

# 実践2 『登録画面』を作成する

## 事前作業

### 手順



## 画面の確認

「例：『サービス登録画面』 DG\_170Uを作成する。」

「『一覧画面』で [登録訂正ボタン] を押し、『登録・訂正画面』へ移動する。」

### ①一覧画面

The screenshot shows a web application interface with a navigation bar at the top containing the Rmenu logo and various menu items like '登録訂正', '検索', 'システム機能一覧', etc. The main content area displays a table with columns for 'サブシステム', '管理単位', 'サービスID', 'サービス名称', and '概要'. The first row of data is highlighted with a red dashed box, showing '共通マスタ' for both 'サブシステム' and '管理単位', and '11111' for 'サービスID'. Below the table are navigation buttons for '最初: F9', '前へ: F10', '1/1ページ(1件)', '次へ: F11', and '最後: F12'.



## ER図を確認する

「サービスのデータの更新時の注意☆サービスから出ているリレーションに注意すること。

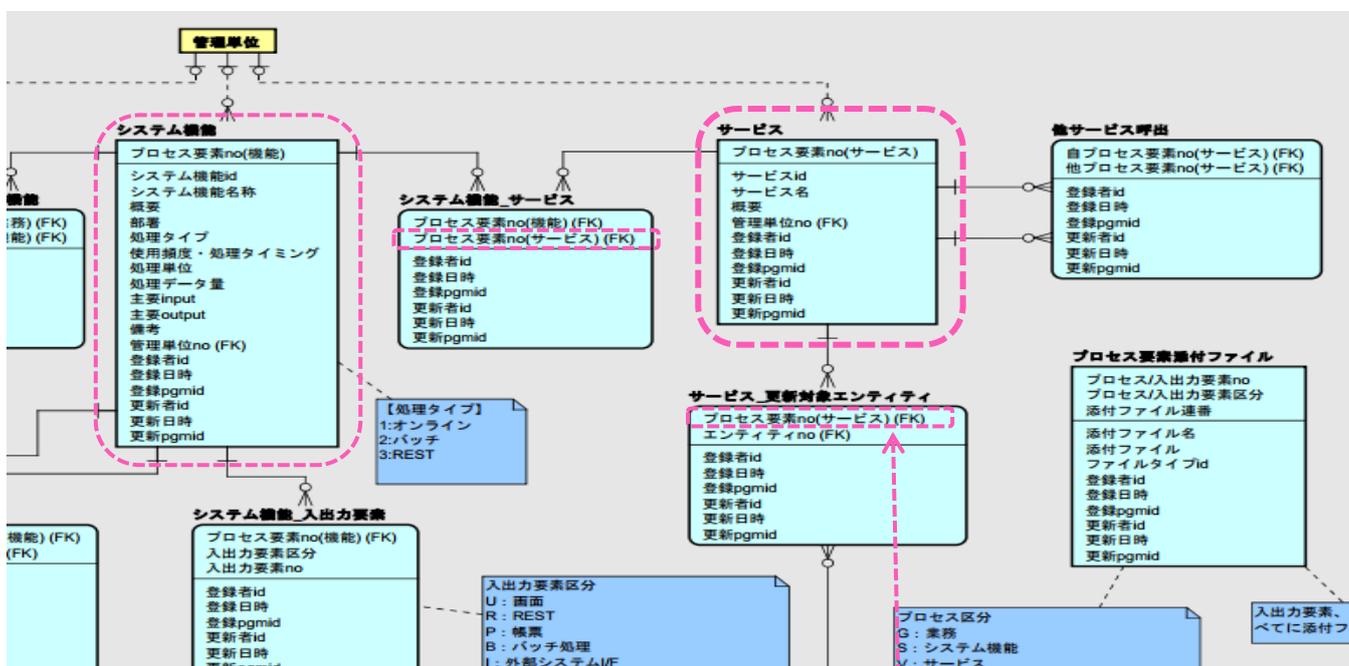
サービスで項目を削除するときに子を残したまま削除ができないようにする。

親無しの項目が残らないかER図を確認すること。」

※子テーブルのkey上段に親のkeyがある場合は親を消さないこと。

### ※SQL作成時に要確認

[ER図]



## プログラムのコピー

「一覧登録で作成したDG\_160Rの登録画面の為プログラム名 [DG\_170U] を作成する。

※DG\_140Rの登録画面版のプログラム [DG\_150U] をコピーする。」

# クライアント側を作成する

「一覧画面作成時とupdateSQLまでは同じ作業の為復習になります。」※詳細は一覧画面作成で確認

## ①Html

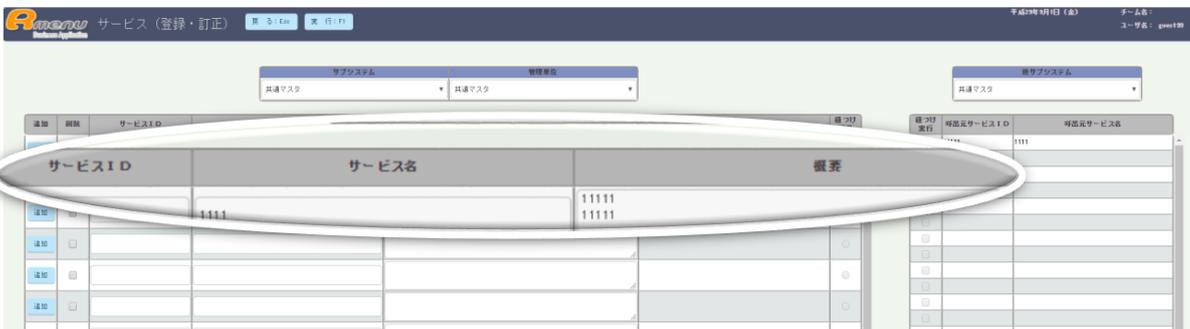
### 1. 「タイトルを変更する」 (2か所ある)

[画面]



### 2. 「項目名を作成する」 (3か所づつあるので忘れないように注意すること)

[画面]



[Html] 例：サービスID

```
<thead>
<tr>
<th class="width50">追加</th>
<th class="width50">削除</th>
<th class="width160">サービスID</th>
<th class="width300">サービス名</th>
<th class="width400">概要</th>
</tr>
</thead>
</table>
</div> <!-- テーブル スクロール エンド -->
<div id="scrollable-TableD1"> <!-- テーブル スクロール スタート -->
<table id="mainTable1" class="table table-condensed mmenuable">
<tbody>
<tr>
<td class="width50">
<input name="行追加" type="button" class="行追加 btn color4 btn-sm rfocusblue" value="追加">
</td>
<td class="width50">
<input name="削除" type="checkbox" class="削除 rfocusblue" value="削除">
</td>
<td class="width160">
<input name="サービスID" type="text" value="" class="form-control input-sm rfocusblue menuLeft サービスID">
</td>
<td class="width300">
<input name="サービス名" type="text" value="" class="form-control input-sm rfocusblue menuLeft サービス名">
</td>
<td class="width400">
<textarea name="概要" rows="3" class="form-control input-sm rfocusblue mmenuLeft 概要"></textarea>
</td>
</tr>
</tbody>
</table>
```

### 3. 「入力エリアの作成」

「インプット・テキストエリア・チェックBOXを作成する

どれにするのかは、ポスグレでレコードの最大文字数を確認して決める」

#### 3-1. 「インプット」

[画面]



```
<td class="width160">↓  
<input name="サービスID" type="text" value="" class="form-control input-sm rfocusblue menuLeft サービスID">↓  
</td>↓
```

#### 3-2. 「テキストエリア」 ※テキストエリアは改行が可能になる。



```
<td class="width400">↓  
<textarea name="概要" rows="3" class="form-control input-sm rfocusblue menuLeft 概要"></textarea>↓  
</td>↓
```

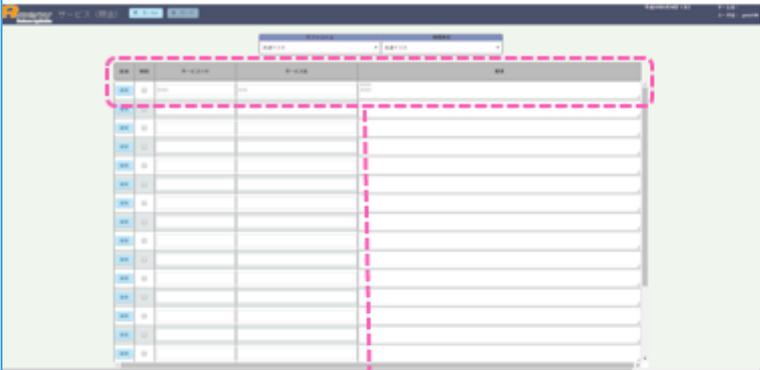
#### 3-3. 「チェックBOX」



```
<td class="width50">↓  
<input name="有剰余" type="checkbox" class="削除 rfocusblue" value="有剰余">↓  
</td>↓
```

## 4. 「幅を決める」

[画面]



[Html]

```
6 <div id="scrollable-table1" />
```

### ③Appspec

#### 1. 「ネクストname」作成

beforenameを作成する。Nextnameはないので""空白になる。

[Appspec]

```
5 //インスタンスプロパティを追加する↓
6 var AppSpec = App.AppSpec = new $R.Class($R.AppSpec);↓
7 AppSpec.fn.init = function(name) {↓
8   $R.log("AppSpec:initu:start");↓
9   ↓
10  this.name = name;↓
11  this.nextname = "";↓
12  this.beforename = "DG_140R";↓
13  ↓
14  $R.log("AppSpec:initu:end");↓
```

#### 2. 「ヘッダーの引き継ぎの確認」

「セレクトBOXで選択した内容を前の画面から引き継ぎ、照会画面で表示できるようにする。」

[Appspec]

```
136 //前画面からの引き継ぎデータ定義↓
137 //↓
138 ,beforeStorageData: {↓
139   ↓
140   dataset id: "header"↓
141   , dataname: ["検索サブシステムNO", "検索管理単位NO"]↓
142   , classname: ["検索サブシステム選択", "検索管理単位選択"]↓
143   , typename: ["select", "select"]↓
144 }
```

#### 3. 「チェックBOXの確認」

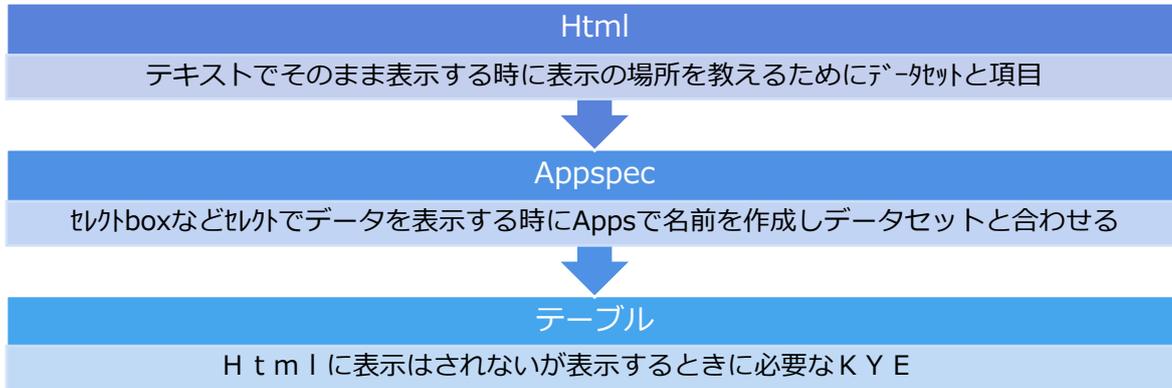
「削除のチェックBOXが増えているので指示を確認する。コピー元と同じの為触らない。」 [Appspec]

```
88 ,selectorEvent: {↓
89   //ここから追加処理↓
90   ["#検索サブシステム選択", "change", "onChange検索サブシステム選択"]↓
91   ["#検索管理単位選択", "change", "onChange検索管理単位選択"]↓
92   ["#検索他サブシステム選択", "change", "onChange検索他サブシステム選択"]↓
93   ["削除", "change", "onChange削除"]↓
94 }↓
-----↓
188 //↓
189 //チェックボックス定義↓
190 //selectorid: HTMLのIDを指定する↓
191 //selectorname: HTMLの名を指定する (テーブル内で使用する時に定義する) ↓
192 //dataset id: チェックボックスの値が設定されるdatasetのID名を指定する↓
193 //datasetname: チェックボックスの値が設定されるdatasetの項目名を指定する↓
194 //value: ↓
195 //on: チェックされた時に取る値を指定する↓
196 //off: チェックが外された時に取る値を指定する↓
197 //↓
198 checkbox1: {↓
199   ↓
200   selectorid: "", selectorname: "削除", dataset id: "detail", datasetname: "削除"↓
201   , value: {on: "9", off: "0"}↓
202 }↓
203 }↓
204 }
```

# サーバー側 JSONを作成する

## データセットJSON

「必要な項目名をセットする」



## バリデーションJSON

「データセットから項目をそろえる」

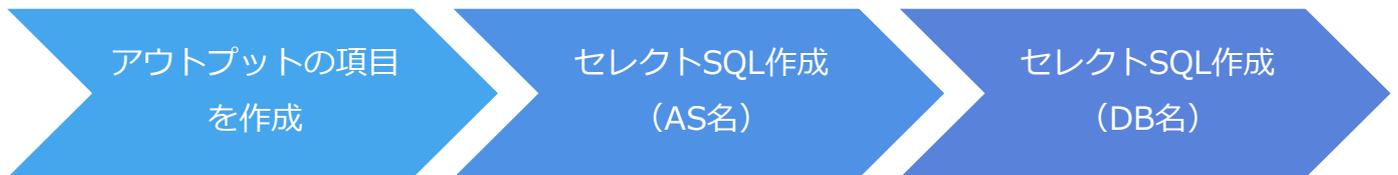
## トランJSON

「データセットから項目をそろえる」

## selectSQLJSON

「SQLJSONの作成は**セレクトSQL**と**アップデートSQL**の2つを作成する。

**照会ボタンを押下するとセレクトSQLが動き、登録ボタンを押下するとアップデートSQLが動く。」**



1.selectSQLのアウトプットの項目を作成する。

「トランJSONと項目名を合わせる」

2.selectSQLのアウトプットの項目データが取得できるようにSQLを作成する

※ DBの項目名（半角小文字）に注意

※「¥”」の付け忘れ注意

※ここまでは一覧画面作成と同じ

アップデート  
updateSQLJSON

「updateSQLは登録画面そのものになる。新規・削除・更新がこのSQLJSONでできる。」

手順



①updateSQLの役割とは

「updateSQLは、1つの画面で **新規・削除・更新** の3つのSQLを実行できる。」

delete	削除（デリート）
update	新規（インサート）
insert	更新・上書き（アップデート）

②updateSQLの構成

「このupdateSQLは6つのidで構成されています。」

※1画面で3つの処理を見分けて実行する

削除チェック1	"id": "deleteCheck1",
削除チェック2	"id": "deleteCheck2",
削除処理	"id": "delete",
更新処理	"id": "update",
新規処理	"id": "insert",
セレクト	"id": "detail",

- 「子テーブルのデータの有無を確認する  
削除できないときはエラーメッセージを出す」
- 「データの削除を行う  
データの変更を上書き更新する  
新規登録をする」
- 「実行後にDBのデータをセレクトで表示する」

### ③updateSQLのアウトプットを作成する

「id:detailのアウトプットから作成する。データセット・トランを参考に項目名を作成する。

ジェネレートSQLの時は、[field] [table] の作成に注意する。」

[updatesql]

<pre>370 "output": {↓ 371   "multiline": "yes",↓ 372   "record": {↓ 373     "input要素NO": {↓ 374       "value": [""],↓ 375       "field": "プロセス要素no(サービス)%",↓ 376       "table": "A"↓ 377     },↓ 378     "サービスID": {↓ 379       "value": [""],↓ 380       "field": "サービスid",↓ 381       "table": "A"↓ 382     },↓ 383     "サービス名": {↓ 384       "value": [""],↓ 385       "field": "サービス名",↓ 386       "table": "A"↓ 387     },↓ 388     "概要": {↓ 389       "value": [""],↓ 390       "field": "概要",↓ 391       "table": "A"↓ 392     } 393   } 394 }</pre>	<p>←アウトプット</p> <p>←項目名作成</p> <p>←ジェネレートSQLの場合fieldにDBの項目名を書く (ASの代わり)</p> <p>←ジェネレートSQLの場合tableへFROMを設定する (この項目をどこから持ってくるのかの指示)</p>
---	---

### ④updateSQLの「照会処理」を作成する

「id:detailのselectを作成する。項目名 [サービス] へ変更する。」

[updatesql]

<pre>40 "comment": "サービス 照会処理",↓ 41 "id": "detail",↓ 42 "before": "",↓ 43 "after": "",↓ 44 "sql": {↓ 45   "type": "select",↓ 46   "reesql": {↓ 47     "genesql": {↓ 48       "dist": "",↓ 49       "from": ""↓ 50     },↓ 51     "サービス AS A"↓ 52   },↓ 53   "where": "管理単位no = ?",↓ 54   "order": "A.サービスid"↓ 55 },↓ 56 "input": {↓ 57   "multiline": "no",↓ 58   "record": {↓ 59     "管理単位no": {↓ 60       "value": [""],↓ 61       "field": "管理単位no",↓ 62       "frontype": "request",↓ 63       "fromid": "header",↓ 64       "fromio": "",↓ 65       "fromname": "検索管理単位NO"↓ 66     } 67   } 68 },↓ 69 "output": {↓ 70   "multiline": "yes",↓ 71   "record": {↓ 72     "input要素NO": {↓ 73       "value": [""],↓ 74       "field": "プロセス要素no(サービス)%",↓ 75       "table": "A"↓ 76     },↓ 77     "サービスID": {↓ 78       "value": [""],↓ 79       "field": "サービスid",↓ 80       "table": "A"↓ 81     },↓ 82     "サービス名": {↓ 83       "value": [""],↓ 84       "field": "サービス名",↓ 85       "table": "A"↓ 86     },↓ 87     "概要": {↓ 88       "value": [""],↓ 89       "field": "概要",↓ 90       "table": "A"↓ 91     } 92   } 93 }</pre>	
---	--

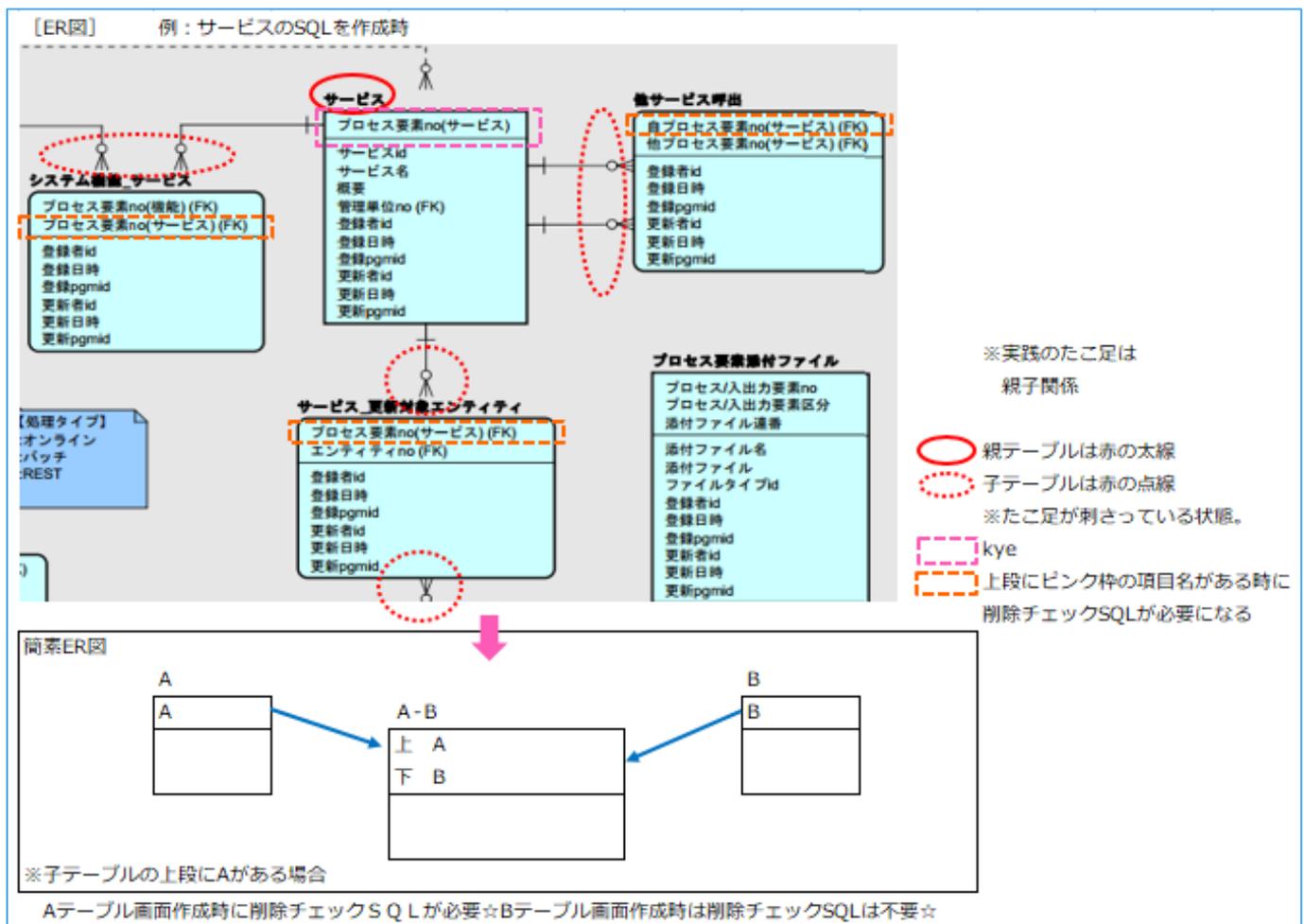
## ⑤updateSQLの「削除チェック」部分を作成する

### 1. 「ER図の確認をする」

「データの削除実行時に、データを消しても良いかチェックが必要になる。」

ER図を確認するとテーブルの親子関係がわかり、削除チェックSQLを作成できる。」

[ER図] 例：サービスのSQLを作成時



ER図を確認した結果：「ER図を確認すると2回削除チェックが必要になる。」

※子テーブルがあるのに親テーブルのデータを削除してしまうと

子テーブルのデータが浮いてしまう為、子テーブルのデータの有り/無しをチェックしてから

親テーブルのデータ削除を更新できる仕組みが削除チェックSQL。」

## 2. 「1つ目の削除チェック作成」

[updatesql]

```
11 sqls : {}
12 comment : "削除データチェック1のシステム機能 入出力要素",
13 id : "deleteCheck1",
14 before : "setDeleteDataOfDrmTools('detail1','deleteCheck1','入出力要素NO')",
15 after : "",
16 sql : "",
17 type : "select",
18 check : "found error",
19 errmsg : "サービス 更新対象エンティティのデータが、サービス 更新対象エンティティに存在します。削除出来ません、確認して下さい。",
20 freesql : "",
21 genesql : "",
22 dist : "",
23 from : "サービス 更新対象エンティティ",
24 where : "¥プロセス要素no(サービス)¥",
25 order : "",
26 },
27 input : {
28 multiline : "yes",
29 record : {
30 "入出力要素NO": {
31 value : ""
32 },
33 "field": "¥プロセス要素no(サービス)¥",
34 },
35 },
36 output : {
37 multiline : "yes",
38 record : {
39 "プロセス要素NO機能": {
40 value : ""
41 },
42 "field": "¥エンティティno¥",
43 }
```

←1つ目の削除チェック

←エラー時のエラーメッセージの設定を作成する

←条件はこのレコードの有無

プロセス要素no(サービス) (FK)
エンティティno (FK)
登録者id
登録日時
登録pgmid
更新者id
更新日時
更新pgmid

## 3. 「2つ目の削除チェック作成」

[updatesql]

```
18 comment : "削除データチェック2のシステム機能 入出力要素",
19 id : "deleteCheck2",
20 before : "setDeleteDataOfDrmTools('detail1','deleteCheck2','入出力要素NO')",
21 after : "",
22 sql : "",
23 type : "select",
24 check : "found error",
25 errmsg : "他サービス呼出のデータが、他サービス呼出に存在します。削除出来ません、確認して下さい。",
26 freesql : "",
27 genesql : "",
28 dist : "",
29 from : "他サービス呼出",
30 where : "¥自プロセス要素no(サービス)¥",
31 order : "",
32 },
33 input : {
34 multiline : "yes",
35 record : {
36 "入出力要素NO": {
37 value : ""
38 },
39 "field": "¥自プロセス要素no(サービス)¥",
40 },
41 },
42 output : {
43 multiline : "yes",
44 record : {
45 "プロセス要素NO機能": {
46 value : ""
47 },
48 "field": "¥他プロセス要素no(サービス)¥",
49 }
```

←エラー時のエラーメッセージの設定を作成する

←条件はこのレコードの有無

自プロセス要素no(サービス) (FK)
他プロセス要素no(サービス) (FK)
登録者id
登録日時
登録pgmid
更新者id
更新日時
更新pgmid

- ・ エラーメッセージをそれぞれ作成する（コピー元のテーブル名をこのテーブル名へ変更する。）
- ・ テーブル名とレコード名を作成する（コピー元のテーブル名をこのテーブル名へ変更する。）

## ⑥updateSQLの「削除」部分を作成する

「削除実行時に、データを消しても良いかチェックができたので、削除するSQLを作成する。」

[updatesql・削除処理]

```
84 | comment : "サービス〇削除処理",↓
85 | id : "delete",↓
86 | before : setDeleteDataOfDrmltools('detail', 'delete', '入出力要素NO'),↓
87 | after : "",↓
88 | sql : ↓
89 | type : "delete",↓
90 | freesql : "",↓
91 | genesql : ↓
92 | dist : "",↓
93 | from : "サービス",↓
94 | where : "プロセス要素no(サービス)%" = "?" ,↓
95 | order : ↓
96 | ↓
97 | ↓
98 | input : {↓
99 | multiline : "yes",↓
00 | record : {↓
01 | 入出力要素NO : {↓
02 | value : [""],↓
03 | field : "プロセス要素no(サービス)%"↓
04 | ↓
05 | }
```

サービス
プロセス要素no(サービス)
サービスid
サービス名
概要
管理単位no (FK)
登録者id
登録日時
登録pgmid
更新者id
更新日時
更新pgmid

## ⑦updateSQLの「更新処理」部分を作成する

「すでに入力済の項目のデータが変更されたときの処理。項目名を作成する。」

```
84 | comment : "サービス〇更新処理",↓
85 | id : "update",↓
86 | before : setUpdateDataOfDrmltools('detail', 'update', '入出力要素NO'),↓
87 | after : "",↓
88 | sql : ↓
89 | type : "update",↓
90 | freesql : "",↓
91 | genesql : ↓
92 | dist : "",↓
93 | from : "サービス",↓
94 | where : "プロセス要素no(サービス)%" = "?" ,↓
95 | order : ↓
96 | ↓
97 | ↓
98 | input : {↓
99 | multiline : "yes",↓
00 | record : {↓
01 | サービスID : {↓
02 | value : [""],↓
03 | field : "サービスid",↓
04 | ↓
05 | サービス名 : {↓
06 | value : [""],↓
07 | ↓
08 | 概要 : {↓
09 | value : [""],↓
10 | ↓
11 | 検索管理単位NO : {↓
12 | value : [""],↓
13 | field : "管理単位no",↓
14 | fromtype : "request",↓
15 | fromid : "header",↓
16 | fromio : "header",↓
17 | fromname : "検索管理単位NO",↓
18 | ↓
19 | 更新日時 : {↓
20 | value : ["CURRENT_TIMESTAMP"],↓
21 | funct : "CURRENT_TIMESTAMP",↓
22 | ↓
23 | 更新者id : {↓
24 | value : [""],↓
25 | fromtype : "request",↓
26 | fromid : "login",↓
27 | fromio : "login",↓
28 | fromname : "ユーザID",↓
29 | ↓
30 | 更新pgmid : {↓
31 | value : ["DG_170U"],↓
32 | ↓
33 | 入出力要素NO : {↓
34 | value : [""],↓
35 | field : "プロセス要素no(サービス)%"↓
36 | ↓
37 | }
```

項目	詳細	サービスID	サービス名	概要
更新	更新			

## ⑧updateSQLの「新規処理」部分を作成する

「ピンク枠内を確認するとインサートになっている。青枠部分へ項目名を作成する。」

[updatesql・新規処理]

```
3  LLLLLL "comment": "サービス新規処理",
4  LLLLLL "id": "insert",
5  LLLLLL "before": "setInsertDataOfDrmTools('detail','insert','入出力要素NO')",
6  LLLLLL "after": "",
7  LLLLLL "sql": "",
8  LLLLLL "type": "insert",
9  LLLLLL "freesql": "",
10 LLLLLL "genesql": "",
11 LLLLLL "dist": "",
12 LLLLLL "from": "サービス",
13 LLLLLL "where": "",
14 LLLLLL "order": "",
15 LLLLLL "input": {
16 LLLLLL "multiline": "yes",
17 LLLLLL "record": {
18 LLLLLL "サービスID": {
19 LLLLLL "value": "",
20 LLLLLL "field": "サービスid",
21 LLLLLL "サービス名": {
22 LLLLLL "value": [""],
23 LLLLLL "概要": {
24 LLLLLL "value": [""]
25 LLLLLL "検索管理単位NO": {
26 LLLLLL "value": [""],
27 LLLLLL "field": "管理単位no",
28 LLLLLL "frontype": "request",
29 LLLLLL "fromid": "header",
30 LLLLLL "fromio": "",
31 LLLLLL "fromname": "検索管理単位NO",
32 LLLLLL "登録日時": {
33 LLLLLL "value": [""],
34 LLLLLL "funct": "CURRENT_TIMESTAMP",
35 LLLLLL "登録者id": {
36 LLLLLL "value": [""],
37 LLLLLL "frontype": "request",
38 LLLLLL "fromid": "login",
39 LLLLLL "fromio": "",
40 LLLLLL "fromname": "ユーザID",
41 LLLLLL "登録pgmid": {
42 LLLLLL "value": ["DG_170U"],
43 LLLLLL "更新日時": {
44 LLLLLL "value": [""],
45 LLLLLL "funct": "CURRENT_TIMESTAMP",
46 LLLLLL "更新者id": {
47 LLLLLL "value": [""],
48 LLLLLL "frontype": "request",
49 LLLLLL "fromid": "login",
50 LLLLLL "fromio": "",
51 LLLLLL "fromname": "ユーザID",
52 LLLLLL "更新pgmid": {
53 LLLLLL "value": ["DG_170U"],
```

## Ruby

「SQLJSONのbeforeでrubyを使用する場合はserverのrubyを修正する。

プログラム名 [DG\_170U] へ変更を行いデータが流れるように作成する。」

