

## Rejestry

### Tworzenie rejestru

Aby założyć rejestr w module rejestry musimy rozpocząć od założenia tabeli. Tabelę tworzymy za pomocą komendy *create table*.

Przykład założenia tabeli

```
-- Table: cregisters.creg_r_imi

-- DROP TABLE cregisters.creg_r_imi;

CREATE TABLE cregisters.creg_r_imi
(
    nazwa character varying(250),
    ulica character varying(80),
    budnr character varying(10),
    kwota double precision
)
INHERITS (cregisters.register_entry)
WITH (
    OIDS=FALSE
);
ALTER TABLE cregisters.creg_r_imi OWNER TO edokumenty;
GRANT ALL ON TABLE cregisters.creg_r_imi TO edokumenty;
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE cregisters.creg_r_imi TO http;
```

**Uwaga ! Pola używane przez eDokumenty są dziedziczone - nie trzeba ich zakładać. Są to pola:**

```
id____, cregid, uid____, is_del, adduid, adddat, lm_uid, lm_dat, doc_id, prc_id, cre_id, tpstid, stcuid, stcdat
```

Następnie w module rejestry zakładamy nowy rejestr, a w polu nazwa tabeli wprowadzamy nazwę założonej tabeli. System będzie od nas żądać, aby nazwa tabeli rozpoczynała się od "creg\_".

### Tworzenie raportu

Po utworzeniu tabeli w schema cregisters rozpoczynającej się od ciągu creg\_ należy utworzyć raport np:

```
SELECT ('CREGISTER_ENTRY') AS clsnam, cd.id____ AS keyval, cd.*
FROM cregisters.creg_ddm_dokumenty cd
INNER JOIN cregisters.creg_archiv_formularz af ON cd.menuid = af.formularz
WHERE {FILTER_STRING} AND cd.is_del IS NOT true
{ORDER_BY}
{LIMIT}
```

Raport należy podlinkować do rejestru ustawiając w tabeli registers pole rep\_id. W tabeli reports.reports dla rep\_id = raportowi dla rejestru należy ustawić is\_sys = TRUE

### Sumowanie oraz grupowanie

Lista pozycji rejestru obsługuje grupowanie oraz sumowanie zdefiniowane w raporcie. **Do działania niezbędne jest użycie w zapytaniu SQL, znacznika {ORDER\_BY}.**

Raport może zawierać grupowanie po kilku polach/kolumnach - należy je wpisać (rozdzielone przecinkiem) do sekcji "Grupowanie" na zakładce "Definicja" edytora raportów.

Możliwe jest sumowanie wartości poszczególnych kolumn - należy je wpisać (rozdzielone przecinkiem) do sekcji "Sumowanie" na zakładce "Definicja" edytora raportów.

### Ważne tabele

```

register (klucz główny: id____)
register_entry (klucz główny: id____ , klucz obcy: )
register_fields (klucz główny: id____, klucz obcy: )
register_links (klucz główny: id____, klucz obcy: )

```

Tabele:

- cregisters.register - lista rejestrów.
- cregisters.register\_entry -
- cregisters.register\_fields
- cregisters.register\_links

Uwaga! Kluczem obcym w registers\_entry referującym do rejestru jest XXXX

## Rejestr jako lista w dokumencie

Definiujemy powiązanie rejestru z typem dokumentu (jeżeli ma to być lista a nie formularz to ustawiamy parametr *collection* na *true*):

```

INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params)
VALUES ({cregisters.register.id____}, {types_of_documents.dctpid}, 'DOCUMENT', '{"collection":true}')

```

## Rejestr jako lista w zadaniu

Definiujemy powiązanie rejestru z typem zdarzenia:

```

INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params) VALUES ({cregisters.register.id____}, -1, 'EVENT',
(można zastępować EVENT innym typem zdarzenia: TODO, MEETING, PHONECALL).

```

## Rejestr jako lista w sprawie

Definiujemy powiązanie rejestru z kategorią spraw:

```

INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params) VALUES ({cregisters.register.id____}, {dossiers.dos_id}, 'EVENT',
Jeżeli w miejsce {dossiers.dos_id} wstawimy wartość -1, wówczas zakładka z listą pojawi się na każdej sprawie.

```

Jeżeli chcemy aby lista/zakładka pokazała się dopiero po dodaniu pierwszego wpisu to parametry należy dodatkowo ustawić parametr *always\_visible* na *false*.

```

{"collection":true,"always_visible":false}

```

## Podrejestr w rejestrze

Aby zbudować strukturę hierarchiczną rejestru wystarczy zlinkować odpowiednio 2 wcześniej utworzone rejestry. Pierwszy ze wskazanych zacznie się pojawiać jako lista rekordów w formacie rejestru nadrzędnego.

```

INSERT INTO cregisters.register_links (cregid, keyval, clsnam, params)
VALUES ({cregisters.register.id____}, {cregisters.register.id____}, 'REGISTER', '{"collection":true}')

```

**Uwaga! Id podrejestru jest wprowadzany w insercie jako pierwsze, następny jest id rejestru do którego będzie należeć podrejestr.**

W raporcie w podrejestrze za filtrowanie rekordów odpowiada makro {FILTER\_STRING}, które dokleja do zapytania warunek po atrybucie *cre\_id* (*cre\_id* wskazuje na rekord rejestru nadrzędnego).

## Definicja rejestru - parametry

### Walidacja wpisu w rejestrze

Walidacja odbywa się po zapisaniu formularza (rekord jest już w bazie ale transakcja nie jest jeszcze zatwierdzona). Dane zostaną zapisane jeżeli zapytanie SQL *validatorQuery* zwróci TRUE. W przeciwnym wypadku zmiany nie zostaną zapisane (ROLLBACK) i pokaże się komunikat o treści zdefiniowanej w parametrze *validatorMessage*.

przykład:

```
{ "validatorMessage": "Nieprawidłowe dane!", "validatorQuery": "SELECT (data_urodzenia < now()) AND (strlen(pesel) = 11) FROM "}
```

## Definicje pól dla rejestru

### Walidacja wartości w polach

przykłady:

```
-- liczba dowolnej długości
{ "validator": "/^\d+$/"}
-- kwota (np. 1111111,11)
{ "validator": "/^\d{1,7}(\?:[\.,]\d{1,2})?$/"}

```

ustalenie wymagalności dla pola:

```
{ "required": true}
```

### Ustawianie wartości domyślnych

Jeżeli chcemy aby pole było listą wyboru, to definiujemy w parametrach (*register\_fields.params*) domyślną wartość (*defaultValue*):

```
-- Id tworzącego dokument
{ "defaultValue": "{SQL::SELECT adduid FROM documents WHERE doc_id = {doc_id}}" }

-- domyślne dane zalogowanego użytkownika
{ "defaultValue": "{SQL::SELECT o.firnam || ' ' || o.lasnam || ' (' || COALESCE(o.orunsm, '') || ' - ' || o.ndenam || ') ' AS "}
```

Możliwe jest też ustawienie wartości wyliczanej za każdym razem gdy dokonujemy zapisu rejestru (dla pól ukrytych):

```
-- Imię i nazwisko dokonującego zmian w rejestrze
{ "value": "{SQL::select firnam || ' ' || lasnam from users where usr_id={LOGGED_USR_ID}}" }
```

1. **defaultValue** jest parsowane tylko dla formularza nowego wpisu w rejestrze (na akcji Open oraz Save).

**value** jest parsowane zawsze na akcji Save niezależnie od trybu (edycja, nowy) wyłącznie dla pól:

1. ukrytych poprzez definicję pola (*register\_fields.hidden = TRUE*)
2. ukrytych poprzez parametr *visible* (*register\_fields.params = {"visible": false}*)
3. nieaktywnych (*register\_fields.params = {"enabled": false}*)

### Pole jako lista wyboru

Jeżeli chcemy aby pole było listą wyboru, to definiujemy w parametrach (*register\_fields.params*) zapytanie zwracające rekordy typu (klucz, wartość), dodatkowo ustawiamy domyślną wartość (*defaultValue*):

```
{ "sql": "SELECT usr_id, lasfir FROM orgtree_view WHERE is_del IS NOT TRUE ORDER BY lasfir", "defaultValue": "{SQL::SELECT ad"}
```

Parametry: *sql*, *defaultValue*, są objęte standardowym mechanizmem parsowania [parametrów](#) (tak jak np. w przypisaniach w [workflow](#)).

### Listy połączone

Użycie znacznika pola, które jest listą wyboru, SQLu innej listy spowoduje jej automatyczne odświeżanie/filtrowanie.

przykład:

w rejestrze cregisters.creg\_moj\_rejestr, pole "grupa" jest zdefiniowana jako select z listą grup

```
{"sql":"SELECT grp_id,grpnam FROM groups"}
```

"pracownik" jest listą pracowników/użytkowników

```
{"sql":"SELECT usr_id,usrnam FROM users WHERE is_del IS NOT TRUE AND (CASE WHEN {cregisters.creg_moj_rejestr.grupa} = ' ' T
```

Taka konfiguracja spowoduje przeładowanie listy pracowników przy każdej zmianie grupy.

### Pole tekstowe typu HTML

```
{"type":"html"}
```

### Pole tekstowe typu ComboBox

```
{"type":"combobox","autoSearch":2,"sql":"SELECT usr_id,usrnam FROM users WHERE is_del IS NOT TRUE AND (firnam ~* E'^{SEARCH
```

Znacznik {SEARCH\_TEXT} zostanie zastąpiony wpisanym w pole tekstem

1. autoSearch - ilość znaków po których wpisaniu zostanie uruchomione wyszukiwanie / podpowiadanie (wartość -1 spowoduje wyłączenie automatycznego wyszukiwania i pokazanie ikony lupki)

### Pole typu Lookup

Pole to wygląda jak ComboBox. Różnica polega na tym, że wyszukiwanie odbywa się tylko za pomocą "lupki", a wartością pola będzie dana pobrana z bazy pod kluczem {valueField}. Wartość prezentowaną na formularzu określamy w parametrze {labelField}.

```
{"sql":"select usr_id, usrnam, 'USER' as clsnam FROM users WHERE {FILTER_STRING}","sql_filter":"firnam ~* E'^{SEARCH_TEXT}"
{"sql":"SELECT devcid, name__ || ' - ' || COALESCE(sernum) AS device FROM cregisters.creg_devices WHERE {FILTER_STRING}","
{"sql":"SELECT contid, COALESCE(name_2, name_1) || ' ' || f_addr AS caption, 'CONTACT' as clsnam FROM contacts_view WHERE
```

Kolumna clsnam w zapytaniu spowoduje pokazanie ikony "i" umożliwiającej otwarcie formularza powiązanego z danym clsnam (np. 'CONTACT' as clsnam da możliwość otwarcia panelu klienta). Ikona pojawi się po kliknięciu na lupkę (jeśli wyszukanie zakończyło się sukcesem).

Znacznik {SEARCH\_TEXT} zostanie zastąpiony wpisanym w pole tekstem

Znacznik {FILTER\_STRING} zostanie zastąpiony wartością z parametru "sql\_filter"

### Pole typu wybór z drzewka

Wybór wartości następuje poprzez zaznaczenia węzła z drzewka a następnie zapamiętanie opisu węzła w polu. Drzewko definiujemy za pomocą SQL. Dla przykładu niech posłuży drzewko magazynów dostępne poprzez słownik. Istotne jest aby kwerenda SQL zwracała odpowiednio nazwane pola (poprzez aliasy).

| Nazwa pola | Opis                                                                                                                                                                                                                  |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| keyval     | Klucz główny                                                                                                                                                                                                          |
| prn_id     | Klucz referujący do klucza głównego wskazujący na element nadrzędny                                                                                                                                                   |
| name       | Nazwa węzła                                                                                                                                                                                                           |
| icon       | Wyświetlana ikona. Można użyć słowa kluczowe jak FOLDER oraz ITEM, które wskazują odpowiednio na węzeł grupujący (FOLDER) oraz np. element końcowy struktury drzewa (ITEM). Przykład: foldery a w folderach dokumenty |

|        |                                                                                           |
|--------|-------------------------------------------------------------------------------------------|
| enable | Czy dany element jest aktywny. Jeśli nie ma tego atrybutu to domyślnie każdy jest aktywny |
|        |                                                                                           |

```
{"type":"dbtreeselector", "sql":"SELECT wahaaid AS keyval, prn_id, name__, 'FOLDER' AS icon__ FROM warehouses"}
```

### Pole jako status

W definicji pola, w polu Alias wpisujemy "tpstid"

### Disablowanie pola

Jeśli pole ma być tylko do odczytu to należy dla niego określić atrybut enabled:

```
{"enabled":false}
```

### ToolBar

```
{"type":"toolbutton", "icon":"new.gif", "visible":1, "doRefresh":true, "onclick":["moj_skrypt.inc", "MojaKlasa1", "mojaFunkcja",
```

1. icon: plik ikony bez ścieżki która wskazuje domyślnie na ./public\_html/framework/img/toolbaricons/24x24/

Skrypt "app/edokumenty/scripts/moj\_skrypt.inc"

1. doRefresh: wartość true spowoduje przeładowanie formularza wpisu w rejestrze

```
<?php
class MojaKlasa1 {

    public function __construct() {
    }

    public function mojaFunkcja($params) {
        $params = json_decode($params, TRUE);

        jscript::alert(json_encode($params));
    }
}
?>
```

Wywołanie / otwarcie formularza poprzez clsnam i keyval (np. otwarcie tego samego wpisu w nowym oknie czyli edycja):

```
{"type":"toolbutton", "icon":"edit.gif", "enabled":1, "onclick":["", "Application", "openDialogByCls", "", "CREGISTER_ENTRY", "SQL
```

Usuń wpis z rejestru:

```
{"type":"toolbutton", "icon":"del.gif", "enabled":1, "onclick":["", "Application", "openDialogByCls", {"mode":"del"}, "CREGISTER_
```

### Filtrowanie listy

Dla rejestru można ustawić stały filtr w parametrach (cregisters.register.params)

```
{"FILTER_STRING":"is_del IS TRUE"}
```

### Liczniki

Wykorzystując tabelkę i funkcję get\_counter można generować własne numery dla pozycji rejestrów.

```
{ "value": "{SQL::select case when {cregisters.creg_umowy_handlowe.nr_umowy}::text=' ' then user_workspace.get_nr_umowy('HO')}
```

## Modyfikacje JSON bezpośrednio w bazie danych

Sposób na zmianę wartości jednego pola w obiekcie typu JSON (dla PostgreSQL v9.3+):

```
CREATE OR REPLACE FUNCTION "json_set_value"(
    "json"          json,
    "key_to_set"    TEXT,
    "value_to_set"  anyelement
)
RETURNS json
LANGUAGE sql
IMMUTABLE
STRICT
AS $function$
SELECT COALESCE(
    (SELECT ('{' || string_agg(to_json("key") || ':' || "value", ',') || '}')
     FROM (SELECT *
           FROM json_each("json")
           WHERE "key" <> "key_to_set"
           UNION ALL
           SELECT "key_to_set", to_json("value_to_set")) AS "fields"),
    '{}'
)::json
$function$;

UPDATE cregisters.register_field SET params = json_set_value(params, 'doRefresh', true) WHERE id_____ = 1;

UPDATE cregisters.register_field SET params = json_set_value(params, 'value', 'SQL::SELECT ''tekst "koło"'') WHERE id_____
```

## Migracja rejestrów z innej bazy

[Import rejestrów](#)

## Przydatne konstrukcje i zapytania

```
-- użycie w parametrach do przycisków i pól wartości {DOC_ID} powoduje błąd po wejściu na rekord rejestru jeśli jest pusty
select pprosm from documents where doc_id = COALESCE(NULLIF('{DOC_ID}',''), '0')::int
```