Baza de date pentru teatru

Baza de date pentru teatru poate fi utilizată pentru a gestiona datele unui teatru, cum ar fi informațiile despre clienți, tranzacții, actori, piese, roluri, săli de spectacol și spectacole. Utilizatorii pot adăuga, actualiza și șterge datele din aceste tabele. De asemenea, pot genera rapoarte despre profitul fiecărui spectacol, numărul de vizionări pentru fiecare actor și gradul de ocupare al fiecărei săli.
Un exemplu de utilizare a acestei aplicații ar fi următorul: un administrator al teatrului poate utiliza interfața de administrare a bazei de date pentru a adăuga o nouă piesă, cu titlul "Romeo și Julieta", autorul "William Shakespeare" și o descriere scurtă a acțiunii. El poate adăuga apoi rolurile din această piesă și actorii care vor juca aceste roluri. După aceea, administratorul poate programa o serie de reprezentări(performances) ale acestei piese în diferite săli de spectacol, cu datele de începere și sfârșit, prețul biletelor și capacitatea fiecărei săli. Clienții pot cumpăra bilete pentru aceste spectacole prin intermediul interfației de utilizator și pot genera rapoarte despre profitul fiecărui spectacol, numărul de vizionări pentru fiecare actor și gradul de ocupare al fiecărei săli.

Acest script creează o bază de date numită "Teatru" și mai multe tabele în interiorul ei:

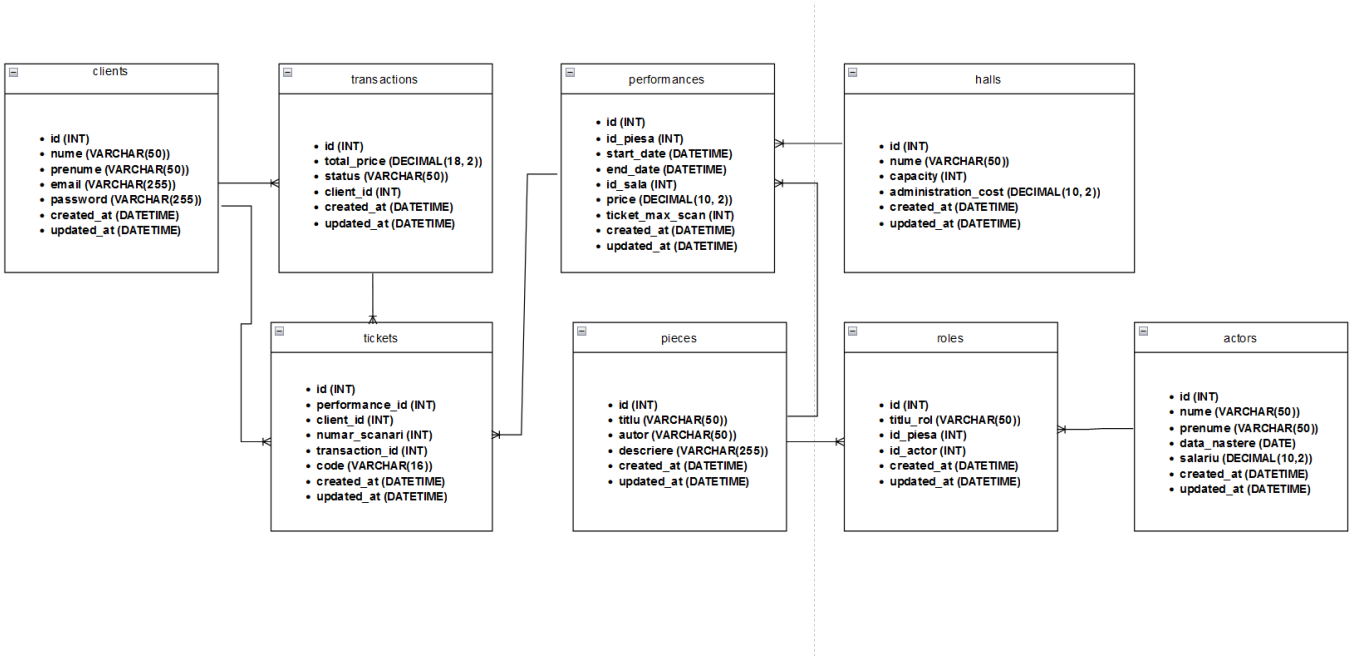


Tabela clients: Această tabelă stochează informații despre clienți, cum ar fi numele, prenumele, adresa de email și parola, precum și data la care au fost create sau actualizate înregistrările.

Tabela transactions: Această tabelă stochează informații despre tranzacții, cum ar fi prețul total, statusul tranzacției și id-ul clientului care a efectuat tranzacția, precum și data la care au fost create sau actualizate înregistrările.

Tabela actors: Această tabelă stochează informații despre actori, cum ar fi numele, prenumele, data nașterii, salariu și data la care au fost create sau actualizate înregistrările.

Tabela pieces: Această tabelă stochează informații despre piese, cum ar fi titlul, autorul, descrierea și data la care au fost create sau actualizate înregistrările.

Tabela roles: Această tabelă stochează informații despre rolurile jucate de actori, cum ar fi titlul rolului, id-ul piesei și id-ul actorului, precum și data la care au fost create sau actualizate înregistrările.

Tabela halls: Această tabelă stochează informații despre sălile de teatru, cum ar fi numele, capacitatea, costul de administrare și data la care au fost create sau actualizate înregistrările.

Tabela performances: Această tabelă stochează informații despre reprezentații, cum ar fi id-ul piesei, data de început și sfârșit, id-ul sălii, prețul biletului și numărul maxim de scanări, precum și data la care au fost create sau actualizate înregistrările.

Tabela tickets: Această tabelă stochează informații despre bilete, cum ar fi id-ul reprezentației, id-ul clientului, numărul de scanări, id-ul tranzacției, codul biletului generat automat și data la care au fost create sau actualizate înregistrările.

Fiecare tabel are un câmp "id" care este setat ca "IDENTITY" cu valoarea inițială 1 și incrementarea 1, acest câmp este cheia primară a tabelului. Fiecare tabel are, de asemenea, câmpuri "created_at" și "updated_at" care stochează data și ora la care au fost create sau actualizate înregistrările în tabel.
Există anumite restricții impuse asupra coloanelor cum ar fi NOT NULL sau UNIQUE, acestea asigură validarea datelor în tabele.

Acest script crează o structură de bază pentru o bază de date de teatru, unde se pot stoca și gestiona informații despre clienți, tranzacții, actori, piese, roluri, săli de teatru și reprezentații, precum și biletele vândute pentru acestea.

Exista declanșatoare (triggers) care sunt utilizate pentru a actualiza coloana "updated_at" din tabelele "transactions", "actors", "pieces", "roles", "performances", "halls" și "tickets" cu data și ora curentă atunci când se efectuează o actualizare.

```
CREATE TRIGGER update_transactions_updated_at
ON transactions
AFTER UPDATE
AS
BEGIN
    UPDATE transactions
    SET updated_at = GETDATE()
    FROM inserted
    WHERE transactions.id = inserted.id;
END
GO
```

```
CREATE TRIGGER update_pieces_updated_at
ON pieces
AFTER UPDATE
AS
BEGIN
    UPDATE pieces
    SET updated_at = GETDATE()
    FROM inserted
    WHERE pieces.id = inserted.id;
END
GO
```

-Trigger-ul "check_ticket_availability" este declanșat după inserarea unui bilet în tabela "tickets". Acesta calculează numărul de bilete vândute pentru o reprezentație specifică și

capacitatea sălii în care se desfășoară reprezentația. Dacă numărul de bilete vândute depășește capacitatea sălii, declanșatorul aruncă o eroare și anulează tranzacția.

```
CREATE TRIGGER check_ticket_availability
ON tickets
AFTER INSERT
AS
BEGIN
    DECLARE @performance_id INT, @ticket_count INT, @hall_capacity INT;
    SET @performance_id = (SELECT top(1) performance_id FROM inserted);
    SET @ticket_count = (SELECT COUNT(*) FROM tickets WHERE performance_id = @performance_id);
    SET @hall_capacity = (SELECT capacity FROM halls h join performances p on p.id_sala = h.id WHERE p.id = @performance_id);
    IF (@ticket_count > @hall_capacity)
    BEGIN
        RAISERROR ('Sala este plina, nu mai sunt bilete disponibile.', 16, 1);
        ROLLBACK TRANSACTION;
    END;
END;
GO
```

-Trigger-ul "update_transaction_total_price" este declanșat după inserarea unui bilet în tabela "tickets". Acesta verifică statusul tranzacției și prețul biletului și actualizează prețul total al tranzacției dacă tranzacția este încă deschisă. Dacă tranzacția nu este deschisă, declanșatorul aruncă o eroare și anulează tranzacția.

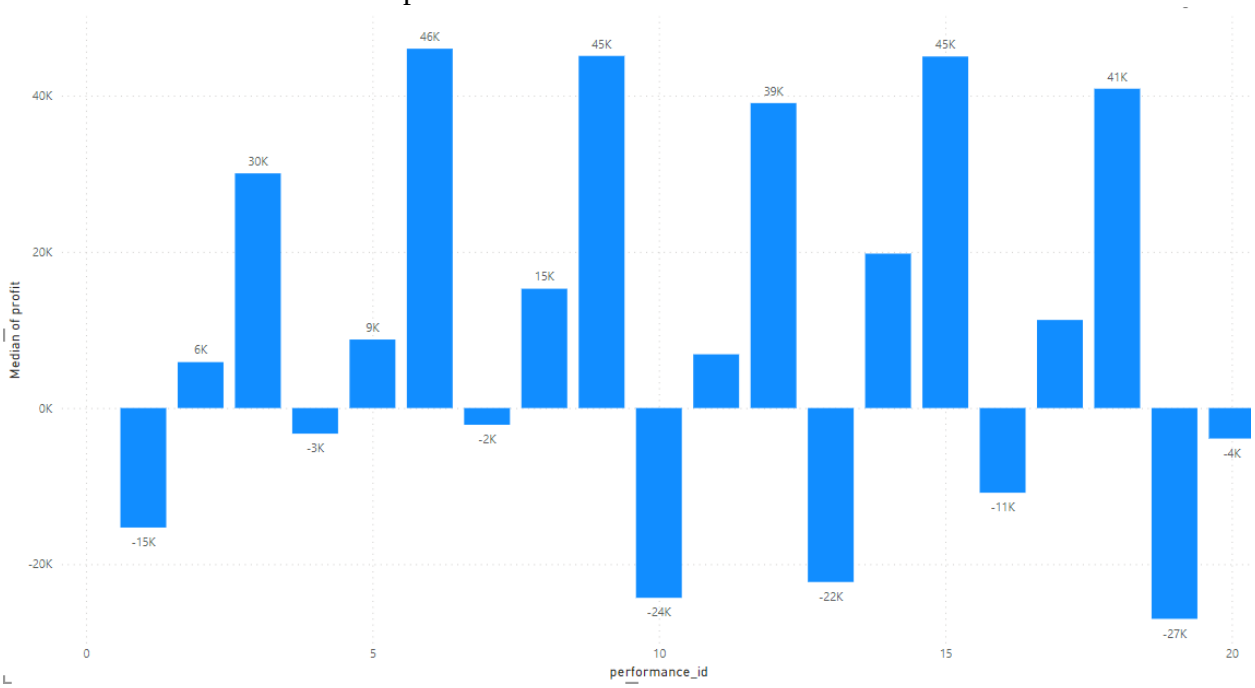
```
CREATE TRIGGER update_transaction_total_price
ON tickets
AFTER INSERT
AS
BEGIN
    DECLARE @performance_id INT;
    DECLARE @price DECIMAL(10, 2);
    DECLARE @transaction_id INT;
    DECLARE @status VARCHAR(50);

    SELECT @performance_id = performance_id, @transaction_id = transaction_id
    FROM inserted;

    SELECT @price = price, @status = status
    FROM performances JOIN transactions
    ON performances.id = @performance_id AND transactions.id = @transaction_id;

    IF @status = 'open'
    BEGIN
        UPDATE transactions
        SET total_price = total_price + @price
        WHERE id = @transaction_id;
    END
    ELSE
    BEGIN
        RAISERROR ('Ticket cannot be added because the transaction is not open', 16, 1);
        ROLLBACK TRANSACTION;
    END
END
GO
```

Aceste declanșatoare au rolul de a adăuga validări si reguli de business pentru baza ta de date, pentru a asigura integritatea datelor și a evita erorile sau problemele care ar putea apărea în cazul în care aceste reguli nu ar fi implementate.

Procedura "sp_generate_profit_report" este utilizată pentru a genera un raport de profit pentru fiecare reprezentație. Aceasta utilizează un cursor pentru a parcurge fiecare reprezentație din tabela "performances" și calculează costul total, venitul total și profitul pentru fiecare reprezentație. Acestea sunt apoi adăugate într-o tabelă temporară și în final selectate și afișate.

```
4   CREATE PROCEDURE sp_generate_profit_report
5   AS
6   BEGIN
7       DECLARE @cost decimal(10,2) = 0,
8               @revenue decimal(10,2) = 0,
9               @profit decimal(10,2) = 0
10      DECLARE @performance_id INT
11      DECLARE @profit_results TABLE (performance_id INT, cost decimal(10,2), revenue decimal(10,2), profit decimal(10,2))
12
13      DECLARE performance_cursor CURSOR FOR
14          SELECT id FROM performances
15      OPEN performance_cursor
16
17      FETCH NEXT FROM performance_cursor INTO @performance_id
18
19      WHILE @@FETCH_STATUS = 0
20      BEGIN
21          SELECT @cost = SUM(a.salariu) + h.administration_cost
22          FROM performances AS p
23          JOIN roles AS r ON r.id_piesa = p.id_piesa
24          JOIN actors AS a ON a.id = r.id_actor
25          JOIN halls AS h ON h.id = p.id_sala
26          WHERE p.id = @performance_id
27          GROUP BY p.id, h.administration_cost
28
29          SELECT @revenue = SUM(tickets_count * p.price)
30          FROM performances AS p
31          JOIN (SELECT performance_id, COUNT(*) as tickets_count
32                FROM tickets
33                JOIN transactions ON tickets.transaction_id = transactions.id
34                WHERE transactions.status = 'closed'
35                GROUP BY performance_id
36                ) AS t
37          ON t.performance_id = p.id
38          WHERE p.id = @performance_id
39
40          SET @profit = @revenue - @cost
41
42          INSERT INTO @profit_results(performance_id, cost, revenue, profit)
43          VALUES(@performance_id, @cost, @revenue, @profit)
44
45          FETCH NEXT FROM performance_cursor INTO @performance_id
46      END
47
48      SELECT performance_id, cost, revenue, profit FROM @profit_results
49
50      CLOSE performance_cursor
51      DEALLOCATE performance_cursor
52
53  END
54  GO
55
```

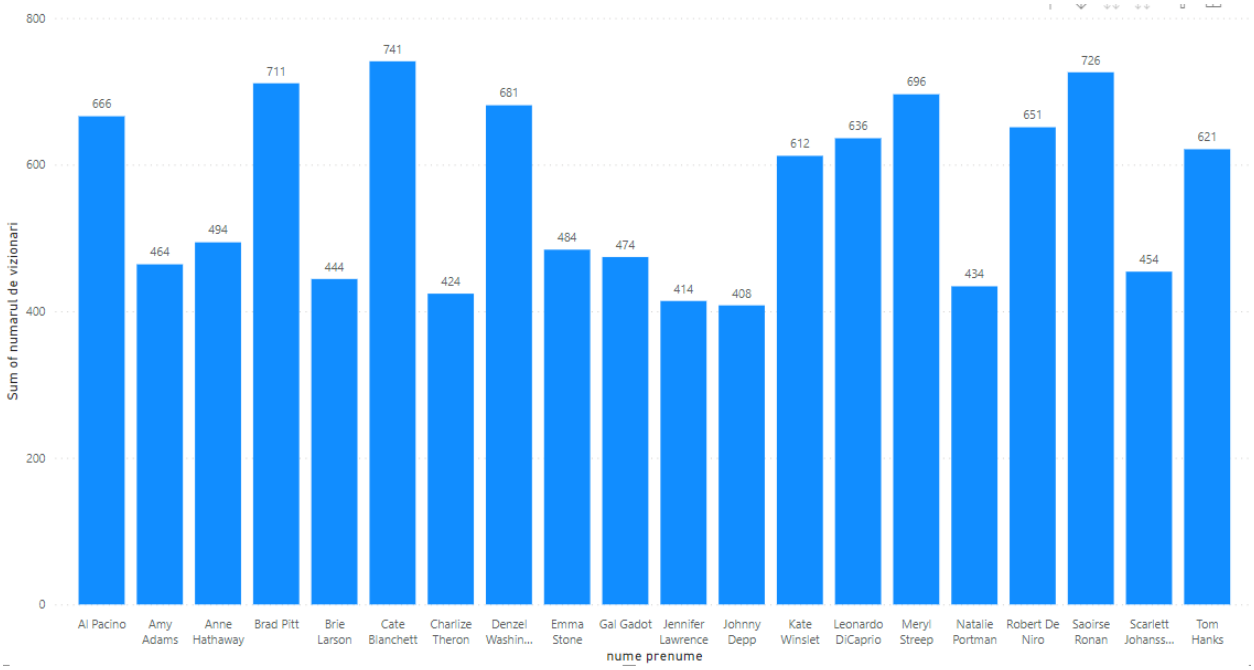
La initializarea bazei de date raportul arata astfel:



Procedura "sp_generate_actor_report" este utilizată pentru a genera un raport care afișează numele și prenumele actorilor, precum și numărul de vizionări pentru fiecare dintre acestea. Acest raport este generat prin agregarea datelor din mai multe tabele (actors, roles, pieces, performances, tickets, transactions) și prin utilizarea filtrului "transactions.status = 'closed'" pentru a include doar biletele vândute.

```sql
CREATE PROCEDURE sp_generate_actor_report
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        actors.nume, actors.prenume, COUNT(tickets.id) AS 'numarul de vizionari'
    FROM actors
    JOIN roles ON actors.id = roles.id_actor
    JOIN pieces ON roles.id_piesa = pieces.id
    JOIN performances ON pieces.id = performances.id_piesa
    JOIN tickets ON performances.id = tickets.performance_id
    JOIN transactions ON tickets.transaction_id = transactions.id
    WHERE transactions.status = 'closed'
    GROUP BY actors.nume, actors.prenume
    ORDER BY COUNT(tickets.id) DESC
END
GO
```

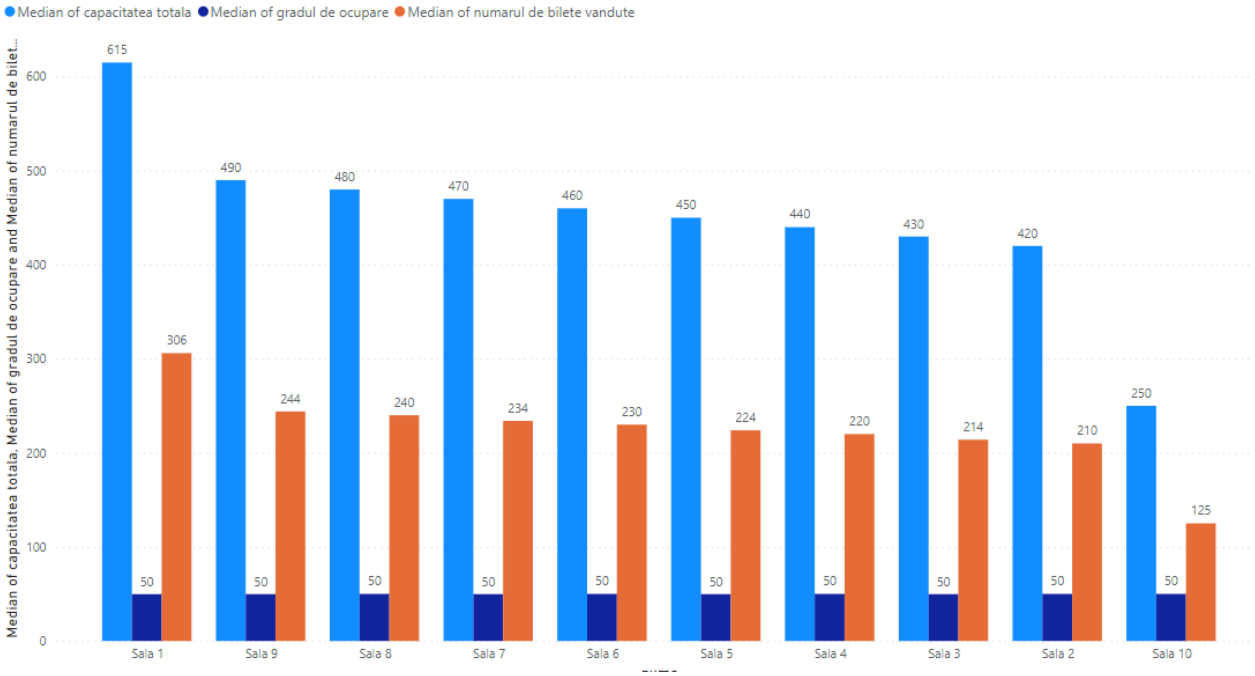
La initializarea bazei de date raportul arata astfel:



Procedura "sp_generate_hall_occupancy_report" este utilizată pentru a genera un raport care afișează numele sălii, numărul de bilete vândute, capacitatea totală a sălii și gradul de ocupare pentru fiecare sală. Raportul este generat prin agregarea datelor din mai multe tabele (halls, performances, tickets, transactions) și prin utilizarea filtrului "transactions.status = 'closed'" pentru a include doar biletele vândute.

```
CREATE PROCEDURE sp_generate_hall_occupancy_report
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        h.nume,
        COUNT(tickets.id) as 'numarul de bilete vandute',
        h.capacity * (SELECT COUNT(DISTINCT performances.id) FROM performances WHERE performances.id_sala = h.id) as 'capacitatea totala',
        ((100.00 * COUNT(tickets.id)) / (h.capacity * (SELECT COUNT(DISTINCT performances.id) FROM performances WHERE performances.id_sala = h.id))) as 'gradul de ocupare'
    FROM halls h
    JOIN performances ON h.id = performances.id_sala
    JOIN tickets ON performances.id = tickets.performance_id
    JOIN transactions ON tickets.transaction_id = transactions.id
    WHERE transactions.status = 'closed'
    GROUP BY h.nume, h.id, h.capacity
    ORDER BY SUM(tickets.id) DESC
END
```

La initializarea bazei de date raportul arata astfel:



In concluzie, baza de date creata, numita "Teatru", ofera o structura de organizare si stocare a informatiilor necesare pentru managementul unei companii de teatru. Tabelele si procedurile stocate ofera posibilitatea de a pastra evidenta clientilor, tranzactiilor, actorilor, pieselor, rolurilor, salilor, reprezentatiilor, biletelor si altele. De asemenea, declansatoarele si procedurile stocate asigura functionalitatea si mentinerea integritatii datelor in baza de date. Rapoartele generate prin procedurile stocate permit analiza si intelegerea performantei si profitabilitatii companiei de teatru. Aceasta documentatie ofera o imagine de ansamblu asupra structurii si functionalitatii bazei de date, si poate fi utilizata pentru mentenanta si extinderea acesteia in viitor.