

Title: Rejestry

Subject: eDokumenty - elektroniczny system obiegu dokumentów, workflow i CRM -
DeployerGuide/AdvancedConfiguration/CustomRegisters

Version: 103

Date: 05/13/24 10:33:21

Table of Contents

| | |
|---|---|
| <i>Rejestry</i> | 3 |
| <i>Tworzenie rejestru</i> | 3 |
| <i>Tworzenie raportu</i> | 3 |
| <i>Ważne tabele</i> | 3 |
| <i>Rejestr jako lista w dokumencie</i> | 4 |
| <i>Rejestr jako lista w zadaniu</i> | 4 |
| <i>Podrejestr w rejestrze</i> | 4 |
| <i>Definicje pól dla rejestru</i> | 4 |
| <i>Walidacja wartości w polach</i> | 4 |
| <i>Ustawianie wartości domyślnych</i> | 4 |
| <i>Pole jako lista wyboru</i> | 5 |
| <i>Pole tekstowe typu HTML</i> | 5 |
| <i>Pole tekstowe typu ComboBox</i> | 5 |
| <i>Pole jako status</i> | 5 |
| <i>Disablowanie pola</i> | 5 |
| <i>ToolBar</i> | 5 |
| <i>Filtrowanie listy</i> | 6 |
| <i>Modyfikacje JSON bezpośrednio w bazie danych</i> | 6 |
| <i>Migracja rejestrów z innej bazy</i> | 6 |
| <i>Przydatne konstrukcje i zapytania</i> | 6 |

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

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 na 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 (TODO,PHONECALL,EVENT)

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

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 podrejeście za filtrowanie rekordów odpowiada makro {FILTER_STRING}, które dokleja do zapytania warunek po atrybucie cre_id (cre_id wskazuje na rekord rejestru nadrzędnego).

Definicje pól dla rejestru

Walidacja wartości w polach

przykłady:

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

```

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,usrnam FROM users WHERE is_del IS NOT TRUE", "defaultValue": "{SQL::SELECT adduid FROM documents WHERE" }
```

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

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 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 ./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","","","REGISTER_ENTRY","SQL::SELECT 'tekst "kolo"''],"value":"SQL::SELECT 'tekst "kolo"'"};
```

Usuń wpis z rejestru:

```
{"type":"toolbutton","icon":"del.gif","enabled":1,"onclick":["","Application","openDialogByCls",{ "mode":"del"}, "REGISTER_ENTRY","SQL::DELETE FROM documents WHERE doc_id = {DOC_ID}"],"value":"SQL::DELETE FROM documents WHERE doc_id = {DOC_ID}"};
```

Filtrowanie listy

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

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

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 "kolo"'') WHERE id_____ = 1;
```

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
```