INSTITUTO UNIVERSITÁRIO DE LISBOA

Projeto Base de Dados
Relatório - Parte 2

1º semestre
2023/2024
Informática e Gestão de Empresas

Grupo 44
Alexandre Carvoeiro nº110905
Francisco Lopes nº111358
Francisco Rainho nº106312

17 dezembro de 2023

1.Índice

2.Introdução

Na continuidade do projeto dedicado à disciplina de Bases de Dados, a Parte 2 concentra-se no refinamento e expansão da base de dados "MUSISYS", concebida na etapa anterior. Esta fase do trabalho visa aprimorar a eficiência da estrutura existente por meio de otimizações (1) e, a partir do modelo relacional otimizado, propõe-se desenvolver automatismos (2), implementar um conjunto de pesquisas (3) e criar um protótipo web-based (HTML/PHP) destinado à demonstração prática do sistema (4).

Ao contrário da Parte 1, onde a ênfase recaiu na conceção inicial da base de dados, a Parte 2 alinha-se com a evolução dinâmica do projeto. Nesta etapa, o foco desloca-se para a implementação de melhorias estratégicas na estrutura da base de dados, considerando as necessidades específicas do Sistema de Informação destinado à gestão de festivais musicais. Desta forma, o objetivo central é proporcionar uma base de trabalho comum para todos os envolvidos no projeto, garantindo a avaliação imparcial e consistente, sem a interferência de decisões preexistentes.

A abordagem adotada nesta fase visa não apenas a eficácia técnica, mas também a praticidade na utilização do sistema. As otimizações visam aperfeiçoar a performance, enquanto os automatismos, pesquisas e o protótipo web-based buscam agregar funcionalidades e usabilidade ao Sistema de Informação, tornando-o mais completo e acessível.

Deste modo, esta segunda parte do projeto representa um passo significativo na transformação da visão conceitual da base de dados em uma solução prática e funcional, alinhada com os requisitos dinâmicos de gestão de festivais musicais.

3.Desenvolvimento

3.1Otimização da base de dados

Com o modelo relacional fornecido, decidimos otimizar as tabelas que apresentavam associações de generalização de forma a eliminar a necessidade de realizar joins na consulta de dados. O diagrama de classes que originou o modelo relacional, continha quatro generalizações onde as subclasses tinham identidade própria.
Primeira generalização: A tabela pagante e convidado eram subclasses da tabela espetador_com_bilhete. Neste caso, decidimos que estas subclasses já não iriam referenciar a tabela espetador_com_bilhete, eliminado as suas chaves estrangeiras.
Segunda Generalização: A tabela espetador_com_bilhete e a tabela jornalista eram subclasses da tabela espetador. Nesta eliminamos as chaves estrangeiras que referenciavam a tabela espetador. Como consequência disto, já não seria necessário dar uso à tabela espetador e acabamos por eliminá-la, passando todos os seus atributos para as suas subclasses.
Terceira Generalização: A tabela individual e grupo eram subclasses da tabela participante. Tal como nas outras, também eliminamos as chaves estrangeiras que referenciavam a tabela participante. Apesar de não eliminarmos a tabela Participante dado que esta continha relações com outras tabelas, passamos o atributo “nome” para as suas subclasses.
Quarta Generalização: A tabela roadie era subclasse da tabela tecnico. Mais uma vez, eliminamos a chave estrangeira que referenciava a tabela tecnico. Tal como a tabela participante, não eliminamos a tabela tecnico visto ter relações com outras tabelas, porém passamos o atributo “nome” para as suas subclasses.
Estas alterações na estrutura das tabelas afetadas não causaram limitações em termos de validação dos dados.

O modelo relacional continha tabelas que possuíam chaves primárias ineficientes, ou seja chaves que não continham domínio numérico. Portanto, por razões de eficiência, as chaves primárias dessas tabelas passaram a ser colunas criadas por nós que continham um mecanismo de identificação (um id). As tabelas afetadas por esta otimização foram as seguintes: estilo, media, pais, papel e tipo_de_bilhete. As tabelas que foram afetadas por estas alterações, causaram constrangimentos para as chaves estrangeiras que referenciavam as suas chaves primárias anteriores. Para contornar estas limitações, tivemos de eliminar essas chaves estrangeiras e criar outras de forma a que referenciassem as novas chaves primárias criadas.

Para evitar que sejam feitos inúmeros “Select” , decidimos criar campos com valores pré-calculados na tabela edição (totalBilhetesVendidos e total_bilhetes_devolvidos) e

na tabela convidado (total_convidados). Estes campos não causaram qualquer tipo de limitação na estrutura das tabelas.

3.2 Automatismos

3.2.1 Triggers

```
T1
BEGIN

DECLARE participante_artista_codigo INT;
DECLARE palco_artista_codigo INT;

   -- Obter o codigo do participante (artista) associado ao roadie
   SELECT Participante_codigo INTO participante_artista_codigo
   FROM roadie
   WHERE Tecnico_numero = NEW.Tecnico_numero_;

   -- Obter o codigo do palco onde o artista atua
   SELECT Palco_codigo
   INTO palco_artista_codigo
   FROM contrata
   WHERE Edicao_numero_ = NEW.Palco_Edicao_numero_
      AND Participante_codigo_ = participante_artista_codigo;

   -- Verificar se o palco onde o roadie esta sendo atribuido e o mesmo onde o artista atua
   IF palco_artista_codigo IS NULL OR palco_artista_codigo != NEW.Palco_codigo_
THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Roadie nao pode montar em palco onde o artista nao atua';
   END IF;
END
```

Este trigger denominado roadie_montagem_palco é acionado antes de inserir uma linha na tabela montado. Este verifica se o palco ao qual o roadie está associado é o mesmo que o palco onde o seu artista vai atuar. Se não for o caso, o trigger cancela a inserção usando SIGNAL SQLSTATE ‘45000’.

T2. a)

```
BEGIN
   DECLARE num_serie_bilhete INT;
   DECLARE tipo_bilhete_id INT;
   DECLARE data_festival DATE;

   IF NEW.devolvido=0 THEN
   -- Obter informacoes do bilhete recem-criado
   SELECT NEW.num_serie, NEW.Tipo_de_bilhete_id
   INTO num_serie_bilhete, tipo_bilhete_id;

   -- Obter a data do festival associada ao bilhete
   SELECT Dia_festival_data_
   INTO data_festival
   FROM acesso
   WHERE Tipo_de_bilhete_id = tipo_bilhete_id
      AND Dia_festival_data_ IS NOT NULL;

   -- Atualizar qtd_espetadores no dia de festival correspondente
   UPDATE dia_festival
   SET qtd_espetadores = qtd_espetadores + 1
   WHERE data = data_festival;

   ELSE
     UPDATE dia_festival
   SET qtd_espetadores = qtd_espetadores - 1
   WHERE data = data_festival;
   END IF;
END
```

Este trigger denominado “after_bilhete_insert”, tal como o nome indica é acionado depois de ser inserido dados na tabela bilhete. Caso o bilhete inserido na base de dados não seja para devolução, este obtém o número do bilhete e o tipo de bilhete associado. Em seguida, encontra a data do festival associado a esse bilhete na tabela acesso e atualiza a coluna qtd_espetadores na tabela dia_festival correspondente.

T2.b)

```
BEGIN
   DECLARE capacidade_diaria INT;
   DECLARE total_espetadores INT;
   DECLARE data_festival DATE;
   DECLARE tipo_bilhete_id INT;

   -- Obter informações do bilhete a ser inserido
   SELECT NEW.Tipo_de_bilhete_id
   INTO tipo_bilhete_id;

   -- Obter a data do festival associada ao bilhete
   SELECT Dia_festival_data_
   INTO data_festival
   FROM acesso
   WHERE Tipo_de_bilhete_id = tipo_bilhete_id
      AND Dia_festival_data_ IS NOT NULL;

   -- Obter a capacidade diaria do recinto para o dia de festival
   SELECT lotacao, qtd_espetadores
   INTO capacidade_diaria, total_espetadores
   FROM edicao e
   JOIN dia_festival df ON e.numero = df.Edicao_numero
   WHERE df.data = data_festival;

   -- Verificar se a lotacao diaria foi excedida
   IF total_espetadores >= capacidade_diaria THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Lotacao diaria excedida. Nao e possivel adicionar mais
bilhetes para este dia.';
   END IF;
END
```

O trigger que cumpre objetivo T2. b) é designado por “before_insert_bilhete”, o que significa que este trigger é acionado antes da inserção de dados na tabela bilhete. Este obtém o tipo de bilhete a ser inserido e encontra a data do festival associado a esse bilhete na tabela acesso. Em seguida verifica se a lotação diária foi excedida, consultando a capacidade diária e a quantidade de espetadores para o dia de festival correspondente. Se a lotação foi excedida, um sinal de erro é gerado e a inserção é cancelada.

3.2.2 Stored Procedures

```
P1
BEGIN
   DECLARE clone_numero INT;

   -- Clonar edicao
   INSERT INTO edicao (numero, nome, localidade, local, data_inicio, data_fim, lotacao,
totalBilhetesVendidos, total_bilhetes_devolvidos)
   SELECT MAX(numero) + 1, nome, localidade, local, clone_data_inicio, NULL, lotacao,
NULL, NULL
   FROM edicao WHERE numero = original_numero;

   SET clone_numero = (SELECT MAX(numero) FROM edicao);

   -- Clonar palcos
   INSERT INTO palco (Edicao_numero, codigo, nome)
   SELECT clone_numero, codigo, nome FROM palco WHERE Edicao_numero =
original_numero;

   -- Clonar dias do festival
   INSERT INTO dia_festival (Edicao_numero, data, qtd_espetadores)
   SELECT clone_numero, DATE_ADD(data, INTERVAL DATEDIFF(clone_data_inicio,
data) DAY), qtd_espetadores
   FROM dia_festival WHERE Edicao_numero = original_numero;
END
```

Este procedimento denominado por “CloneEdicao” tem como parâmetros: o número da edição original (original_numero) do tipo INT e a data de início para o clone (clone_data_inicio) do tipo DATE. Este clona a edição, os palcos e os dias de festival. É de notar que os dias de festival são clonados de acordo com a diferença de dias entre a data de início da edição original e a nova data de início especificada para o clone.

P2 a)

```
BEGIN
   -- Criacao da Edicao
   INSERT INTO edicao(numero, nome, localidade, local, data_inicio, data_fim, lotacao)
   VALUES(p_numero, p_nome, p_localidade, p_local, p_data_inicio, p_data_fim,
p_lotacao);
END
```

P2 b)

```
BEGIN
   DECLARE edicao_existe INT;

   -- Verificar se a edicao existe
   SELECT COUNT(*) INTO edicao_existe
   FROM edicao
   WHERE numero = p_numero;

   IF edicao_existe > 0 THEN
      -- A edicao existe, então podemos adicionar os palcos

      -- Criação dos Palco associado a Edicao
      INSERT INTO palco(Edicao_numero, codigo, nome)
      VALUES(p_numero, p_palco_codigo_1, p_palco_nome_1);
    END IF;
   END
```

Para atingir o objetivo de p2, decidimos criar duas stored procedures. A primeira designada “CriarEdicao” tem como parâmetros, todos os dados necessários para a criação de uma edição. Esta insere os dados recebidos na tabela edicao. A segunda designada “criar_Palco_Para_Edicao” tem como parâmetros, os dados necessários para a criação de um palco e a edição que se pretende que o palco seja inserido. Esta verifica se a edição recebida como parâmetro existe. Se esta existe, então é permitida a criação de um palco com as características pretendidas associadas à edição escolhida.

3.2.3 Functions

```
F1
BEGIN
   DECLARE total_lucro DECIMAL(10, 2);
   DECLARE total_edicoes INT;

   -- Inicialize as variaveis
   SET total_lucro = 0;
   SET total_edicoes = 0;

   -- Loop atraves das edicoes para calcular o total do lucro
   FOR each_edicao IN (SELECT DISTINCT Edicao_numero FROM contrata) DO
      SET total_lucro = total_lucro + (
         SELECT SUM(cachet) FROM contrata WHERE Edicao_numero_ =
each_edicao.Edicao_numero
      );
      SET total_edicoes = total_edicoes + 1;
   END FOR;

   -- Retorna a media do lucro por edicao
   RETURN total_lucro / total_edicoes;
END
```

Esta função designada “CalcularMedia”, não recebe qualquer tipo de parâmetro e tem como retorno dados do tipo DECIMAL(10,2). Esta usa um loop para iterar através de todas as edições distintas presentes na tabela contrata. Para cada edição, o código dentro do loop adiciona o cachê total (SUM(cachet)) associado a essa edição ao total_lucro. Além disso, o contador total_edicoes é incrementado em 1 para cada edição. A média do lucro por edição retornada pela função é calculada dividindo o total_lucro pelo total_edicoes.

```
F2
BEGIN
DECLARE numeroParticipantes INT;
   -- Encontrar o numero de participantes da ultima edicao
   SELECT COUNT(DISTINCT Participantecodigo) INTO numeroParticipantes
   FROM Contrata
   WHERE Edicaonumero = (SELECT MAX(numero) FROM Edicao);

   -- Retornar o numero de participantes
   RETURN numeroParticipantes;
```

END

Esta função designada “calcularNumeroParticipantes” não recebe qualquer parâmetro e tem como retorno dados do tipo INT. Esta encontra o número de participantes da última edição através da tabela contrata e retorna esse mesmo valor.

3.3Pesquisa de dados

Q1_Cartaz:

```
BEGIN
   SELECT participante.nome, contrata.Dia_festival_data
   FROM contrata
   JOIN participante ON contrata.Participante_codigo_ = participante.codigo
   WHERE contrata.Edicao_numero_ = edicao_numero_p
   ORDER BY contrata.Dia_festival_data ASC, contrata.cachet DESC;
END
```

Para responder a Q1, decidimos criar uma stored procedure chamada Q1_Cartaz que tem como parâmetro “edicao_numero_p” do tipo TINTY e tamanho 4. Esta implementação seleciona o campo nome da tabela participante e o campo Dia_festival_data da tabela contrata e faz um join entre a tabela contrata e a tabela participante, ou seja seleciona apenas os participantes que têm contrato, usando como condição a igualdade entre a chave estrangeira Participante_codigo e a chave primária codigo. Este join filtra os resultados para incluir apenas os participantes da edição especificada pelo parâmetro. Os resultados são ordenados pelo dia de festival em ordem ascendente (do dia mais longínquo para o dia mais recente) e dentro de cada dia, pelo cachet em ordem descendente.

Q2_Resultados_diarios

```
CREATE VIEW Q2_Resultados_diarios AS
SELECT
   dia_festival.data AS data,
   dia_festival.qtd_espetadores AS qtd_espetadores,
   SUM(CASE WHEN bilhete.devolvido = 0 THEN tipo_de_bilhete.preco ELSE 0 END) AS
faturacao
FROM
   dia_festival
```

```
LEFT JOIN
   acesso ON dia_festival.data = acesso.Dia_festival_data_
LEFT JOIN
   bilhete ON acesso.Tipo_de_bilhete_id = bilhete.Tipo_de_bilhete_id
LEFT JOIN
   tipo_de_bilhete ON bilhete.Tipo_de_bilhete_id = tipo_de_bilhete.id_tipo_de_bilhete
GROUP By  dia_festival.data;
```

Para responder a Q2, criamos um view chamada Q2_Resultados_Diarios. Primeiramente esta view seleciona o campo data e o campo qtd_espetadores da tabela dia_festival. Depois usa a função "SUM" para calcular a soma dos preços dos bilhetes que não foram devolvidos para cada dia do festival. O resultado é renomeado como faturação e também é selecionado pela view. De seguida faz left join da tabela dia_festival com a tabela acesso, left join da tabela acesso com a tabela bilhete e left join da tabela bilhete com a tabela tipo_de_bilhete. Os resultados são agrupados pelas datas correspondentes aos dias de festival.

Q3

```
BEGIN
   DECLARE qtd_espetadores INT;

   SELECT qtd_espetadores
   INTO qtd_espetadores
   FROM dia_festival
   WHERE Edicao_numero = edicao_numero_p AND data = data_do_festival_p;

   RETURN qtd_espetadores;
END
```

Para responder à Q3, decidimos criar uma stored function chamada Qtd_espetadores_no_dia que recebe como parâmetros a edicao_numero_p do tipo INT e data_do_festival_p do tipo DATE. Esta implementação realiza uma consulta na tabela dia_festival para obter a quantidade de espectadores num determinado dia e numa edição específica do festival, retornando esse valor.

Q4

```
CREATE VIEW Q4_Estilos_musicais_por_edicao AS
SELECT
   contrata.Edicao_numero_ AS Edicao,
   estilo.Nome AS Estilo,
   COUNT(contrata.Participante_codigo_) AS Qtd_artistas
FROM
   contrata
JOIN
   estilo_de_artista ON contrata.Participante_codigo_ =
estilo_de_artista.Participante_codigo_
JOIN
   estilo ON estilo_de_artista.Estilo_id = estilo.id_estilo
GROUP BY
   contrata.Edicao_numero_, estilo.Nome;
```

Para responder à Q4, foi criada uma view chamada Q4_Estilos_musicais_por_edicao que oferece uma representação estruturada da distribuição de artistas nos estilos musicais em cada edição do festival. Em primeiro lugar, da tabela contrata, seleciona a coluna contra.Edicao_numero e renomeia para Edicao, esta representa o número da edição. Renomeia também a estilo.Nome para Estilo e calcula a contagem de artistas agrupados por edição e estilo, renomeando como Qtd_artistas. De seguida realiza um join entre a table contrata e estilo_de_artista com base na correspondência das colunas Participante_codigo, isto associa os artistas aos estilos músicais. Após o primeiro join, o segundo join é entre as tabelas estilo_de_artista e estilo usando as colunas estilo_id e id_estilo, o que estabelece a relação entre os códigos de estilo e os nomes do estilos musicais. Finalmente, os resultados são agrupados com base nos campos Edicao_numero e Nome (Edição e Estilo), permitindo a contagem da quantidade de artistas por estilo em cada edição.

Q5

```
CREATE VIEW Q5_Todos_os_participantes AS
SELECT
   participante.nome,
   DATEDIFF(CURDATE(), contrata.Dia_festival_data) / 365 AS
Anos_desde_ultima_atuacao,
   contrata.cachet AS Ultimo_cachet
FROM
   participante
LEFT JOIN
   contrata ON participante.codigo = contrata.Participante_codigo_
WHERE
```

```
  (contrata.Participante_codigo_, contrata.Dia_festival_data) IN (
    SELECT Participante_codigo_, MAX(Dia_festival_data) AS Ultima_data
    FROM contrata
    GROUP BY Participante_codigo_
  );
```

Para responder à Q5, decidimos criar uma view chamada Q5_Todos_os_participantes, que lista informações sobre os artistas participantes registados num festival. Em primeiro lugar, seleciona os campos que serão incluídos na view: o participante.nome, que seleciona o nome do participante da tabela participante; o Anos_desde_ultima_atuacao que foi o nome para que foi renomeada a diferença entre a data atual e data da última atuação do participante no festival. Em segundo lugar o FROM que especifica as tabelas a serem usadas na seleção, a participante, que contém informações sobre os participantes e após isso realiza um LEFT JOIN entre as tabelas participante e contrata. Em terceiro lugar, no WHERE, que serve para especificar o filtro, verifica se o par de valores (código do participante, data do festival) está presente nos resultados da sub seleção que retorna o código do participante e a data máxima de atuação para cada participante.

Q6

```
BEGIN
  SELECT participante.nome
  FROM participante
   JOIN entrevista ON participante.codigo = entrevista.Participante_codigo_
   JOIN jornalista ON entrevista.Jornalista_num_carteira_profissional_ =
jornalista.num_carteira_profissional
   JOIN edicao ON participante.edicao_numero = edicao.numero
  WHERE edicao.numero = edicao_numero_p
    AND jornalista.nome = nome_jornalista_p;

END
```

Para responder à Q6, foi criada uma stored procedure chamada Q6_Entrevistados_Por que tem como parâmetros a edicao_numero_p do tipo INT e o nome_jornalista_p do tipo VARCHAR de tamanho 100. Esta implementação seleciona a coluna nome da tabela participante e faz um join da tabela entrevista com a tabela participante, ou seja seleciona todos os jornalistas que realizaram entrevistas aos participantes, depois faz um join da tabela jornalista com a tabela entrevista, ou seja seleciona apenas os jornalistas que realizaram entrevistas e de seguida faz um join da tabela edicao com a tabela jornalista, ou seja, seleciona todos os jornalistas presentes na edição

especificada. Os resultados são filtrados com base na edição e o nome do jornalista que foram dados como argumentos.

Q7

```
BEGIN

SELECT participante.nome
FROM participante
LEFT JOIN entrevista ON participante.codigo = entrevista.Participante_codigo_
LEFT JOIN jornalista ON entrevista.Jornalista_num_carteira_profissional_ =
jornalista.num_carteira_profissional
WHERE jornalista.nome = nome_jornalista_p
   AND participante.edicao_numero = (SELECT MAX(numero) FROM edicao)
   AND entrevista.Participante_codigo_ IS NULL;

END
```

Para responder a Q7, foi criada uma stored function chamada Q7_Ainda_Nao_Entrevistados_Por que tem como parâmetro nome_jornalista_p do tipo VARCHAR e tamanho 100. Esta implementação seleciona a coluna nome da tabela participante e depois faz um left join da tabela participante com a tabela entrevista, ou seja, seleciona todos os participantes independentemente se deram uma entrevista, de seguida é feito outro left join da tabela entrevista com a tabela jornalista, ou seja, seleciona todas as entrevistas planeadas para serem feitas aos participantes independentemente se já foram realizadas. Os resultados são filtrados de forma a serem selecionados apenas os participantes associados à edição mais recente que não tenham sido entrevistados pelo jornalista recebido como parâmetro.

4.Conclusão

Em síntese, a Parte 2 do projeto de Base de Dados "MUSISYS" foi fundamental para a evolução e aprimoramento do sistema concebido anteriormente. Priorizando otimizações na estrutura existente, a fase atual buscou não apenas eficiência técnica, mas também a praticidade na utilização do sistema. A implementação de automatismos, pesquisas e um protótipo web-based.

Ao diferir da Parte 1, que se concentrou na conceção inicial da base de dados, a Parte 2 alinhou-se à dinâmica evolutiva do projeto. As otimizações implementadas não só aprimoraram o desempenho, mas também garantiram uma avaliação imparcial e consistente, livre de interferências de decisões preexistentes.

Concluindo, esta segunda parte do projeto marca um passo significativo na transformação da visão conceitual da base de dados em uma solução prática e funcional. A abordagem adotada reflete não apenas a eficácia técnica, mas também a adaptação dinâmica às necessidades do Sistema de Informação, consolidando assim um trabalho que atende de forma abrangente aos requisitos de gestão de festivais musicais.