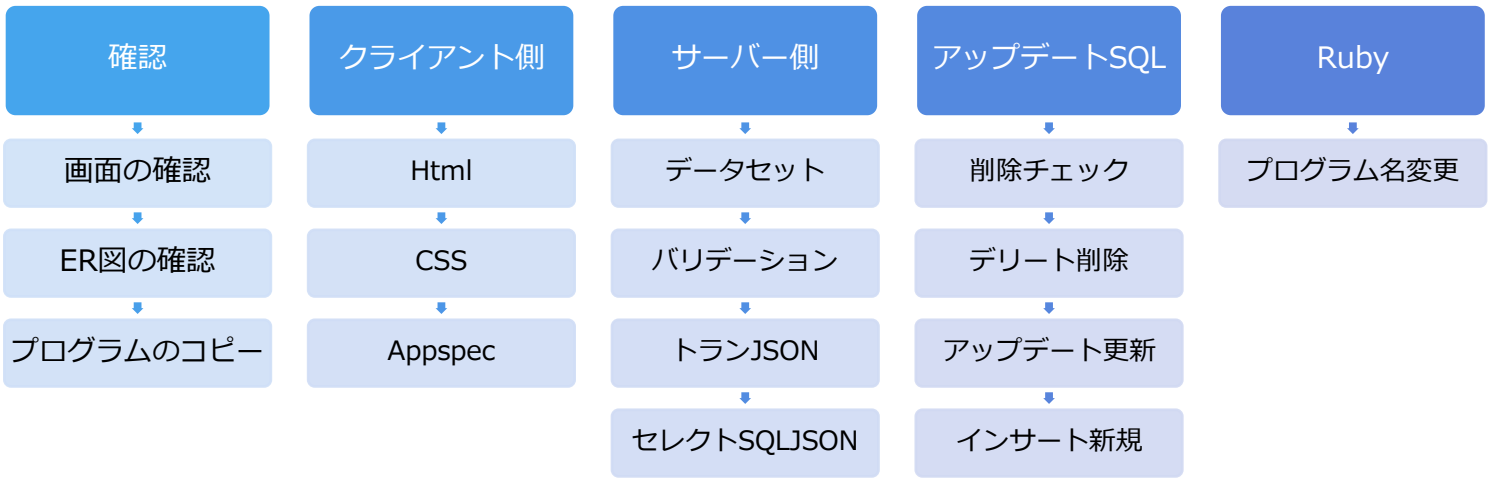
実践2『登録画面』を作成する

事前作業

手順

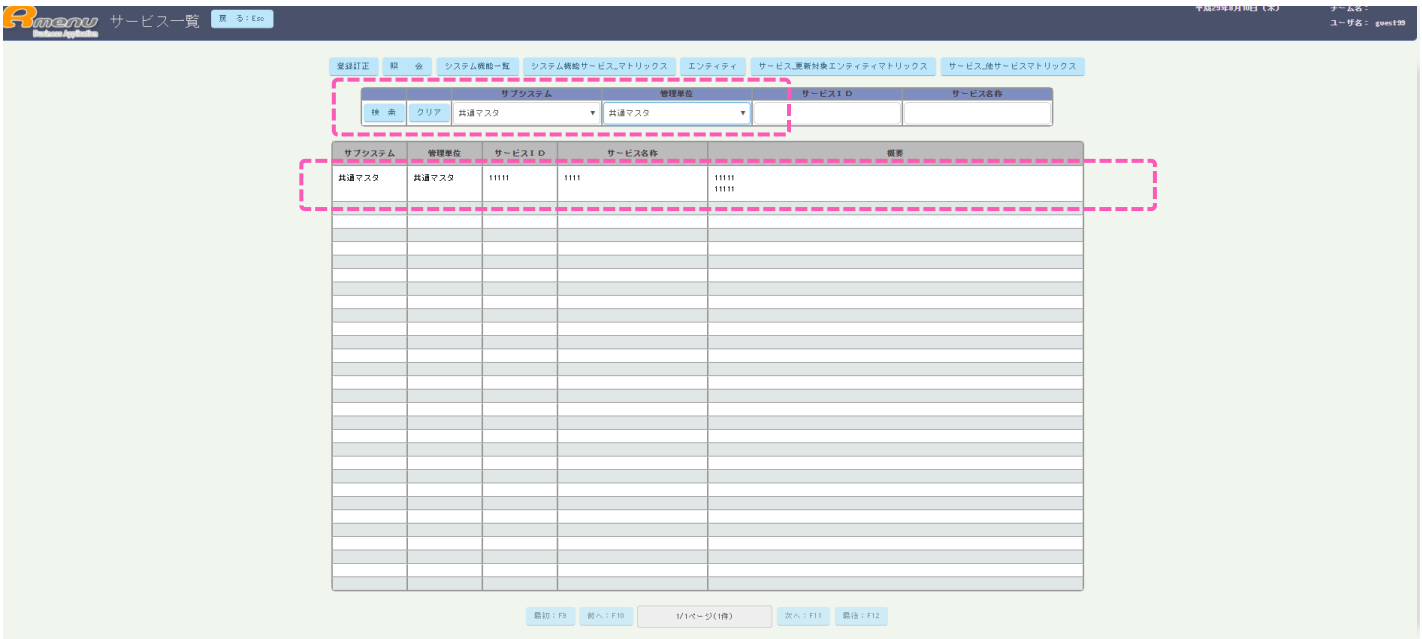


画面の確認

「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**］を押下し、『**登録・訂正画面**』へ移動する。」

①一覧画面

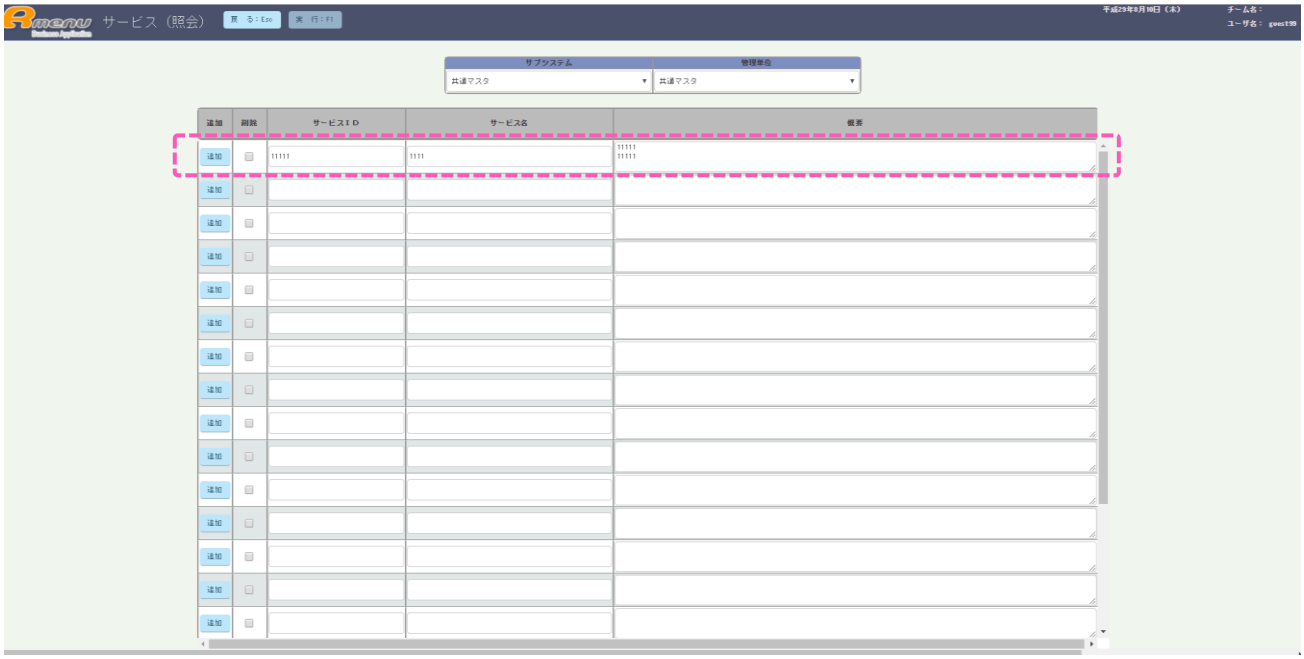


②照会画面　selectのプログラムが動きます。

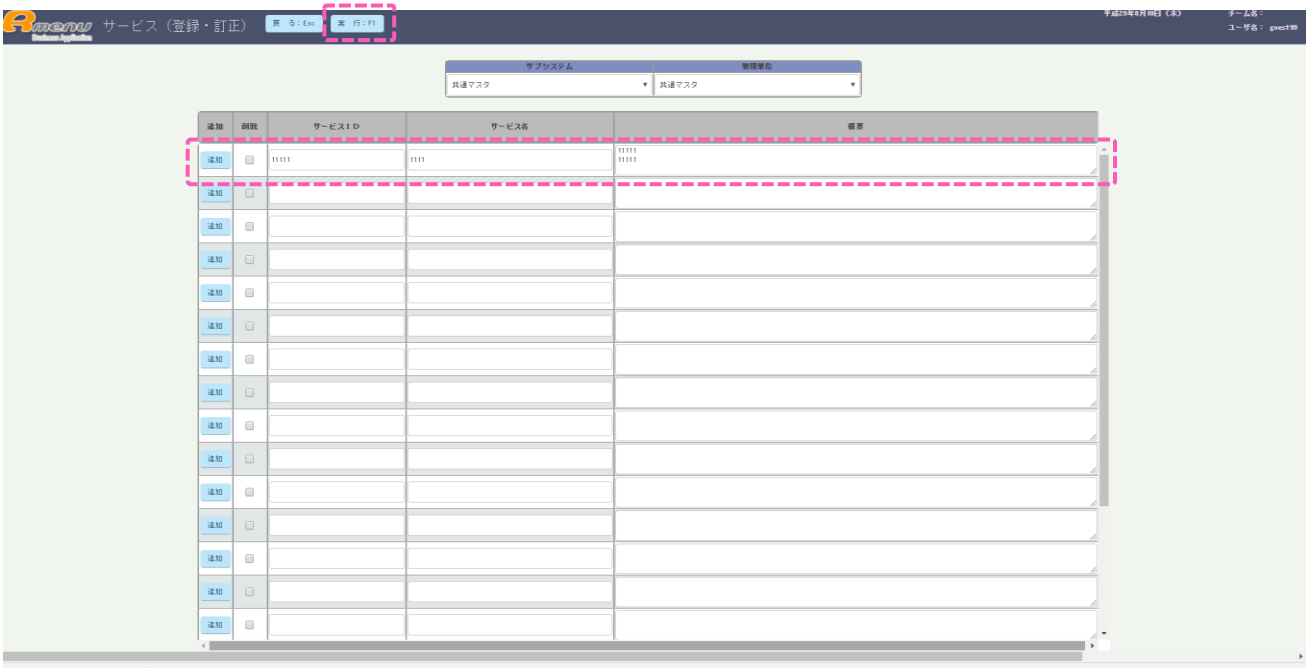
「『**登録・訂正画面**』で**新規登録・内容の変更・登録データの削除**を行うことができる。

最後に［**実行ボタン**］を押下し、入力したデータがすべて更新される仕組みになっている。」

［画面］



③登録訂正画面　updateのプログラムが動きます。



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。

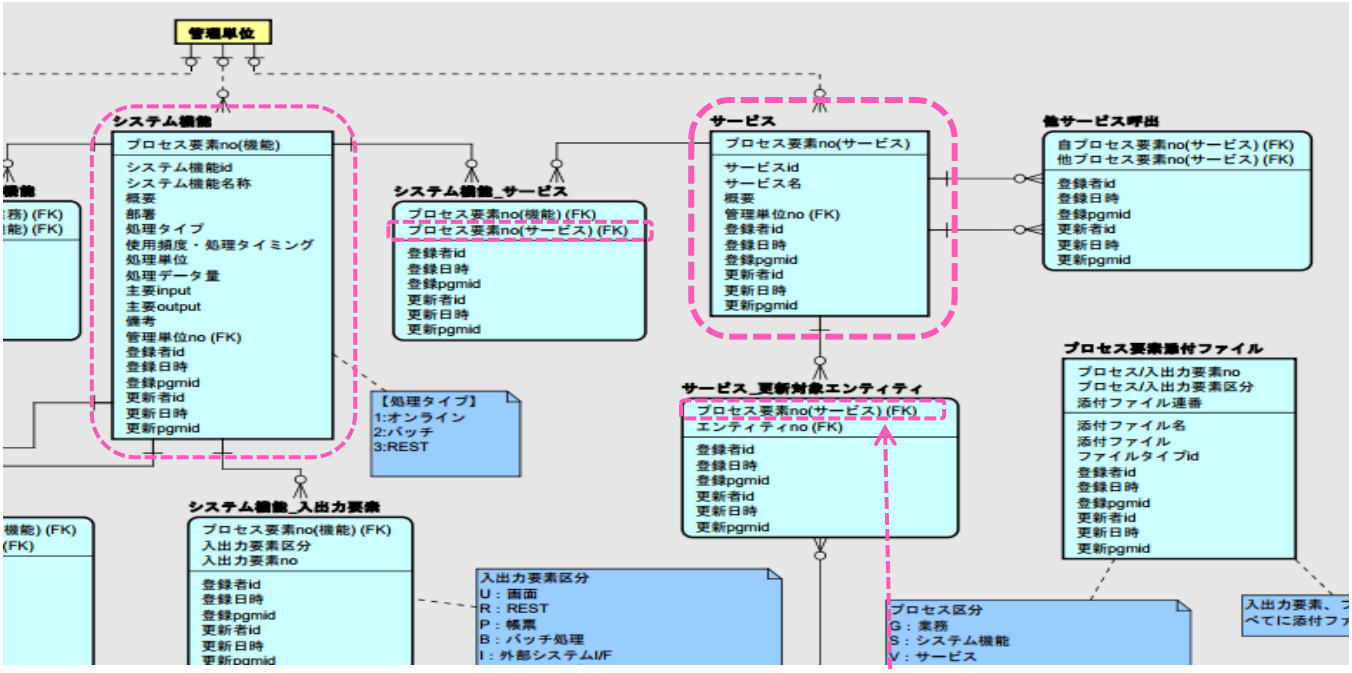
サービスで項目を削除するときに**子を残したまま削除ができないようにする**。

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

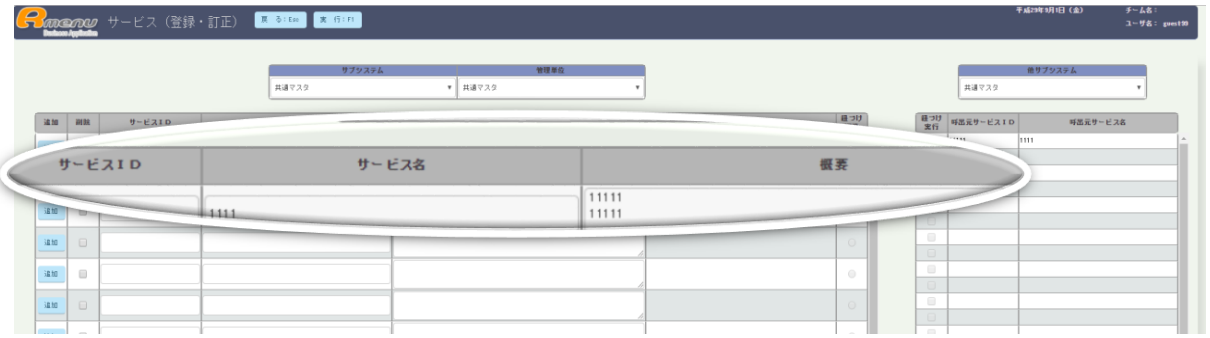
①Html

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か所づづ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

```
    <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>↓
      </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 rmenuTable">↓
  <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 rmenuLeft 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 rmenuLeft サービス名">↓
      </td>↓
      <td class="width400">↓
        <textarea name="概要" rows="3" class="form-control input-sm rfocusblue rmenuLeft 概要"></textarea>↓
      </td>↓
```

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 rmenuLeft サービスＩＤ">
```

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。

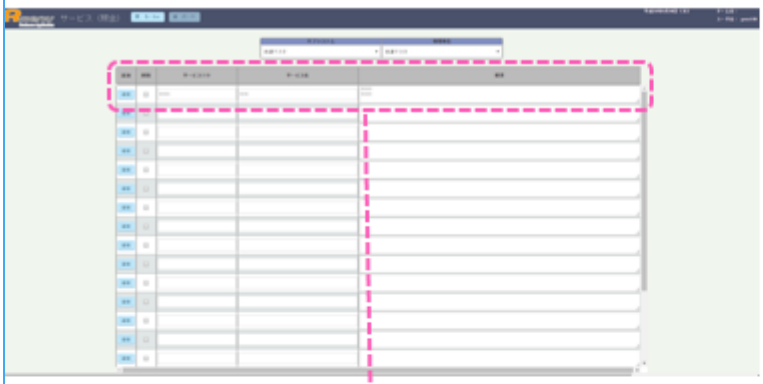

```
<td class="width400">↓
<textarea name="概要" rows="3" class="form-control input-sm rfocusblue rmenuLeft 概要"></textarea>↓
```

3-3.「**チェックBOX**」


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

4.「幅を決める」

[画面]



[Html]

```
      <table id="mainTableH1" class="table table-condensed rmenuTable">↓
       <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>↓
          </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 rmenuTable">↓
       <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 rmenuLeft 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 rmenuLeft サービス名">↓
          </td>↓
          <td class="width400">↓
            <textarea name="概要" rows="3" class="form-control input-sm rfocusblue rmenuLeft 概要"></textarea>↓
          </td>↓
        </tr>↓
```

②CSS

1.Htmlで作成した幅の合計を作成する

2. スクロールバーの17ｐｘを追加する

※詳しくは一覧画面作成に記載済

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］

```
//インスタンスプロパティを追加する
var AppSpec = App.AppSpec = new $R.Class($R.AppSpec);
AppSpec.fn.init = function(name) {
    $R.log("AppSpec init : start");

    this.name       = name;
    this.nextname   = "";
    this.beforename = "DG_140R";

    $R.log("AppSpec init : 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］

```
//前画面からの引き継ぎデータ定義
//------------------------------------------------------------
,beforeStorageData: [
    {
        datasetid: "header"
        ,dataname: ["検索サブシステムＮＯ","検索管理単位ＮＯ"]
        ,classname: ["検索サブシステム選択","検索管理単位選択"]
        ,typename: ["select","select"]
    }
```

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］

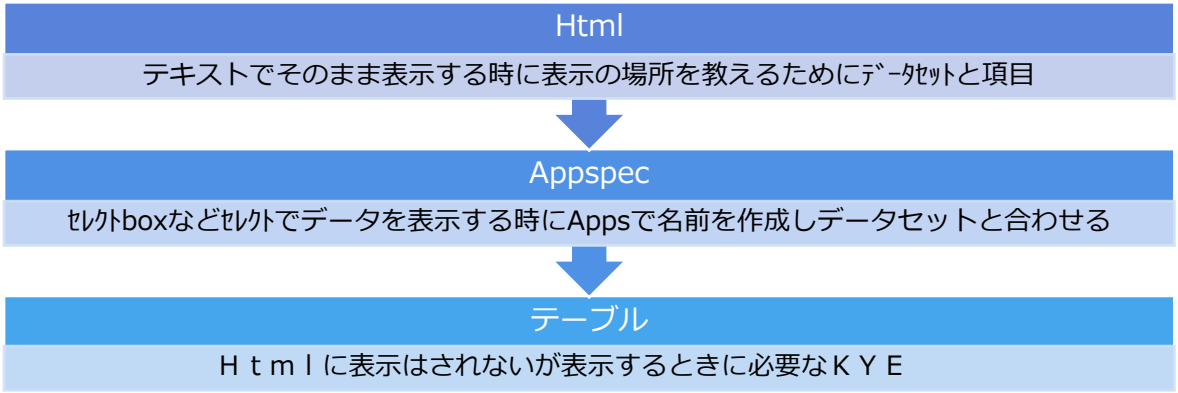
```
,selectorEvent: [
//ここから追加処理
    ["#検索サブシステム選択", "change", "onChange検索サブシステム選択"]
    ,["#検索管理単位選択", "change", "onChange検索管理単位選択"]
    ,["#検索他サブシステム選択", "change", "onChange検索他サブシステム選択"]
    ,[".削除", "change", "onChange削除"]
]
```

```
//------------------------------------------------------------------------------
//チェックボックス□定義
//selectorid    :HTMLのIDを指定する
//selectorname  :HTMLのnameを指定する（テーブル内で使用する時に定義する）
//datasetid     :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
//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
//value         :
//on            :チェックされた時に取る値を指定する
//off           :チェックが外された時に取る値を指定する
//------------------------------------------------------------------------------
,checkbox1: [
    {
        selectorid: "", selectorname: "削除", datasetid: "detail1", datasetname: "削除"
        ,value:     {on: "9", off: "0"}
    }
]
```

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」

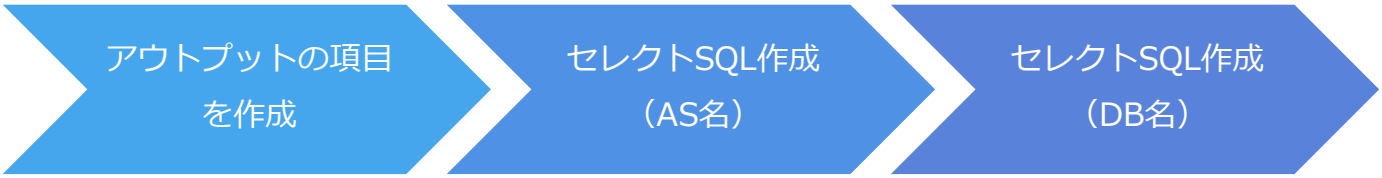
トランJSON

「データセットから項目をそろえる」

selectSQLJSON（セレクト）

「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」

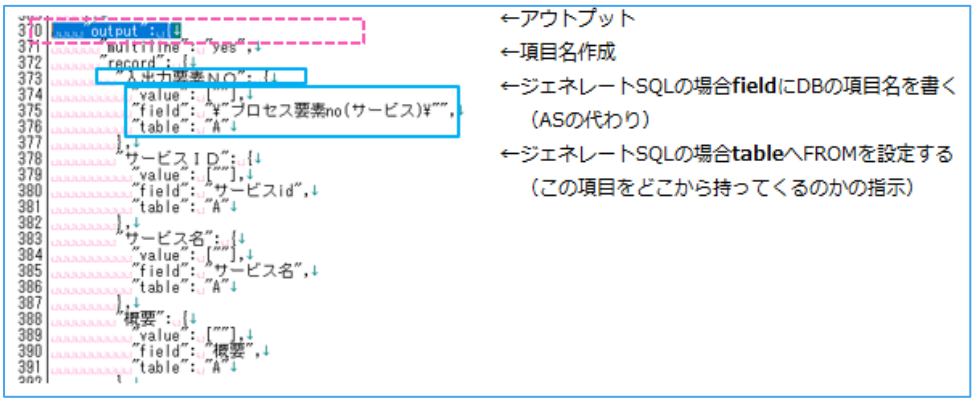
「実行後にDBのデータをセレクトで表示する」

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**］［**teble**］の作成に注意する。」

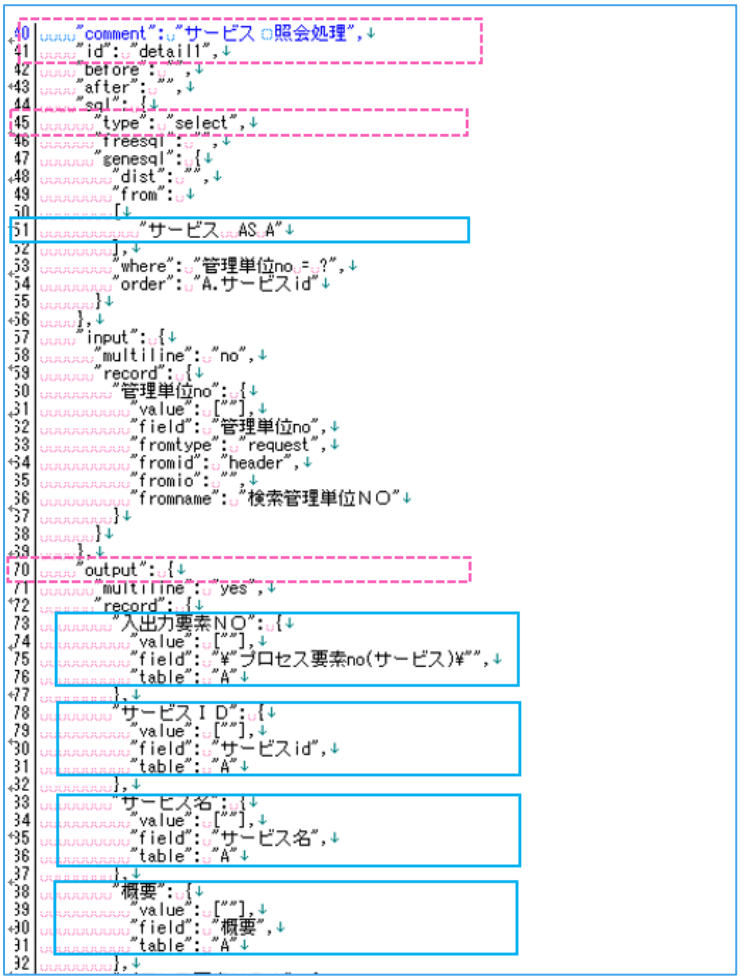
［updatesql］



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］



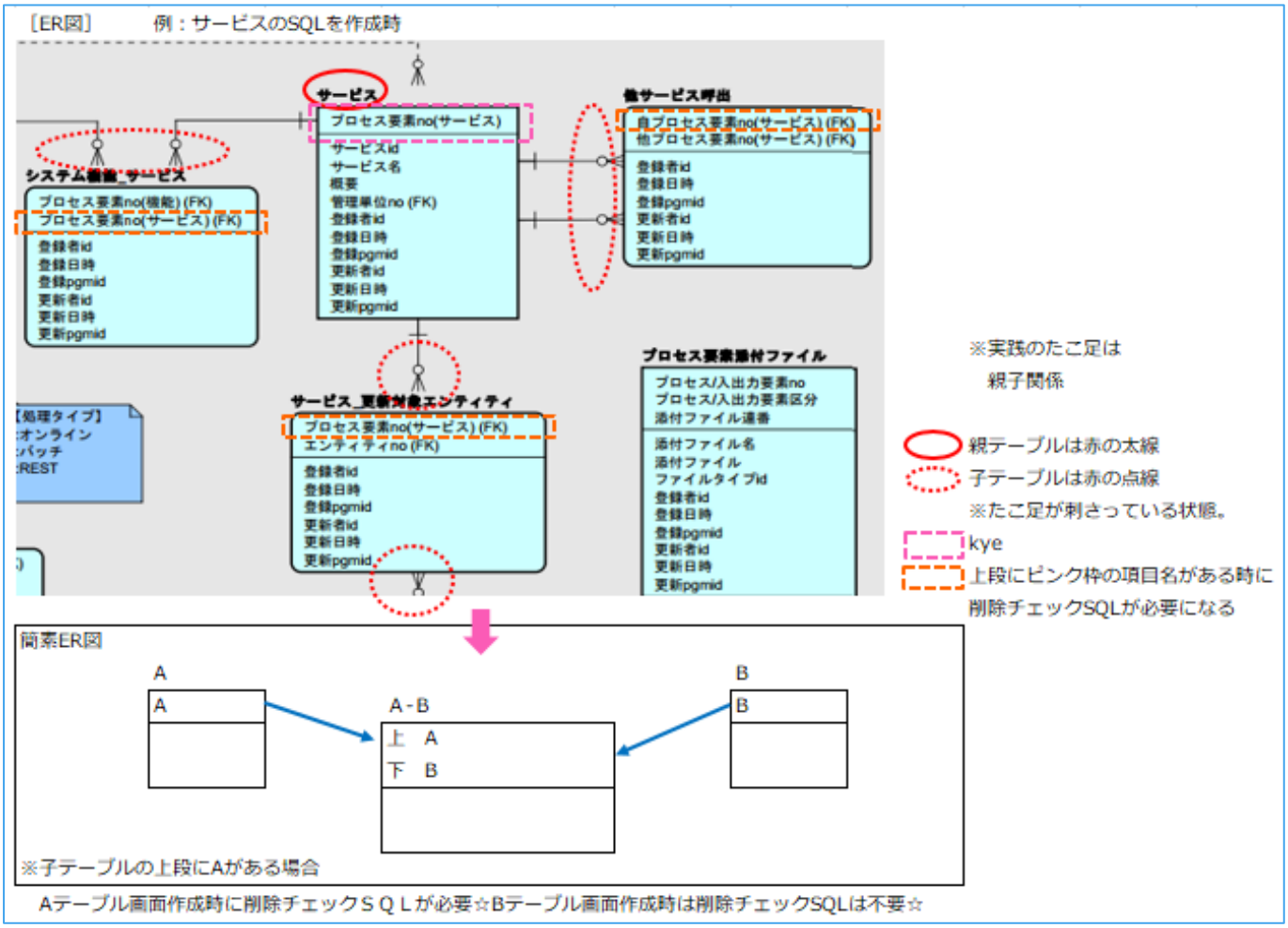
⑤updateSQLの「削除チェック」部分を作成する

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。」**

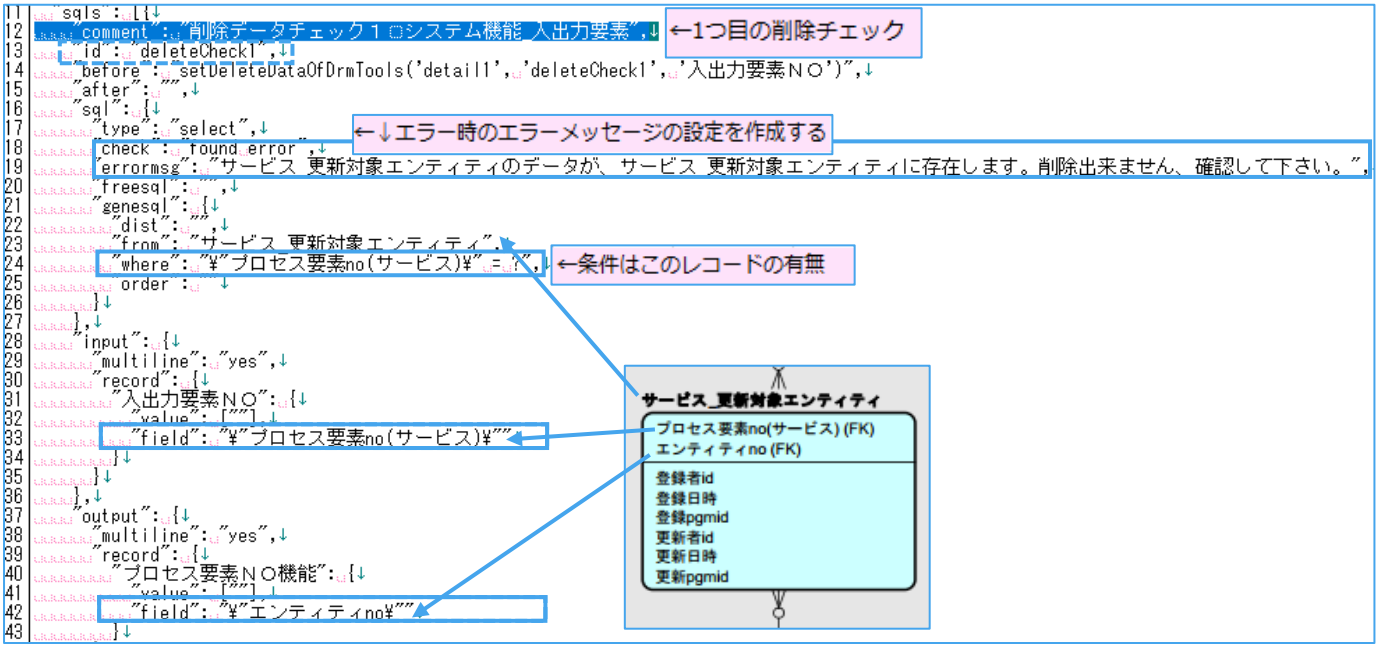
※子テーブルがあるのに親テーブルのデータを削除してしまうと

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**。」

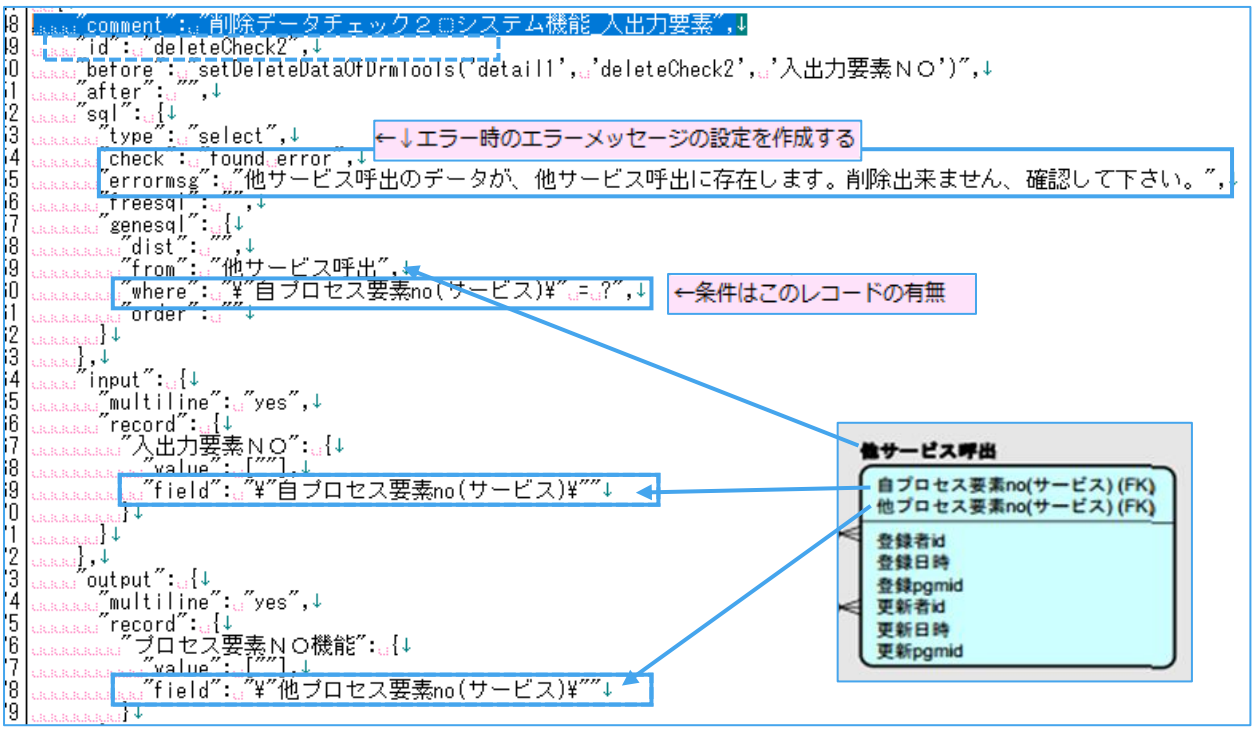
2. 「1つ目の削除チェック作成」

［updatesql］



3. 「2つ目の削除チェック作成」

［updatesql］



・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。）

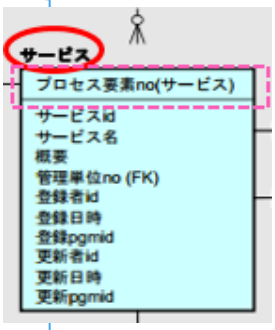
・テーブル名とレコード名を作成する（コピー元のテーブル名をこのテーブル名へ変更する。）

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］

```
"comment": "サービス□削除処理",
"id": "delete1",
"before": "setDeleteDataOfDrmTools('detail1','delete1','入出力要素ＮＯ')",
"after": "",
"sql": {
  "type": "delete",
  "freesql": "",
  "genesql": {
    "dist": "",
    "from": "サービス",
    "where": "¥"プロセス要素no(サービス)¥" = ?",
    "order": ""
  }
},
"input": {
  "multiline": "yes",
  "record": {
    "入出力要素ＮＯ": {
      "value": [""],
      "field": "¥"プロセス要素no(サービス)¥""
    }
```



⑦updateSQLの「更新処理」部分を作成する

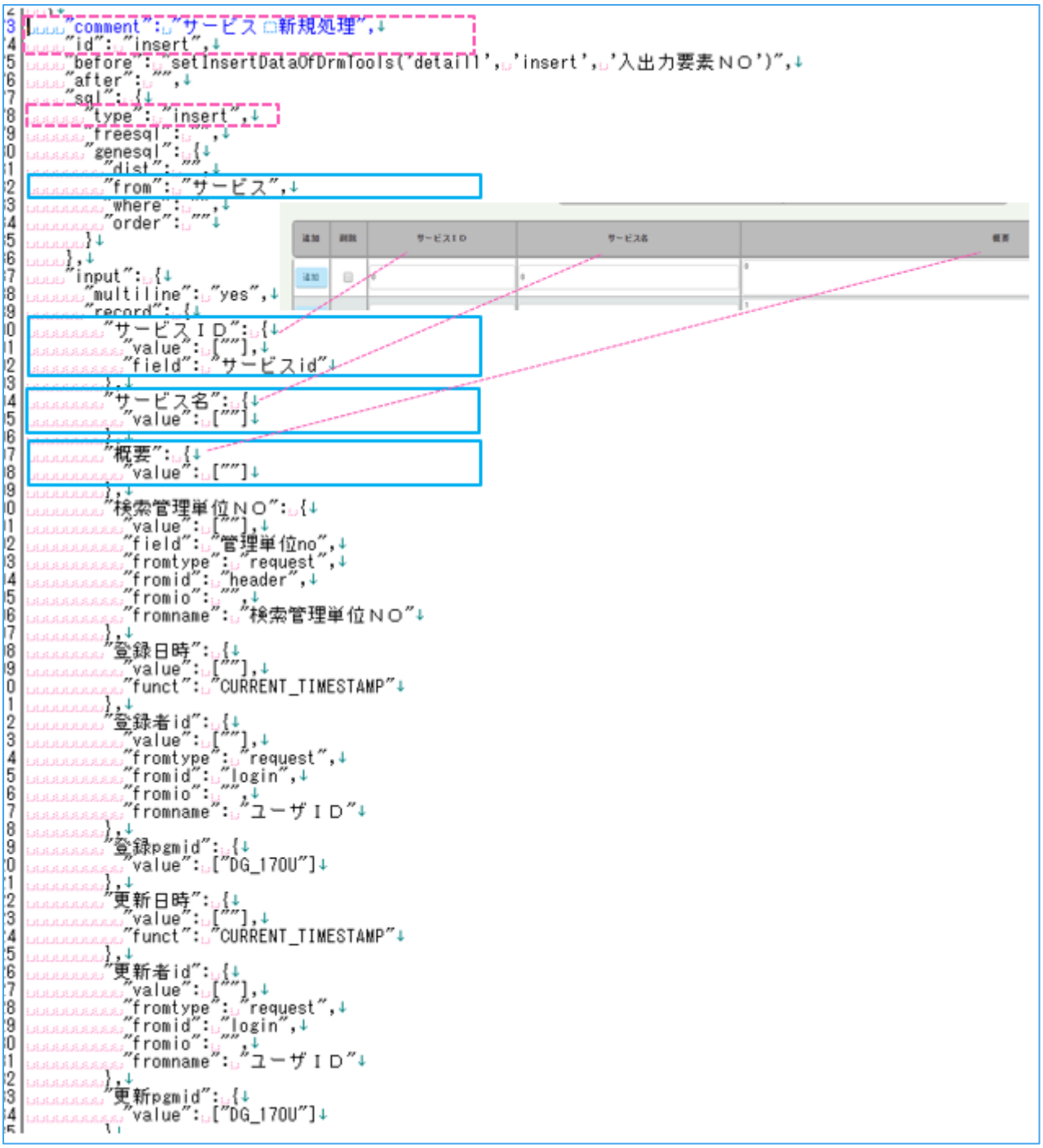
「すでに入力済の項目の**データが変更されたときの処理**。項目名を作成する。」

```
"comment": "サービス□更新処理",
"id": "update",
"before": "setUpdateDataOfDrmTools('detail1','update','入出力要素ＮＯ')",
"after": "",
"sql": {
  "type": "update",
  "freesql": "",
  "genesql": {
    "dist": "",
    "from": "サービス",
    "where": "¥"プロセス要素no(サービス)¥" = ?",
    "order": ""
  }
},
"input": {
  "multiline": "yes",
  "record": {
    "サービスＩＤ": {
      "value": [""],
      "field": "サービスid"
    },
    "サービス名": {
      "value": [""]
    },
    "概要": {
      "value": [""]
    },
    "検索管理単位ＮＯ": {
      "value": [""],
      "field": "管理単位no",
      "fromtype": "request",
      "fromid": "header",
      "fromio": "",
      "fromname": "検索管理単位ＮＯ"
    },
    "更新日時": {
      "value": [""],
      "funct": "CURRENT_TIMESTAMP"
    },
    "更新者id": {
      "value": [""],
      "fromtype": "request",
      "fromid": "login",
      "fromio": "",
      "fromname": "ユーザＩＤ"
    },
    "更新pgmid": {
      "value": ["DG_170U"]
    },
    "入出力要素ＮＯ": {
      "value": [""],
      "field": "¥"プロセス要素no(サービス)¥""
    }
```

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］



```
{
"comment":"サービス□新規処理",
"id":"insert",
"before":"setInsertDataOfDrmTools('detail','insert','入出力要素ＮＯ')",
"after":"",
"sql":{
"type":"insert",
"freesql":"",
"genesql":{
"dist":"",
"from":"サービス",
"where":"",
"order":""
}
},
"input":{
"multiline":"yes",
"record":{
"サービスＩＤ":{
"value":[""],
"field":"サービスid"
},
"サービス名":{
"value":[""]
},
"概要":{
"value":[""]
},
"検索管理単位ＮＯ":{
"value":[""],
"field":"管理単位no",
"fromtype":"request",
"fromid":"header",
"fromio":"",
"fromname":"検索管理単位ＮＯ"
},
"登録日時":{
"value":[""],
"funct":"CURRENT_TIMESTAMP"
},
"登録者id":{
"value":[""],
"fromtype":"request",
"fromid":"login",
"fromio":"",
"fromname":"ユーザＩＤ"
},
"登録pgmid":{
"value":["DG_170U"]
},
"更新日時":{
"value":[""],
"funct":"CURRENT_TIMESTAMP"
},
"更新者id":{
"value":[""],
"fromtype":"request",
"fromid":"login",
"fromio":"",
"fromname":"ユーザＩＤ"
},
"更新pgmid":{
"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。」