

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MASTERING RELATIONAL AND NON-RELATIONAL DATABASE



Econo Med⁺
Medicina Acessível

RM: 98266 / Mel Maia Rodrigues

RM: 98078 / Augusto Barcelos Barros

RM: 98570 / Gabriel Souza de Queiroz

RM: 551629 / Gabriela Zanotto Rodrigues

RM: 97707 / Lucas Pinheiro de Melo

Problemática

A elevada carga de despesas enfrentada pelas operadoras de planos de saúde e seus beneficiários é uma preocupação relevante. Atualmente, o valor do plano de seguro de vida depende predominantemente da faixa etária do beneficiário, resultando em disparidades, com os indivíduos mais jovens pagando menos e os mais idosos pagando mais. Essa prática reflete a percepção das empresas de saúde de que os indivíduos mais velhos apresentam mais dificuldades de saúde, embora não sejam exclusivamente eles os principais usuários dos serviços de saúde.

Além do valor inicial estabelecido em contrato, há os reajustes anuais, aplicados no aniversário da contratação do plano. Esses reajustes variam de acordo com cada plano e incluem o aumento por sinistralidade, que ocorre quando o número de procedimentos e atendimentos cobertos supera as expectativas da operadora em um determinado período.

A verdadeira razão por trás dos altos custos dos planos de saúde está associada a diversos fatores. A mão de obra na área da medicina é cara, devido aos altos custos de formação e especialização dos profissionais. Além disso, há os gastos com manutenção de equipamentos e reposição de materiais, que também representam um ônus significativo para os hospitais e clínicas.

Esses fatores, somados à complexidade dos procedimentos médicos e aos avanços tecnológicos constantes na área da saúde, contribuem para o encarecimento dos planos de saúde. As operadoras custeiam cada consulta realizada pelo seu beneficiário, podendo ser uma consulta mais cara ou mais barata, dependendo do hospital ao qual o beneficiário recorre.

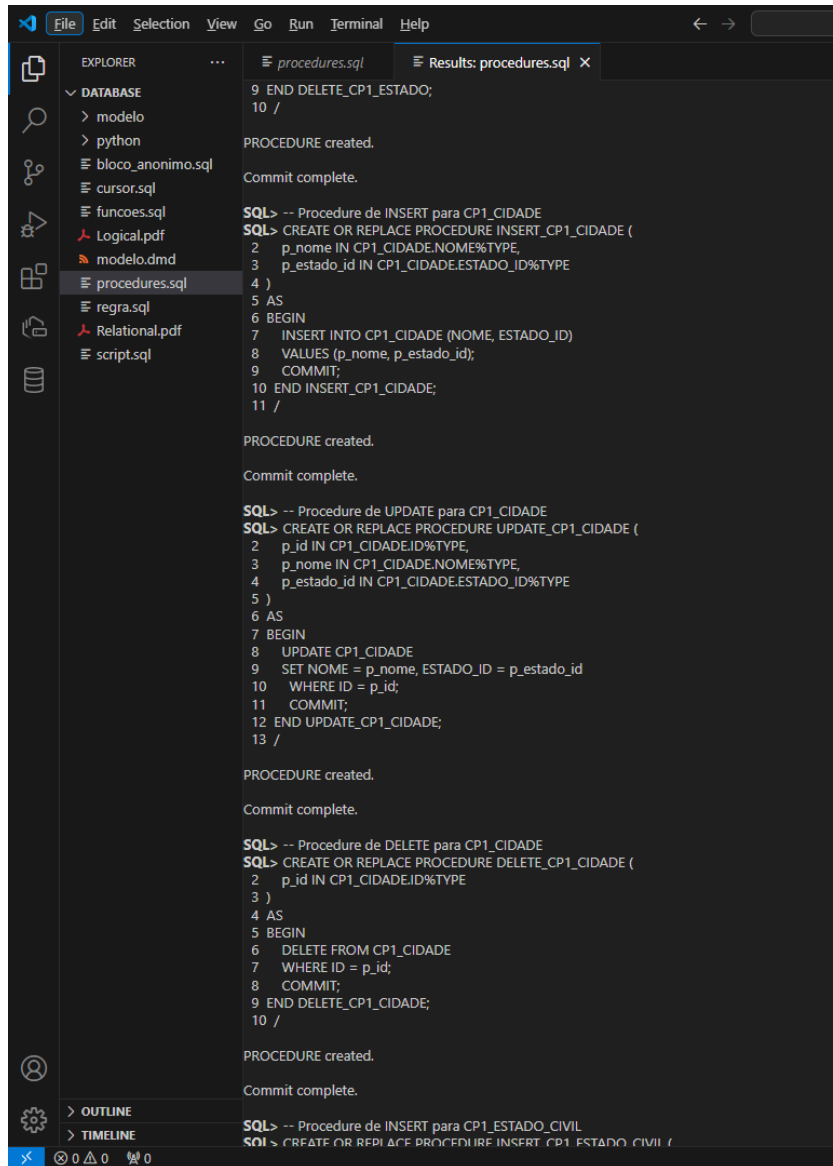
Objetivo

O objetivo principal desta solução é fornecer aos clientes um atendimento de saúde que seja não apenas eficaz em termos de custo, mas também personalizado e de alta qualidade, visando reduzir os gastos para a operadora de saúde. A abordagem proposta baseia-se inteiramente em análises avançadas de dados, permitindo que o software realize uma caracterização detalhada do perfil de cada beneficiário, utilizando tanto informações já disponíveis quanto dados externos selecionados cuidadosamente. Esse perfil detalhado possibilita direcionar o beneficiário para o tipo de atendimento mais vantajoso em termos de custo, encaminhando-o para clínicas ou hospitais especializados na área correspondente à sua necessidade de saúde, mas que pratiquem preços mais acessíveis para a operadora.

No processo de análise do perfil do beneficiário, diversas informações relevantes são consideradas, como dados demográficos, histórico completo de utilização dos serviços de saúde, mapeamento dos melhores prestadores na região onde o beneficiário se encontra, consulta ao dicionário de Classificação Internacional de Doenças (CID) para uma compreensão precisa das condições de saúde do paciente, e uma base de dados abrangente dos prestadores de saúde especializados no tratamento de comorbidades específicas. Com esses dados, a operadora é capaz de identificar não apenas o tipo de paciente, mas também suas tendências e necessidades de saúde mais frequentes, permitindo assim a seleção de hospitais ou clínicas próximos que ofereçam um custo por paciente mais baixo, sem comprometer a qualidade do atendimento necessário.

Essa abordagem estratégica não apenas proporciona economia financeira à operadora, mas também garante que os beneficiários recebam o cuidado mais adequado às suas condições, resultando em uma experiência de saúde mais satisfatória e eficaz.

Fotos da Procedure Banco de Dados:



```
9 END DELETE_CP1_ESTADO;
10 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de INSERT para CP1_CIDADE
SQL> CREATE OR REPLACE PROCEDURE INSERT_CP1_CIDADE (
2   p_nome IN CP1_CIDADE.NOME%TYPE,
3   p_estado_id IN CP1_CIDADE.ESTADO_ID%TYPE
4 )
5 AS
6 BEGIN
7   INSERT INTO CP1_CIDADE (NOME, ESTADO_ID)
8   VALUES (p_nome, p_estado_id);
9   COMMIT;
10 END INSERT_CP1_CIDADE;
11 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de UPDATE para CP1_CIDADE
SQL> CREATE OR REPLACE PROCEDURE UPDATE_CP1_CIDADE (
2   p_id IN CP1_CIDADE.ID%TYPE,
3   p_nome IN CP1_CIDADE.NOME%TYPE,
4   p_estado_id IN CP1_CIDADE.ESTADO_ID%TYPE
5 )
6 AS
7 BEGIN
8   UPDATE CP1_CIDADE
9   SET NOME = p_nome, ESTADO_ID = p_estado_id
10  WHERE ID = p_id;
11  COMMIT;
12 END UPDATE_CP1_CIDADE;
13 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de DELETE para CP1_CIDADE
SQL> CREATE OR REPLACE PROCEDURE DELETE_CP1_CIDADE (
2   p_id IN CP1_CIDADE.ID%TYPE
3 )
4 AS
5 BEGIN
6   DELETE FROM CP1_CIDADE
7   WHERE ID = p_id;
8   COMMIT;
9 END DELETE_CP1_CIDADE;
10 /

PROCEDURE created.

Commit complete.

> OUTLINE
SQL> -- Procedure de INSERT para CP1_ESTADO_CIVIL
SQL> CREATE OR REPLACE PROCEDURE INSERT_CP1_ESTADO_CIVIL (
```

The screenshot shows a SQL IDE with a dark theme. The Explorer panel on the left lists files under a 'DATABASE' folder: modelo, python, bloco_anonimo.sql, cursor.sql, funcoes.sql, Logical.pdf, modelo.dmd, procedures.sql (selected), regra.sql, Relational.pdf, and script.sql. The main editor displays the SQL code for three procedures. The first procedure, INSERT_CP1_ESTADO, is created and committed successfully. The second procedure, UPDATE_CP1_ESTADO, is also created and committed successfully. The third procedure, INSERT_CP1_CIDADE, is partially visible at the bottom of the screen.

```
SQL> -- Procedure de INSERT para CP1_ESTADO
SQL> CREATE OR REPLACE PROCEDURE INSERT_CP1_ESTADO (
2   p_nome IN CP1_ESTADO.NOME%TYPE
3 )
4 AS
5 BEGIN
6   INSERT INTO CP1_ESTADO (NOME)
7   VALUES (p_nome);
8   COMMIT;
9 END INSERT_CP1_ESTADO;
10 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de UPDATE para CP1_ESTADO
SQL> CREATE OR REPLACE PROCEDURE UPDATE_CP1_ESTADO (
2   p_id IN CP1_ESTADO.ID%TYPE,
3   p_nome IN CP1_ESTADO.NOME%TYPE
4 )
5 AS
6 BEGIN
7   UPDATE CP1_ESTADO
8   SET NOME = p_nome
9   WHERE ID = p_id;
10  COMMIT;
11 END UPDATE_CP1_ESTADO;
12 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de DELETE para CP1_ESTADO
SQL> CREATE OR REPLACE PROCEDURE DELETE_CP1_ESTADO (
2   p_id IN CP1_ESTADO.ID%TYPE
3 )
4 AS
5 BEGIN
6   DELETE FROM CP1_ESTADO
7   WHERE ID = p_id;
8   COMMIT;
9 END DELETE_CP1_ESTADO;
10 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de INSERT para CP1_CIDADE
SQL> CREATE OR REPLACE PROCEDURE INSERT_CP1_CIDADE (
2   p_nome IN CP1_CIDADE.NOME%TYPE,
3   p_estado_id IN CP1_CIDADE.ESTADO_ID%TYPE
4 )
5 AS
6 BEGIN
7   INSERT INTO CP1_CIDADE (NOME, ESTADO_ID)
8   VALUES (p_nome, p_estado_id);
9   COMMIT;
10 END INSERT_CP1_CIDADE;
```

```
File Edit Selection View Go Run Terminal Help
EXPLORER
  DATABASE
    > modelo
    > python
    bloco_anonimo.sql
    cursor.sql
    funcoes.sql
    Logical.pdf
    modelo.dmd
    procedures.sql
    regra.sql
    Relational.pdf
    script.sql
  OUTLINE
  TIMELINE
  0 0 0 0 0 0

procedures.sql
Results: procedures.sql x

PROCEDURE created.

Commit complete.

SQL> -- Procedure de INSERT para CP1_ESTADO_CIVIL
SQL> CREATE OR REPLACE PROCEDURE INSERT_CP1_ESTADO_CIVIL (
2   p_nome IN CP1_ESTADO_CIVIL.NOME%TYPE
3 )
4 AS
5 BEGIN
6   INSERT INTO CP1_ESTADO_CIVIL (NOME)
7   VALUES (p_nome);
8   COMMIT;
9 END INSERT_CP1_ESTADO_CIVIL;
10 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de UPDATE para CP1_ESTADO_CIVIL
SQL> CREATE OR REPLACE PROCEDURE UPDATE_CP1_ESTADO_CIVIL (
2   p_id IN CP1_ESTADO_CIVIL.ID%TYPE,
3   p_nome IN CP1_ESTADO_CIVIL.NOME%TYPE
4 )
5 AS
6 BEGIN
7   UPDATE CP1_ESTADO_CIVIL
8   SET NOME = p_nome
9   WHERE ID = p_id;
10  COMMIT;
11 END UPDATE_CP1_ESTADO_CIVIL;
12 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de DELETE para CP1_ESTADO_CIVIL
SQL> CREATE OR REPLACE PROCEDURE DELETE_CP1_ESTADO_CIVIL (
2   p_id IN CP1_ESTADO_CIVIL.ID%TYPE
3 )
4 AS
5 BEGIN
6   DELETE FROM CP1_ESTADO_CIVIL
7   WHERE ID = p_id;
8   COMMIT;
9 END DELETE_CP1_ESTADO_CIVIL;
10 /

PROCEDURE created.

Commit complete.

SQL> -- Procedure de INSERT para CP1_CONVENIO
SQL> CREATE OR REPLACE PROCEDURE INSERT_CP1_CONVENIO (
2   p_nome IN CP1_CONVENIO.NOME%TYPE
3 )
4 AS
5 BEGIN
6   INSERT INTO CP1_CONVENIO (NOME)
```

Nosso conjunto de procedures SQL foi desenvolvido para gerenciar dados em diversas tabelas de um sistema de saúde. As procedures permitem realizar operações de inserção, atualização e exclusão de registros de maneira eficiente e segura.

Fotos do Dataset:

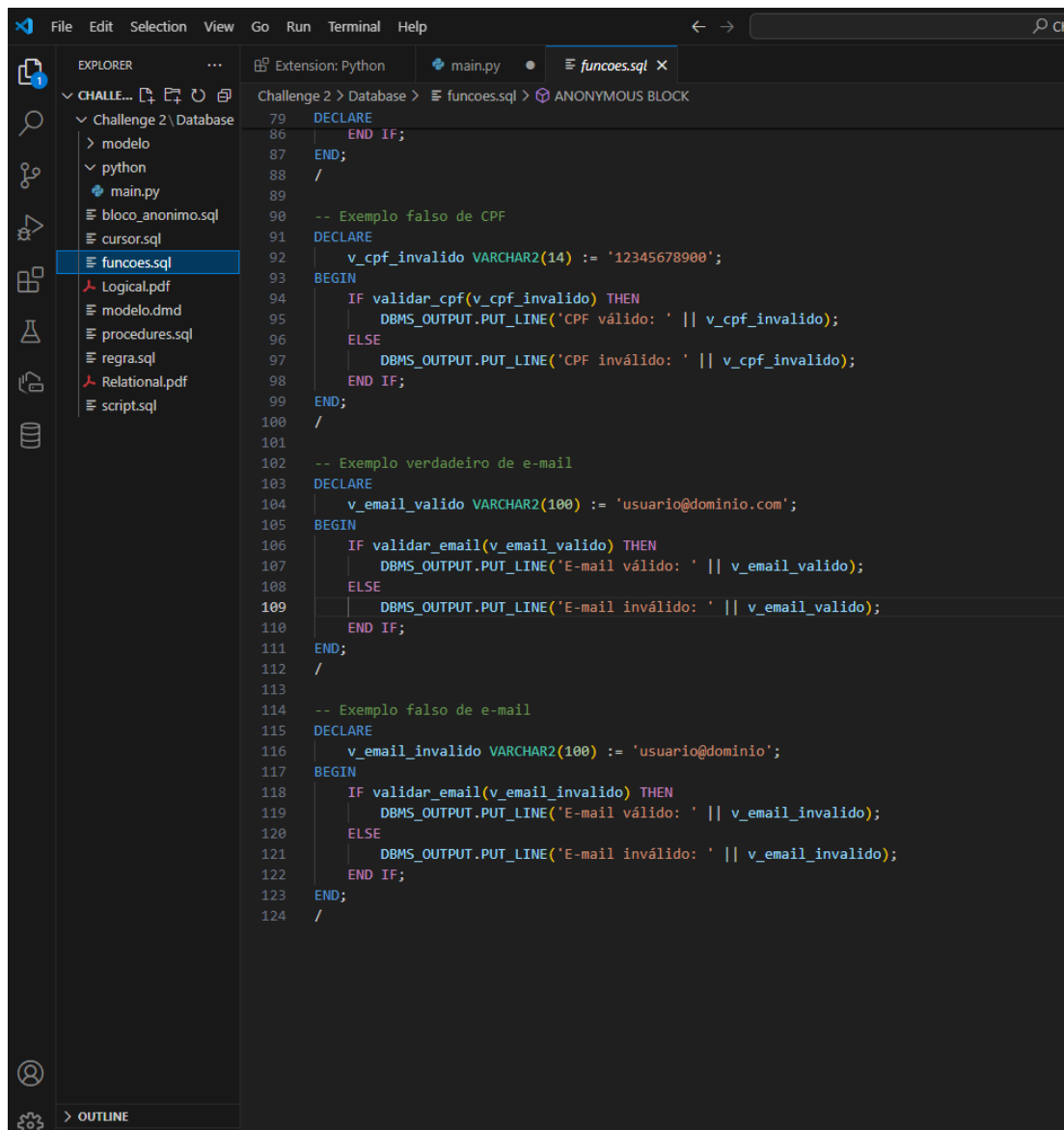
```
6
7 USERNAME = "rm98078"
8 PASSWORD = "261202"
9 DSN = "oracle.fiap.com.br:1521/ORCL"
10
11
12 def call_procedure(cursor, procedure_name, params):
13     try:
14         print(f"\nChamando a procedure {procedure_name} com parâmetros: {params}")
15         result = cursor.callproc(procedure_name, params)
16         print(f"Sucesso ao chamar a procedure {procedure_name}. Resultado: {result}")
17     except cx_Oracle.DatabaseError as e:
18         (error,) = e.args
19         print(f"Erro ao chamar a procedure {procedure_name}: {error.message}")
20
21
22 connection = cx_Oracle.connect(USERNAME, PASSWORD, DSN)
23 cursor = connection.cursor()
24
25 # ----- Chamada das procedures ----- #
26 #                                     #
27 #                                     #
28
29 call_procedure(cursor, "INSERT_CP1_ESTADO", ["São Paulo"])
30
31 call_procedure(cursor, "UPDATE_CP1_ESTADO", [1, "Rio de Janeiro"])
32
33 call_procedure(cursor, "DELETE_CP1_ESTADO", [6])
34
35 call_procedure(cursor, "INSERT_CP1_CIDADE", ["Campinas", 1])
36
37 call_procedure(cursor, "UPDATE_CP1_CIDADE", [1, "Santos", 1])
38
39 call_procedure(cursor, "DELETE_CP1_CIDADE", [6])
40
41 call_procedure(cursor, "INSERT_CP1_ESTADO_CIVIL", ["Namorando"])
42
43 call_procedure(cursor, "UPDATE_CP1_ESTADO_CIVIL", [1, "Ficando"])
44
45 call_procedure(cursor, "DELETE_CP1_ESTADO_CIVIL", [7])
46
47 call_procedure(cursor, "INSERT_CP1_CONVENIO", ["Unimed"])
48
49 call_procedure(cursor, "UPDATE_CP1_CONVENIO", [1, "Amil"])
50
51 call_procedure(cursor, "DELETE_CP1_CONVENIO", [6])
52
53
54 # ----- Finaliza a conexão com o banco de dados ----- #
55
56 cursor.close()
```

Essa main realiza uma série de operações em um banco de dados Oracle por meio de chamadas a procedimentos armazenados (stored procedures).

The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays a file tree for 'Challenge 2' containing folders 'Database' and 'python', and files like 'main.py', 'bloco_anonimo.sql', 'cursor.sql', 'funcoes.sql', 'Logical.pdf', 'modelo.dmd', 'procedures.sql', 'regra.sql', 'Relational.pdf', and 'script.sql'. The 'funcoes.sql' file is selected. The main editor area shows the following PL/SQL code:

```
1 SET SERVEROUTPUT ON
2
3 CREATE OR REPLACE FUNCTION validar_cpf (cpf IN VARCHAR2) RETURN BOOLEAN IS
4 BEGIN
5     -- Verifica se o CPF possui 11 dígitos
6     IF LENGTH(cpf) != 11 THEN
7         RETURN FALSE;
8     END IF;
9
10    -- Verifica se todos os caracteres do CPF são dígitos
11    FOR i IN 1..LENGTH(cpf) LOOP
12        IF NOT REGEXP_LIKE(SUBSTR(cpf, i, 1), '[:digit:]') THEN
13            RETURN FALSE;
14        END IF;
15    END LOOP;
16
17    -- Verifica se o CPF é válido
18    DECLARE
19        total    NUMBER;
20        digito1  NUMBER;
21        digito2  NUMBER;
22        soma     NUMBER := 0;
23        resto    NUMBER;
24        peso     NUMBER := 10;
25    BEGIN
26        -- Calcula o primeiro dígito verificador
27        FOR i IN 1..9 LOOP
28            soma := soma + TO_NUMBER(SUBSTR(cpf, i, 1)) * peso;
29            peso := peso - 1;
30        END LOOP;
31        resto := MOD(soma, 11);
32        IF resto < 2 THEN
33            digito1 := 0;
34        ELSE
35            digito1 := 11 - resto;
36        END IF;
37
38        -- Calcula o segundo dígito verificador
39        soma := 0;
40        peso := 11;
41        FOR i IN 1..9 LOOP
42            soma := soma + TO_NUMBER(SUBSTR(cpf, i, 1)) * peso;
43            peso := peso - 1;
44        END LOOP;
45        soma := soma + digito1 * 2;
46        resto := MOD(soma, 11);
47        IF resto < 2 THEN
48            digito2 := 0;
49        ELSE
50            digito2 := 11 - resto;
51        END IF;
```

Dentro do nosso sistema, o cursor `cliente_convenio_cursor` é uma estrutura utilizada para percorrer os registros associados aos clientes e seus convênios dentro das tabelas `CP1_CLIENTE` e `CP1_CONVENIO`.

A screenshot of a code editor interface, likely VS Code, showing a SQL file named 'funcoes.sql'. The editor has a dark theme. On the left, the 'EXPLORER' sidebar shows a project structure with folders like 'Challenge 2' and 'Database', and files like 'main.py', 'bloco_anonimo.sql', 'cursor.sql', 'funcoes.sql', 'Logical.pdf', 'modelo.dmd', 'procedures.sql', 'regra.sql', 'Relational.pdf', and 'script.sql'. The 'funcoes.sql' file is selected. The main editor area shows the following SQL code:

```
79 DECLARE
80     END IF;
81 END;
82 /
83
84 -- Exemplo falso de CPF
85 DECLARE
86     v_cpf_invalido VARCHAR2(14) := '12345678900';
87 BEGIN
88     IF validar_cpf(v_cpf_invalido) THEN
89         DBMS_OUTPUT.PUT_LINE('CPF válido: ' || v_cpf_invalido);
90     ELSE
91         DBMS_OUTPUT.PUT_LINE('CPF inválido: ' || v_cpf_invalido);
92     END IF;
93 END;
94 /
95
96 -- Exemplo verdadeiro de e-mail
97 DECLARE
98     v_email_valido VARCHAR2(100) := 'usuario@dominio.com';
99 BEGIN
100     IF validar_email(v_email_valido) THEN
101         DBMS_OUTPUT.PUT_LINE('E-mail válido: ' || v_email_valido);
102     ELSE
103         DBMS_OUTPUT.PUT_LINE('E-mail inválido: ' || v_email_valido);
104     END IF;
105 END;
106 /
107
108 -- Exemplo falso de e-mail
109 DECLARE
110     v_email_invalido VARCHAR2(100) := 'usuario@dominio';
111 BEGIN
112     IF validar_email(v_email_invalido) THEN
113         DBMS_OUTPUT.PUT_LINE('E-mail válido: ' || v_email_invalido);
114     ELSE
115         DBMS_OUTPUT.PUT_LINE('E-mail inválido: ' || v_email_invalido);
116     END IF;
117 END;
118 /
```

Essas funções têm como objetivo validar informações importantes no sistema, como CPF e e-mail, garantindo que estão formatados corretamente e seguem as regras de validação adequadas.

```
1 SET SERVEROUTPUT ON
2
3 CREATE OR REPLACE PROCEDURE CLIENTE_CONVENIO_RELATORIO AS
4 BEGIN
5     FOR cliente_info IN (
6         SELECT
7             C.NOME AS NOME_CLIENTE,
8             CO.NOME AS NOME_CONVENIO
9         FROM
10             CP1_CLIENTE C
11             JOIN CP1_CONVENIO CO ON C.CONVENIO_ID = CO.ID
12         ORDER BY
13             C.NOME
14     ) LOOP
15         DBMS_OUTPUT.PUT_LINE('Nome do Cliente: ' || cliente_info.NOME_CLIENTE);
16         DBMS_OUTPUT.PUT_LINE('Convênio: ' || cliente_info.NOME_CONVENIO);
17     END LOOP;
18
19     DBMS_OUTPUT.PUT_LINE(chr(10));
20
21     FOR convenio_count IN (
22         SELECT
23             CO.NOME AS NOME_CONVENIO,
24             COUNT(C.ID) AS NUM_CLIENTES
25         FROM
26             CP1_CLIENTE C
27             JOIN CP1_CONVENIO CO ON C.CONVENIO_ID = CO.ID
28         GROUP BY
29             CO.NOME
30         ORDER BY
31             NUM_CLIENTES DESC
32     ) LOOP
33         DBMS_OUTPUT.PUT_LINE('Convênio: ' || convenio_count.NOME_CONVENIO || ', Número de Clientes: ' || convenio_count.NUM_CLIENTES);
34     END LOOP;
35 END;
36 /
37
38 BEGIN
39     CLIENTE_CONVENIO_RELATORIO;
40 END;
41 /
```

Essa regra de negócio tem como objetivo gerar um relatório sobre os clientes e os convênios associados dentro do sistema.

```
1149
1150
1151 INSERT INTO CPI_HISTORICO_HOSPITAL_CLIENTE (
1152     CLIENTE_ID,
1153     DATA_REGISTRO,
1154     HISTORICO_MEDICO,
1155     EXAMES_REALIZADOS,
1156     MEDICAMENTOS_PRESCRITOS,
1157     OBSERVACOES
1158 ) VALUES (
1159     4,
1160     TO_DATE('2023-01-15', 'YYYY-MM-DD'),
1161     'Fratura de tornozelo',
1162     'Raio-X, ressonância magnética',
1163     'Analgésicos, anti-inflamatórios',
1164     'Cliente em reabilitação fisioterapêutica'
1165 );
1166
1167 INSERT INTO CPI_HISTORICO_HOSPITAL_CLIENTE (
1168     CLIENTE_ID,
1169     DATA_REGISTRO,
1170     HISTORICO_MEDICO,
1171     EXAMES_REALIZADOS,
1172     MEDICAMENTOS_PRESCRITOS,
1173     OBSERVACOES
1174 ) VALUES (
1175     5,
1176     TO_DATE('2023-09-30', 'YYYY-MM-DD'),
1177     'Infarto agudo do miocárdio',
1178     'Eletrocardiograma, angiografia coronariana',
1179     'Antiagregantes plaquetários, estatínicos',
1180     'Cliente em monitoramento cardíaco regular'
1181 );
1182
1183 SELECT
1184     *
1185 FROM
1186     CPI_CLIENTE;
1187
1188 SELECT
1189     C.NOME,
1190     CO.NOME
1191 FROM
1192     CP1_CLIENTE C
1193     JOIN CP1_CONVENIO CO
1194     ON C.CONVENIO_ID = CO.ID;
```

Esse script é responsável por configurar e popular as tabelas de um banco de dados relacional, criando um esquema básico para um sistema de gestão de saúde.