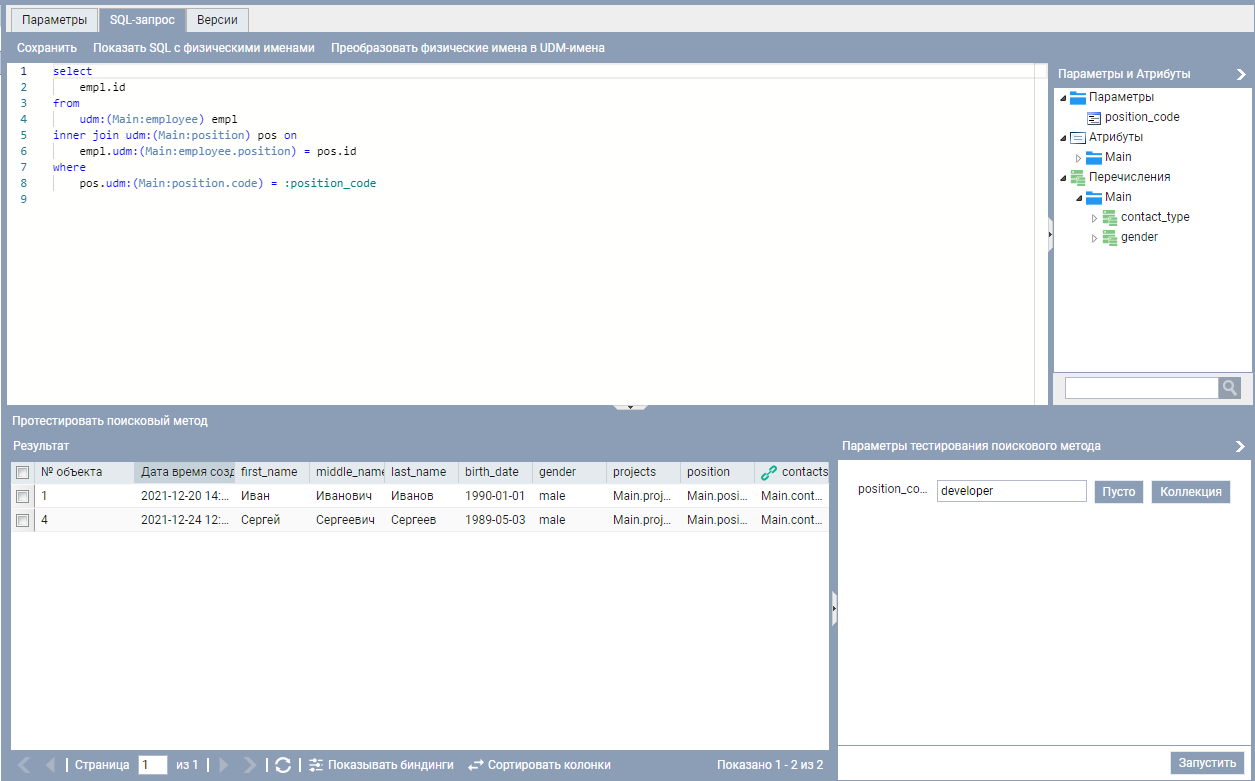
empl.udm:(Main:employee.position) = pos.id

where

pos.udm:(Main:position.code) = :position\_code

Для получения подробной информации о должности тип **position** был присоединен по соответствию внешнего ключа из **employee** и типа **position**.

Также можно заметить, что в SQL-запросе используются алиасы (alias) у типов – это строковые обозначения, следующие за таблицей: empl и pos. Это нужно чтобы упростить обращение к таблице и повысить уровень читаемости. Если эти алиасы не использовать, то мы можем получить ошибки, связанные с неоднозначными именами столбцов: таблицы могут иметь одинаковые названия столбцов (например, **id**) и СУБД не может различить, какой столбец к какой таблице относится.

Использование join для связи с таблицей

Для хранения **коллекционных атрибутов** Case Platform создает сопоставительные таблицы, которые состоят из трех столбцов:

* ID записи сопоставительной таблицы, **физическое имя** –**ID**;
* ID записи из типа, в которую добавлен коллекционный атрибут, **физическое имя**– **host\_mtype**;
* ID записи, на которую ссылается коллекционный атрибут, или же просто значение (если это коллекция простого типа, тип данных будет отличаться в зависимости от типа простых данных). Если это коллекционный атрибут ссылается на тип, то физическое имя – **value\_mtype**, иначе – **value**.

Чтобы получить значения из коллекционного атрибута, который ссылается на тип, необходимо сначала присоединить сопоставительную таблицу, а потом уже присоединить саму дочернюю таблицу.

Рассмотрим данную связь на примере: создадим SQL поисковый метод, который ищет сотрудников по названию проекта, в которых они участвуют:

select

empl.id

from

udm:(Main:employee) empl

inner join udm:(Main:employee.projects) empl\_proj on

empl.id = empl\_proj.host\_mtype

inner join udm:(Main:project) proj on

empl\_proj.value\_mtype = proj.id

where

proj.udm:(Main:project.name) = :project\_name

Для получения подробной информации о списке проектов, в которых работает сотрудник, была присоединена сопоставительная таблица по соответствию первичного ключа **employment**и внешнего ключа **host\_mtype**, а для получения подробной информации о самих проектах был присоединен **project**по соответствию первичного ключа project и внешнего ключа **value\_mtype**.

Отметим, что при присоединении сопоставительной таблицы не нужно указывать алиас таблицы, которая хранит этот коллекционный атрибут (например, empl.udm:(Main:employee.projects)) т.к. физически это отдельная таблица.

**4. Особенности применения параметров в SQL**

Далее перейдем к обзору особенностей применения параметров в SQL-запросах. Все используемые в поисковых методах параметры, которые Case Platform впоследствии заменит на переданное значение, начинаются с двоеточия.

**4.1. Опциональные параметры**

При использовании опциональных параметров в условиях необходимо добавлять конструкцию, которая позволит учесть пустое значение параметра. Есть несколько способов этого учета:

1. Использование конструкции :param is null or column = :param

Т.е. в случае если переданный параметр пустой, то мы не будем проверять это условие. Пример SQL поискового метода для СУБД Postgres ниже:

select

   m.id as id

from

   udm:(Мероприятие:Мероприятие) as m

where

   (

       cast(:Дата\_до as date) is null

       or m.udm:(Мероприятие:Мероприятие.Дата\_создания) <= cast(:Дата\_до as date)

   ) and (

       cast(:Дата\_от as date) is null

       or m.udm:(Мероприятие:Мероприятие.Дата\_создания) >= cast(:Дата\_от as date)

   )

В данном примере представленный фрагмент поискового использует данную конструкцию для необязательных фильтров в таблице.

**Примечание:** конструкция cast(:параметр as тип данных) при проверке на пустоту нужна для корректной работы поискового для СУБД Postgres. В случае, если будет использована конструкция :параметр is null, при подаче непустого значения параметра возникнет ошибка - *could not determine data type of parameter $1* - при подстановке значения из-за ограничений драйвера. Для других СУБД – Oracle, MS SQL – использование cast не требуется.

2. Использование SQL-функции coalesce() (см. <https://www.w3schools.com/sql/func_sqlserver_coalesce.asp>, <https://oracleplsql.ru/coalesce-function.html>)

Её можно использовать примерно таким же образом, что и в предыдущем способе:

SELECT

  m.id as id

FROM

   udm:(Мероприятие:Мероприятие) as m

where

   (

       coalesce(:Департамент, '0') = '0'

       OR  :Департамент = m.udm:(Мероприятие:Мероприятие.Ответственный\_департамент)

   ) AND  (

       coalesce(:Статус, null) is null

       OR  m.udm:(Мероприятие:Мероприятие.Статус\_мероприятия) (:Статус)

   )

Данный фрагмент поискового метода для таблицы использует также эту конструкцию для необязательных фильтров таблице. В случае если было передано пустое значения для параметра :ДепартаментСУБД заменит его на значение ‘0’ и успешно проверит его по первому указанному условию по проверке ответственного департамента.

**4.2. Использование коллекций в параметрах**

Как было указано ранее при использовании коллекции в параметрах поисковых методов Case Platform их заменяет на **список значений через запятую**. В связи с чем их применение в качестве условий будет отличаться от неколлекционных параметров: для проверки вхождения значения в коллекцию необходимо воспользоваться оператором in (<https://www.w3schools.com/sql/sql_in.asp>, <http://2sql.ru/novosti/sql-in/>)

Пример:

select id from udm:(Политика\_безопасности:Пользователь)

where udm:(Политика\_безопасности:Пользователь.Подразделение) in (:Подразделение)

Данный поисковый метод проводит поиск пользователей по переданным подразделениям.

Но стоит также учесть, что в случае, если параметр является опциональным, то нужно будет изменить условие для поиска. Конструкция вида :param is null or column in (:param) здесь не подойдет, т.к. возникнет синтаксическая ошибка, когда будет передан непустой параметр:

select

id

from

udm:(Политика\_безопасности:Пользователь)

where

'1', '2', '3' is null or udm:(Политика\_безопасности:Пользователь.Подразделение) in ('1', '2', '3')

В связи с чем обычно применяют функцию coalesce, т.к. она позволяет корректно учитывать все переданные параметры в метод:

select

id

from

udm:(Политика\_безопасности:Пользователь)

where

coalesce(:Подразделение, '0') = '0'

or udm:(Политика\_безопасности:Пользователь.Подразделение) in (:Подразделение)

**Примечание**: использование ‘0’ рекомендуется только для параметров типа **Ссылка на тип**, т.к. они в SQL-запросах преобразовываются в ID объекта и в Case Platform не могут быть созданы записи в типе с идентификатором ‘0’. В качестве альтернативного варианта можно применить конструкцию coalesce(:параметр, null) is null, которая также будет работать.

**Когда стоит применять конструкторные методы, а когда SQL?**

**Конструкторные поисковые методы** рекомендуется применять в следующих случаях:

* Когда тип, в котором создан поисковый метод, является кешируемым. В данном случае Case Platform при использовании конструкторного поискового метода не будет обращаться к БД, что снизит нагрузку на неё.
* Когда условий поискового метода не так много (меньше 4), либо они являются простыми. Например, поиск записей по внешнему ключу или поиск актуальных записей.

**SQL поисковые методы** рекомендуется применять в следующих случаях:

* Когда требуется применять более комплексные операции и функции в БД, например UNION, GROUP BY и т.д.
* Когда условий в поисковом методе много, например параметры используются для фильтрации записей.