# DESCRIÇÃO

A tabela "regras\_pos\_calculo" armazena regras de negócio que são aplicadas automaticamente após o cálculo de TPE, permitindo ajustes dinâmicos nos resultados conforme condições específicas.

Essas regras permitem alterar chaves de controle, modificar tempos, adicionar ou remover operações baseado em condições lógicas complexas.

* Define regras de ajuste pós-cálculo
* Suporta condições simples e múltiplas com operadores lógicos (E/OU)
* Permite ações diversas (alterar chave, modificar tempo, adicionar operação)
* Automatiza lógicas de negócio complexas
* Mantém histórico de regras aplicadas

# OBJETIVO

* Automatizar ajustes pós-cálculo sem alteração de código
* Permitir regras de negócio configuráveis
* Suportar condições complexas com múltiplos parâmetros
* Integrar com operações e resultados de cálculo
* Facilitar manutenção de lógicas de negócio

# LÓGICA TÉCNICA DE AQUISIÇÃO DE DADOS

**Fontes**

* Interface administrativa do widget (criação/edição de parâmetros);
* Chamadas de API diretamente (criação/edição de parâmetros).

**Script para criar tabela**

CREATE TABLE regras\_pos\_calculo (

id INT IDENTITY(1,1) PRIMARY KEY,

nome NVARCHAR(200) NOT NULL UNIQUE,

descricao NVARCHAR(MAX),

tipo\_condicao NVARCHAR(100) NOT NULL,

condicao\_valor NVARCHAR(MAX),

condicoes\_multiplas NVARCHAR(MAX),

operador\_logico NVARCHAR(10),

acao NVARCHAR(100) NOT NULL,

acao\_valor NVARCHAR(MAX) NOT NULL,

ativo BIT DEFAULT 1,

data\_criacao DATETIME2 DEFAULT GETDATE(),

data\_modificacao DATETIME2 DEFAULT GETDATE()

);

-- Índices para performance

CREATE INDEX idx\_regras\_nome ON regras\_pos\_calculo(nome);

CREATE INDEX idx\_regras\_ativo ON regras\_pos\_calculo(ativo);

CREATE INDEX idx\_regras\_tipo\_condicao ON regras\_pos\_calculo(tipo\_condicao);

-- Constraint para validar JSON no campo condicoes\_multiplas

ALTER TABLE regras\_pos\_calculo

ADD CONSTRAINT chk\_regras\_condicoes\_json

CHECK (condicoes\_multiplas IS NULL OR ISJSON(condicoes\_multiplas) = 1);

# FILTROS

Recomendações de filtros no consumo (API/relatórios/SQL):

* **Por nome**
* **Por tipo\_condicao**
* **Por ativo (ativo/inativo)**
* **Por ação**
* **Busca textual em nome ou descrição**

# RECORRÊNCIA DE ATUALIZAÇÃO

* **Widget/API** → atualização **em tempo real** (transacional).

# CAMPO CHAVE PARA ATUALIZAÇÃO

**Chave primária**: id (INT IDENTITY)

**Chave natural (UNIQUE):** nome - garante que não existam regras com nomes duplicados

**Estratégia de operações:**

**CREATE (POST)**

- Valida se nome já existe (UNIQUE constraint)

- Insere novo registro com ID auto-gerado

**UPDATE (PUT)**

- Identificação por id enviado na URL (/orbis/regras/:id)

- Atualiza campos enviados no body

- Atualiza data\_modificacao = CURRENT\_TIMESTAMP

- Valida unicidade de nome se este campo for alterado

**DELETE (DELETE)**

- Quando usuário excluir via widget: DELETE FROM regras\_pos\_calculo WHERE id = @id

- Remove permanentemente o registro do banco

- Usado quando a regra foi criada por engano ou não é mais necessária

**DESATIVAR (PATCH)**

- Quando usuário desativar via widget: UPDATE regras\_pos\_calculo SET ativo = 0 WHERE id = @id

- Mantém o registro no banco com ativo = 0

- Usado quando a regra não deve mais aparecer, mas precisa ser mantida para histórico/auditoria

# API – DADOS DE ENTRADA

**POST /orbis/regras - Criar nova regra**

* **Descrição:** Cria uma nova regra de pós-cálculo no sistema.
* **Body (JSON):**

{

"nome": "ALTERAR\_ULTIMA\_OP",

"descricao": "Alterar chave de controle da última operação do roteiro",

"tipo\_condicao": "ultima\_operacao",

"condicao\_valor": null,

"condicoes\_multiplas": [],

"operador\_logico": "E",

"acao": "alterar\_chave\_controle",

"acao\_valor": "ZP99",

"ativo": true

}

* **Retorno 201:**

{

"success": true,

"data": {

"id": 11,

"nome": "ALTERAR\_ULTIMA\_OP",

"descricao": "Alterar chave de controle da última operação do roteiro",

"tipo\_condicao": "ultima\_operacao",

"condicao\_valor": null,

"condicoes\_multiplas": [],

"operador\_logico": "E",

"acao": "alterar\_chave\_controle",

"acao\_valor": "ZP99",

"ativo": true,

"data\_criacao": "2025-10-09T14:30:00Z",

"data\_modificacao": "2025-10-09T14:30:00Z"

}

}

**POST /orbis/regras - Criar nova regra com múltiplas condições**

* **Descrição:** Cria uma nova regra de pós-cálculo no sistema.
* **Body (JSON):**

{

"nome": "REMOVER\_OPERACAO\_DOBRA\_SIMPLES",

"descricao": "Remove operação de dobra quando qtd\_dobras <= 2 e espessura < 1.0mm",

"tipo\_condicao": "multiplas",

"condicao\_valor": null,

"condicoes\_multiplas": [

{

"parametro": "qtd\_dobras",

"operador": "<=",

"valor": "2"

},

{

"parametro": "espessura\_chapa",

"operador": "<",

"valor": "1.0"

}

],

"operador\_logico": "E",

"acao": "remover\_operacao",

"acao\_valor": "DA101",

"ativo": true

}

* **Retorno 201:**

{

"success": true,

"data": {

"id": 12,

"nome": "REMOVER\_OPERACAO\_DOBRA\_SIMPLES",

"descricao": "Remove operação de dobra quando qtd\_dobras <= 2 e espessura < 1.0mm",

"tipo\_condicao": "multiplas",

"condicao\_valor": null,

"condicoes\_multiplas": [

{

"parametro": "qtd\_dobras",

"operador": "<=",

"valor": "2"

},

{

"parametro": "espessura\_chapa",

"operador": "<",

"valor": "1.0"

}

],

"operador\_logico": "E",

"acao": "remover\_operacao",

"acao\_valor": "DA101",

"ativo": true,

"data\_criacao": "2025-10-09T14:35:00Z",

"data\_modificacao": "2025-10-09T14:35:00Z"

}

}

**PUT /orbis/regras/:id - Atualizar regra existente**

* **Descrição:** Atualiza dados de uma regra existente identificada pelo ID
* **URL Parameter:** id (obrigatório)
* **Body (JSON):**

{

"descricao": "Alterar chave de controle da última operação (atualizada)",

"acao\_valor": "ZP98"

}

* **Retorno 200:**

{

"success": true,

"data": {

"id": 11,

"nome": "ALTERAR\_ULTIMA\_OP",

"descricao": "Alterar chave de controle da última operação (atualizada)",

"tipo\_condicao": "ultima\_operacao",

"condicao\_valor": null,

"condicoes\_multiplas": [],

"operador\_logico": "E",

"acao": "alterar\_chave\_controle",

"acao\_valor": "ZP98",

"ativo": true,

"data\_criacao": "2025-10-09T14:30:00Z",

"data\_modificacao": "2025-10-09T16:10:00Z"

}

}

* **Retorno** **404**: {"success": false, "error": "Regra não encontrada"}
* **Retorno** **409**: {"success": false, "error": "Já existe uma regra com este nome"}

**PATCH /orbis/regras/:id/desativar - Desativar regra**

* **Descrição:** Soft delete - marca ativo = 0 (mantém registro no banco)
* **URL Parameter:** id (obrigatório)
* **Retorno 200:**

{

"success": true,

"message": "Regra desativada com sucesso",

"data": {

"id": 11,

"ativo": false

}

}

**PATCH /orbis/regras/:id/ativar - Reativar regra**

* **Descrição:** Marca ativo = 1 (reativa tabela desativada)
* **URL Parameter:** id (obrigatório)
* **Retorno 200:**

{

"success": true,

"message": "Regra ativada com sucesso",

"data": {

"id": 15,

"ativo": true

}

}

**DELETE /orbis/regras/:id - Excluir regra permanentemente**

* **Descrição:** DELETE físico - remove registro do banco definitivamente
* **URL Parameter:** id (obrigatório)
* **Retorno 200:**

{

"success": true,

"message": "Regra excluída permanentemente"

}

* **Retorno 404:** {"success": false, "error": "Regras não encontrada"}

# API – DADOS DE SAÍDA

**GET /orbis/regras – Listar regras**

* **Descrição**: Retorna lista de todas as regras
* **Parâmetros**: filtros opcionais
* **Retorno**: coleção de tabelas de coeficientes.

**GET /orbis/regras/:id – Buscar regra por ID**

* **Descrição**: Retorna uma regra específica pelo ID
* **Retorno** **200**: Objeto único de regra
* **Retorno** **404**: {"success": false, "error": "Regra não encontrada"}

**GET /orbis/regras/nome/:nome – Buscar regra por nome**

* **Descrição**: Retorna uma regra específica pelo nome
* **Retorno** **200**: Objeto único de regra
* **Retorno** **404**: {"success": false, "error": "Regra não encontrada"}

# CATEGORIZAÇÃO DO DHIVE

# ÁREAS IMPACTADAS PELA INFORMAÇÃO

Engenharia de Processos

# VALOR AGREGADO PARA A COMPANHIA

* **Automação**: Elimina necessidade de intervenção manual em ajustes pós-cálculo
* **Flexibilidade**: Permite criar regras complexas sem alteração de código
* **Rastreabilidade**: Histórico completo de alterações via data\_criacao/modificacao
* **Padronização**: Garante aplicação consistente de lógicas de negócio
* **Manutenibilidade**: Fácil criação e edição de regras pela engenharia
* **Qualidade**: Reduz erros humanos em ajustes manuais
* **Migração** **para** **3DEXPERIENCE**: Remove dependência de lógicas hardcoded
* **Agilidade**: Mudanças de regras sem necessidade de deploy

# INTEGRAÇÃO COM SISTEMA DE AUDITORIA

Todas as operações de CREATE, UPDATE, DELETE, ACTIVATE e DEACTIVATE nesta tabela DEVEM registrar logs na tabela "logs\_auditoria".

1. RECEBER user\_id no Body

Todas as requisições POST, PUT, PATCH e DELETE devem receber o campo user\_id:

Body exemplo:

{

"nome": "...",

"descricao": "...",

"user\_id": "eduardo.silva" ← OBRIGATÓRIO

}

2. CHAMAR Função de Log

Após cada operação bem-sucedida, chamar:

registrarLog(

tabela: 'nome\_da\_tabela',

registro\_id: id\_do\_registro,

operacao: 'CREATE' | 'UPDATE' | 'DELETE' | 'ACTIVATE' | 'DEACTIVATE',

valores\_anteriores: objeto\_com\_valores\_antigos (NULL para CREATE),

valores\_novos: objeto\_com\_valores\_novos (NULL para DELETE),

user\_id: user\_id\_recebido\_no\_body

);

3. EXEMPLOS POR OPERAÇÃO

**CREATE** (POST):

* Capturar user\_id do body
* Executar INSERT na tabela
* Chamar: registrarLog('tabela', novo\_id, 'CREATE', null, dados\_inseridos, user\_id)

**UPDATE** (PUT):

* Capturar user\_id do body
* Buscar valores anteriores: SELECT \* FROM tabela WHERE id = @id
* Executar UPDATE na tabela
* Buscar valores novos: SELECT \* FROM tabela WHERE id = @id
* Chamar: registrarLog('tabela', id, 'UPDATE', valores\_anteriores, valores\_novos, user\_id)

**DELETE** (DELETE):

* Capturar user\_id do body
* Buscar valores anteriores: SELECT \* FROM tabela WHERE id = @id
* Executar DELETE na tabela
* Chamar: registrarLog('tabela', id, 'DELETE', valores\_anteriores, null, user\_id)

**ACTIVATE** (PATCH /:id/ativar):

* Capturar user\_id do body
* Buscar valor anterior: SELECT ativo FROM tabela WHERE id = @id
* Executar: UPDATE tabela SET ativo = 1 WHERE id = @id
* Chamar: registrarLog('tabela', id, 'ACTIVATE', {ativo: 0}, {ativo: 1}, user\_id)

**DEACTIVATE (PATCH /:id/desativar):**

* Capturar user\_id do body
* Buscar valor anterior: SELECT ativo FROM tabela WHERE id = @id
* Executar: UPDATE tabela SET ativo = 0 WHERE id = @id
* Chamar: registrarLog('tabela', id, 'DEACTIVATE', {ativo: 1}, {ativo: 0}, user\_id)