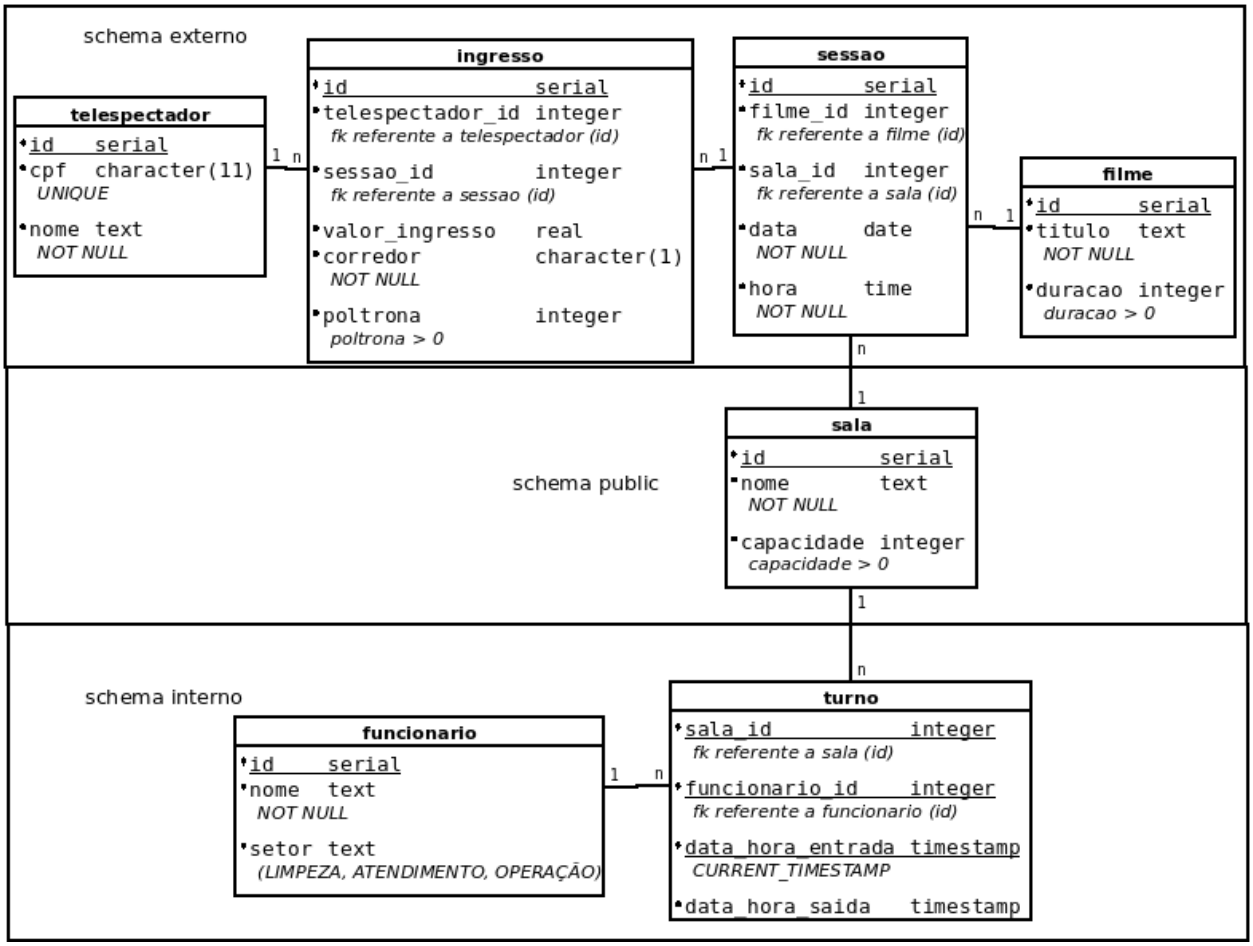
Banco de Dados **Nome:** _____
Atividade Avaliada - 2º Bim. - Valor: 4 **Matrícula:** _____
Professor: Igor Avila Pereira

1. (1.0) Implemente no SGBD PostgreSQL o Banco de Dados projetado pelo Modelo Relacional abaixo:

   Dicas: CREATE SCHEMA, SET search_path TO;



2. (0.5) Construa um STORE PROCEDURE de validação para a coluna *cpf* da tabela **telespectador**.

   Observações:

   - Será preciso acrescentar o STORE PROCEDURE de validação à cláusula (*check*) da coluna *cpf* (tabela **telespectador**);

   Dicas:

   - A função desenvolvida nas aulas pode ser usada. Se não possuir a função em mãos, será preciso implementá-la novamente.

- Lembrando que 00000000000, 11111111111, 22222222222, 33333333333, 44444444444, 55555555555, 66666666666, 77777777777, 88888888888 e 99999999999 também são cpf's **INVÁLIDOS**;

3. (1.0) Sabendo que os Funcionários realizam diversos turnos, construa um STORED PROCEDURE que calcule o salário de um determinado funcionário mediante o número de horas trabalhadas por ele durante o mês atual (mês corrente).

   Observações:

   - Este STORED PROCEDURE deve ter como **parâmetros de entrada**:
     - o **id do funcionário** e,
     - o **valor da hora trabalhada** de acordo com o seu setor:
       * Funcionários da **LIMPEZA** ganham R$ 10 por hora trabalhada,
       * Funcionários do **ATENDIMENTO** ganham R$ 15 e,
       * Funcionários da **OPERAÇÃO** ganham R$ 20.
   - Só vale turnos que começam e terminam no mesmo mês, ou seja, data_hora_entrada e data_hora_saida pertencem ao mesmo mês (mês atual);
   - Utilize os seguintes **casos de teste**:
     - 1 funcionário de cada setor: LIMPEZA, ATENDIMENTO e OPERAÇÃO e,
     - 2 turnos diferentes de trabalho para cada funcionário;
   - **IMPORTANTE:** Caso facilite o cálculo, é **permitido considerar somente horas inteiras/completas de cada turno de trabalho**:
     - **Ex:**
       1. Em um turno que durou 3 horas, 30 minutos e 45 segundos, é permitido considerar somente 3 horas.
       2. Em um turno de 04:59, é permitido considerar somente 4 horas.

   Dicas: EXTRACT, FOR LOOP, CAST e etc.

4. (1.0) Construa uma TRIGGER que, a cada novo funcionário, crie um novo turno de trabalho para este funcionário recém cadastrado.

   Observações:

   - Este novo turno de trabalho deve começar no dia seguinte ao dia atual do cadastramento/inserção e deve começar às 08:00 (oito da manhã).
   - A sala onde este funcionário irá trabalhar em seu primeiro turno de trabalho deve ser escolhida aleatoriamente entre as salas existentes, ou seja, pela TRIGGER.

   Dicas: ORDER BY RANDOM(), LIMIT, CURRENT_DATE + INTERVAL '1 day'.

5. (0.5) Construa uma *view* que retorne o **nome da sala**, o **título do filme**, o **nome do telespectador**, a **data** e **hora** da sessão, o **corredor** e a **poltrona** escolhida por cada telespectador para todos os ingressos. **Obs:** retornar **data** no formato dd/mm/aaaa.

   Dicas: CREATE VIEW, INNER JOIN.

Algoritmo para Validar CPF

Quando se está trabalhando em um sistema corporativo, é comum a necessidade de validar CPF. Muita gente não sabe que um CPF para ser válido não basta apenas atender à máscara “**###.###.###-##”**(o caractere ‘#’ representa um número), existe uma regra matemática que também deve ser verificada para um CPF ser considerado válido. Se você acha que é complicado verificar se um CPF é válido ou não, você vai se surpreender!

REGRA PARA VALIDAR CPF

O cálculo para validar um CPF é especificado pelo Ministério da Fazenda, que disponibiliza no próprio site as funções (em javascript) para validação de CPF. Vamos entender como funciona.

O CPF é formado por 11 dígitos numéricos que seguem a máscara “###.###.###-##”, a verificação do CPF acontece utilizando os 9 primeiros dígitos e, com um cálculo simples, verificando se o resultado corresponde aos dois últimos dígitos (depois do sinal “-“).

> *Vamos usar como exemplo, um CPF fictício “529.982.247-25”.*

Validação do primeiro dígito

Primeiramente multiplica-se os 9 primeiros dígitos pela sequência decrescente de números de 10 à 2 e soma os resultados. Assim:

> *5 * **10** + 2 * **9** + 9 * **8** + 9 * **7** + 8 * **6** + 2 * **5** + 2 * **4** + 4 * **3** + 7 * **2***

O resultado do nosso exemplo é:

> ***295***

O próximo passo da verificação também é simples, basta multiplicarmos esse resultado por 10 e dividirmos por 11.

> *295 * **10** / **11***

O resultado que nos interessa na verdade é o RESTO da divisão. Se ele for igual ao **primeiro dígito verificador** (primeiro dígito depois do ‘-‘), a primeira parte da validação está correta. **Observação Importante:** Se o resto da divisão for igual a 10, nós o consideramos como 0.

Vamos conferir o primeiro dígito verificador do nosso exemplo:

*O resultado da divisão acima é ‘268’ e o RESTO é **2***

Isso significa que o nosso CPF exemplo passou na validação do primeiro dígito.

Validação do segundo dígito

A validação do segundo dígito é semelhante à primeira, porém vamos considerar os 9 primeiros dígitos, mais o primeiro dígito verificador, e vamos multiplicar esses 10 números pela sequencia decrescente de 11 a 2. Vejamos:

*5 * **11** + 2 * **10** + 9 * **9** + 9 * **8** + 8 * **7** + 2 * **6** + 2 * **5** + 4 * **4** + 7 * **3** + 2 * **2***

O resultado é:

347

Seguindo o mesmo processo da primeira verificação, multiplicamos por 10 e dividimos por 11.

*347 * **10** / **11***

Verificando o RESTO, como fizemos anteriormente, temos:

*O resultado da divisão é ‘315’ e o RESTO é **5***

Verificamos, se o resto corresponde ao segundo dígito verificador.

Com essa verificação, constatamos que o CPF 529.982.247-25 é válido.