Projeto Programação com Banco de Dados (Solução)

Criado por: Juan Manuel García Delgado (522097084) - Gustavo Henrique de Oliveira Queiroz (120031064) - Juan Pablo Monteiro Fernandes (219031144) - Lucca França Possas (120083064) - Mauro Gonçalves Bueno (216031126)

Neste pdf você encontra as soluções do projeto de programação do banco de dados. Os scripts podem ser encontrados na pasta compactada.

PARTE 1

1. **Consultar as tabelas de catálogo para listar todos os índices existentes acompanhados das tabelas e colunas indexadas pelo mesmo.**

Solução.

```
# Q1
SELECT DISTINCT
    STATISTICS.TABLE_NAME,
    STATISTICS.INDEX_NAME,
    STATISTICS.COLUMN_NAME
FROM STATISTICS WHERE INDEX_SCHEMA = 'chinook';
```

2. **Criar usando a linguagem de programação do SGBD escolhido um procedimento que remova todos os índices de uma tabela informada como parâmetro.**

Solução.

```
# Q2
USE Chinook;
DROP PROCEDURE IF EXISTS deleteIndex;

delimiter //
create procedure deleteIndex(IN table_name_ varchar(255))
begin
    declare finished INTEGER DEFAULT 0;
    declare dropCommand varchar(255);
    declare curDrop cursor for
        SELECT CONCAT('DROP INDEX ', index_name,
            ' ON ', table_schema, '.', table_name, ';')
        FROM information_schema.statistics
        WHERE index_name != 'PRIMARY'
          AND table_name = table_name_
          AND table_schema = 'Chinook';

    declare continue handler
        for not found set finished = 1;

    open curDrop;
```

```
        lloop:
        loop
            fetch curDrop into dropCommand;
            IF finished=1 then
                leave lloop;
            end IF;

            set @sdropCommand = dropCommand;

            prepare dropClientUpdateKeyStmt FROM @sdropCommand;

            execute dropClientUpdateKeyStmt;

            deallocate prepare dropClientUpdateKeyStmt;
        end loop;

        close curDrop;
    end//
    delimiter ;

    -- TESTE DO PROCEDURE
    CREATE UNIQUE INDEX test1 ON Artist (Name) USING HASH;
    CREATE UNIQUE INDEX test2 ON Artist (Name) USING HASH;

    SELECT DISTINCT TABLE_NAME, INDEX_NAME, COLUMN_NAME
    FROM INFORMATION_SCHEMA.STATISTICS
    WHERE TABLE_SCHEMA = 'Chinook'
      AND TABLE_NAME = 'Artist';

    CALL deleteIndex('Artist');

    SELECT DISTINCT TABLE_NAME, INDEX_NAME, COLUMN_NAME
    FROM INFORMATION_SCHEMA.STATISTICS
    WHERE TABLE_SCHEMA = 'Chinook'
      AND TABLE_NAME = 'Artist';
```

3. **Consultar as tabelas de catálogo para listar todas as chaves estrangeiras existentes informando as tabelas e colunas envolvidas.**

Solução.

```
# Q3
USE information_schema;
SELECT COLUMN_NAME 'Colum_Name',
       j.TABLE_NAME 'Table_name',
       j.CONSTRAINT_NAME 'FK_name'
```

```
FROM TABLE_CONSTRAINTS j
        JOIN KEY_COLUMN_USAGE k
        ON j.CONSTRAINT_NAME = k.CONSTRAINT_NAME
WHERE k.CONSTRAINT_SCHEMA = 'Chinook'
  AND j.CONSTRAINT_TYPE = 'FOREIGN_KEY'
  AND j.TABLE_SCHEMA = 'Chinook';
```

4. **Criar usando a linguagem de programação do SGBD escolhido um script que construa de forma dinâmica a partir do catálogo os comandos create table das tabelas existentes no esquema exemplo considerando pelo menos as informações sobre colunas (nome, tipo e obrigatoriedade) e chaves primárias e estrangeiras.**

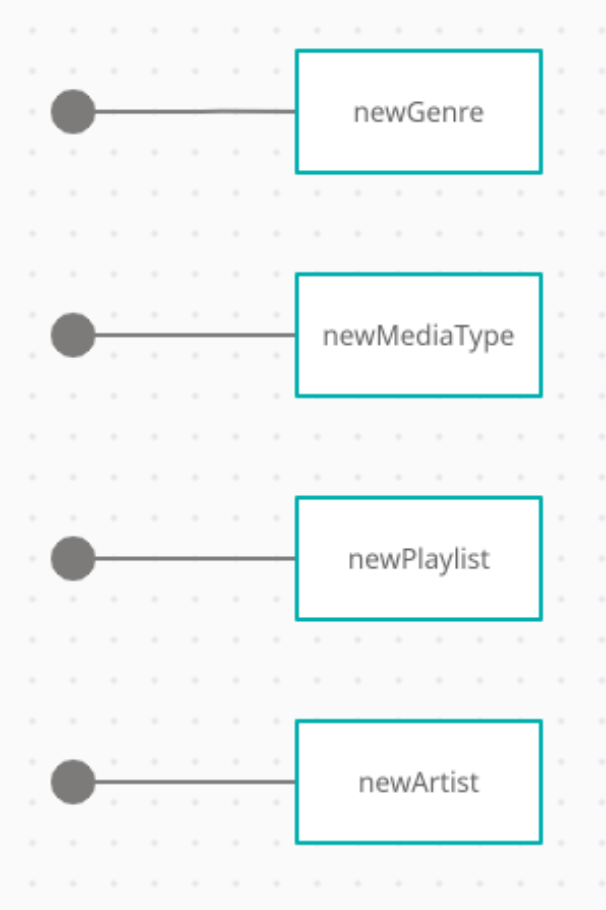
Solução.

```
# Q4
USE information_schema;
select * FROM information_schema.TABLE_CONSTRAINTS
where TABLE_SCHEMA='Chinook' and CONSTRAINT_NAME='PRIMARY';

SELECT CONCAT(
    'CREATE TABLE ',
    t2.table_name,
    ' ( ',
    attrs,
    IF(primary_key IS NOT NULL,
    CONCAT(', PRIMARY KEY(', primary_key, ')'), ''),
    ' );'
) AS '' FROM (
        SELECT GROUP_CONCAT(column_name)
        AS attrs, table_name FROM (
            SELECT CONCAT(
                ' ',
                COLUMN_NAME ,
                ' ',
                DATA_TYPE,
                IF(DATA_TYPE = 'varchar',
                CONCAT('(',CHARACTER_MAXIMUM_LENGTH, ')'), ''),
                ' ',
                IF(IS_NULLABLE = 'No', '', 'NOT_NULL')
            ) AS column_name,
            k.TABLE_NAME AS table_name
            FROM COLUMNS k WHERE k.TABLE_SCHEMA = 'Chinook'

        ) AS t1 GROUP BY table_name
    ) AS t2 INNER JOIN (
                SELECT
                    GROUP_CONCAT( COLUMN_NAME) AS primary_key,
```

```
                    tc.TABLE_NAME AS table_name
                FROM TABLE_CONSTRAINTS tc
                INNER JOIN KEY_COLUMN_USAGE k
                ON tc.CONSTRAINT_NAME = k.CONSTRAINT_NAME
                AND tc.table_name = k.table_name
                WHERE k.CONSTRAINT_SCHEMA = 'Chinook'
                AND tc.TABLE_SCHEMA = 'Chinook'
                AND tc.CONSTRAINT_TYPE = 'PRIMARY_KEY'
                GROUP BY tc.table_name
            ) AS t3 ON t2.table_name = t3.table_name;


SELECT CONCAT(
    'ALTER_TABLE_',
    tc.TABLE_NAME,
    '_ADD_CONSTRAINT_',
    tc.CONSTRAINT_NAME,
    '_FOREIGN_KEY_(',
    COLUMN_NAME,
    ')_REFERENCES_',
    REFERENCED_TABLE_NAME,
    '_(',
    REFERENCED_COLUMN_NAME,
    ');'
) as ''
FROM TABLE_CONSTRAINTS tc
INNER JOIN KEY_COLUMN_USAGE k
ON tc.CONSTRAINT_NAME = k.CONSTRAINT_NAME
WHERE k.CONSTRAINT_SCHEMA = 'Chinook'
AND tc.TABLE_SCHEMA = 'Chinook'
AND tc.CONSTRAINT_TYPE = 'FOREIGN_KEY';
```

5. **Implemente uma solução através da programação em banco de dados para validar os valores de uma coluna que represente uma situação (estado) garantindo que os seus valores e suas transições atendam a especificação de um diagrama de transição de estados (DTE). Quanto mais genérica e reutilizável for a solução melhor a pontuação nessa questão. Junto da solução deverá ser entregue um cenário de teste demonstrando o funcionamento da solução.**

Solução.




```
USE Chinook;
# ---------------------- TRIGGERS ----------------------

CREATE TRIGGER aumentaValorG
    BEFORE INSERT
    ON Genre
    FOR EACH ROW
BEGIN
    SET new.GenreId = (SELECT MAX(GenreId) + 1 FROM Genre);
END;

CREATE TRIGGER aumentaValorM
    BEFORE INSERT
    ON MediaType
    FOR EACH ROW
BEGIN
    SET new.MediaTypeId =
    (SELECT MAX(MediaTypeId) + 1 FROM MediaType);
END;

CREATE TRIGGER aumentaValorPL
    BEFORE INSERT
```

```
    ON Playlist
    FOR EACH ROW
BEGIN
    SET new.PlaylistId =
    (SELECT MAX(PlaylistId) + 1 FROM Playlist);
END;

CREATE TRIGGER aumentaValorAR
    BEFORE INSERT
    ON Artist
    FOR EACH ROW
BEGIN
    SET new.ArtistId = (SELECT MAX(ArtistId) + 1 FROM Artist);
END;

CREATE TRIGGER aumentaValorAlb
    BEFORE INSERT
    ON Album
    FOR EACH ROW
BEGIN
    SET new.AlbumId = (SELECT MAX(AlbumId) + 1 FROM Artist);
END;

# ------------------------------------------------------------


delimiter //
CREATE PROCEDURE newGenre(
    IN name_genre varchar(120)
)
BEGIN
    declare done int default FALSE;
    declare name_g varchar(120);
    declare recorreGenre cursor for
        SELECT Name FROM Genre;

    declare continue handler for not found set done = true;

    open recorreGenre;
    read_loop:
    loop
        fetch recorreGenre INTO name_g;
        IF (name_g LIKE name_genre) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Genre already exists';
        END IF;
        IF done then
```

```
            leave read_loop;
        end IF;
    end loop;

    close recorreGenre;
    INSERT INTO Genre values (null, name_genre);
end;

delimiter //
CALL newGenre('Nuevo');

SELECT *
from Chinook.Genre;

DELETE
FROM Genre
where GenreId = 26;

delimiter //

CREATE PROCEDURE newMediaType(
    IN name_mt_ varchar(120)
)
BEGIN
    declare done int default FALSE;
    declare name_mt varchar(120);
    declare recorreMT cursor for
        SELECT Name FROM MediaType;

    declare continue handler for not found set done = true;

    open recorreMT;
    read_loop:
    loop
        fetch recorreMT INTO name_mt;
        IF (name_mt LIKE name_mt_) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'MediaType already exists';
        END IF;
        IF done then
            leave read_loop;
        end IF;
    end loop;

    close recorreMT;
    INSERT INTO MediaType values (null, name_mt_);
end;
```

```
delimiter //


CREATE PROCEDURE newPlayList(
    IN name_pl_ varchar(120)
)
BEGIN
    declare done int default FALSE;
    declare name_pl varchar(120);
    declare recorrePL cursor for
        SELECT Name FROM Playlist;

    declare continue handler for not found set done = true;

    open recorrePL;
    read_loop:
    loop
        fetch recorrePL INTO name_pl;
        IF (name_pl LIKE name_pl_) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Playlist already exists';
        END IF;
        IF done then
            leave read_loop;
        end IF;
    end loop;

    close recorrePL;
    INSERT INTO Playlist values (null, name_pl_);
end;

delimiter //

CREATE PROCEDURE newArtist(
    IN name_ar_ varchar(120)
)
BEGIN
    declare done int default FALSE;
    declare name_ar varchar(120);
    declare recorreAR cursor for
        SELECT Name FROM Artist;

    declare continue handler for not found set done = true;

    open recorreAR;
    read_loop:
    loop
```

```
            fetch recorreAR INTO name_ar;
            IF (name_ar LIKE name_ar_) THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Artist already exists';
            END IF;
            IF done then
                leave read_loop;
            end IF;
        end loop;

        close recorreAR;
        INSERT INTO Artist values (null, name_ar_);
    end;

    delimiter //

    # ------------------------------ Testes ------------------------------
    #Novo Genre
    CALL newGenre('Trap');
    SELECT *
    FROM Genre;

    #Novo mediaType
    CALL newMediaType('Mp4t');
    SELECT *
    FROM MediaType;

    #Nova Playlist
    CALL newPlayList('Hits Bad Bunny');
    SELECT *
    FROM Playlist;

    #Nova Artist
    CALL newArtist('Bad Bunny');
    SELECT *
    FROM Artist;
```

PARTE 2

1. **Criar regras semânticas, que são regras que não podem ser garantidas pela estrutura do modelo relacional, usando o esquema exemplo fornecido. As regras criadas também devem ser descritas textualmente no trabalho a ser entregue**

Solução.

1. Empregados devem ter no minímo 18 anos.
2. O email do empregado e cliente deve contener '@' e '.'

3. O telefono do empregado e cliente devex começar com '+'

2. **Implementar triggers que garantam a validação das regras semânticas criadas.**

Solução.

```
-- Q2 Parte 2

-- 1. Empregados devem ter no minimo 18 anos
USE Chinook;
delimiter //

CREATE TRIGGER EmployeeInsert
    BEFORE INSERT
    ON Employee
    FOR EACH row
BEGIN
    IF TIMESTAMPDIFF(YEAR, NEW.BirthDate, CURDATE()) >18
    THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Employee must be 16 or older';
    END IF;
END;

delimiter ;

INSERT INTO `Employee` (`EmployeeId`, `LastName`, `FirstName`,
`Title`, `BirthDate`, `HireDate`, `Address`, `City`,
`State`, `Country`, `PostalCode`, `Phone`, `Fax`, `Email`)
VALUES (2007, N'Adams', N'Andrew', N'General Manager',
'2010/2/18', '2002/8/14', N'11120 Jasper Ave NW', N'Edmonton',
N'AB', N'Canada', N'T5K 2N1', N'+1 (780) 428-9482',
N'+1 (780) 428-3457', N'andrew@chinookcorp.com');

delimiter ;

-- 2. O email do empregado e cliente deve contener '@' e '.'
CREATE TRIGGER EmailInsertEmployee
    BEFORE INSERT
    ON Employee
    FOR EACH ROW
BEGIN
    IF NEW.Email NOT LIKE '%_@%_._%' THEN
        SIGNAL SQLSTATE VALUE '45000'
            SET MESSAGE_TEXT = ' `email` column is not valid';
    END IF;
END ;
```

```
CREATE TRIGGER EmailInsertCustumer
    BEFORE INSERT
    ON Customer
    FOR EACH ROW
BEGIN
    IF NEW.Email NOT LIKE '%_@%_._%' THEN
        SIGNAL SQLSTATE VALUE '45000'
            SET MESSAGE_TEXT = ''email' column is not valid';
    END IF;
END ;

INSERT INTO 'Employee' ('EmployeeId', 'LastName', 'FirstName',
 'Title', 'BirthDate', 'HireDate', 'Address', 'City',
'State', 'Country', 'PostalCode', 'Phone', 'Fax', 'Email')
VALUES (2009, N'Adams', N'Andrew', N'General Manager',
'2000/2/18', '2002/8/14', N'11120 Jasper Ave NW', N'Edmonton',
N'AB', N'Canada', N'T5K 2N1', N'+1 (780) 428-9482',
N'+1 (780) 428-3457', N'andrewchinookcorp.com');

-- 3.O telefono do empregado e cliente deve comencar com '+'
CREATE TRIGGER PhoneInsertEmployee
    BEFORE INSERT
    ON Employee
    FOR EACH ROW
BEGIN
    IF NEW.Phone NOT LIKE '%_+_%' THEN
        SIGNAL SQLSTATE VALUE '45000'
            SET MESSAGE_TEXT = ' 'phone' column is not valid';
    END IF;
END ;

CREATE TRIGGER PhoneInsertCustumer
    BEFORE INSERT
    ON Customer
    FOR EACH ROW
BEGIN
    IF NEW.Phone NOT LIKE '_+_%' THEN
        SIGNAL SQLSTATE VALUE '45000'
            SET MESSAGE_TEXT = ' 'phone' column is not valid';
    END IF;
END ;

INSERT INTO 'Employee' ('EmployeeId', 'LastName', 'FirstName',
'Title', 'BirthDate', 'HireDate', 'Address', 'City',
'State', 'Country', 'PostalCode', 'Phone', 'Fax', 'Email')
VALUES (2009, N'Adams', N'Andrew', N'General Manager',
'2000/2/18', '2002/8/14', N'11120 Jasper Ave NW', N'Edmonton',
```

```
N'AB', N'Canada', N'T5K 2N1', N'1 (780) 428-9482',
N'1 (780) 428-3457', N'andrewch@inookcorp.com');
```

3. **Implementar procedimentos armazenados (stored procedures) que garantam a validação das regras semânticas criadas. Nesse caso, o mecanismo de permissões deve ser utilizado para criar um usuário que somente tenha acesso à manipulação dos dados envolvidos através do procedimento definido**

Solução.

```
-- Q3 Parte 2
USE Chinook;

-- 1. Empregados devem ter no minImo 18 anos
-- 2. O email do empregado e cliente deve contener '@' e '.'
-- 3. O telefono do empregado e cliente devex comenar com '+'
DROP PROCEDURE p_insert_employee;

CREATE PROCEDURE p_insert_employee(
    `EmployeeId` int(11),
    `LastName` varchar(20),
    `FirstName` varchar(20),
    `Title` varchar(30),
    `ReportsTo` int(11),
    `BirthDate` datetime,
    `HireDate` datetime,
    `Address` varchar(70),
    `City` varchar(40),
    `State` varchar(40),
    `Country` varchar(40),
    `PostalCode` varchar(10),
    `Phone` varchar(24),
    `Fax` varchar(24),
    `Email` varchar(60))
BEGIN

    IF Email LIKE '%_@%_.__%'
        AND TIMESTAMPDIFF(YEAR, BirthDate, CURDATE()) > 18
        AND Phone LIKE '_+_%'
    THEN
        INSERT INTO Employee (EmployeeId,
        LastName, FirstName, Title, ReportsTo,
        BirthDate, HireDate, Address, City,
        State, Country, PostalCode, Phone,
        Fax, Email)
        VALUES (EmployeeId, LastName,
        FirstName, Title, ReportsTo,
```

```
            BirthDate, HireDate, Address,
            City, State, Country, PostalCode,
            Phone, Fax, Email);
        ELSE
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Incorrect data';
        END IF;

    end;

    DROP PROCEDURE p_insert_customer;

    CREATE PROCEDURE p_insert_customer(
        `FirstName` varchar(40),
        `LastName` varchar(20),
        `Company` varchar(80),
        `Address` varchar(70),
        `City` varchar(40),
        `State` varchar(40),
        `Country` varchar(40),
        `PostalCode` varchar(10),
        `Phone` varchar(24),
        `Fax` varchar(24),
        `Email` varchar(60),
        `SupportRepId` int(11),
        `CustomerID` int(11)
    )
    BEGIN
        IF Email LIKE '%_@%_.__%' AND Phone LIKE '_+_%'
        THEN
            INSERT INTO Customer(CustomerID,
            SupportRepId, Email, Fax, Phone,
            PostalCode,Country, State, City,
            Address, Company, LastName, FirstName)
            VALUES (CustomerID, SupportRepId,
            Email, Fax, Phone, PostalCode,
            Country, State, City, Address,
            Company, LastName, FirstName);
        ELSE
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Incorrect data';
        end if;
    end;
```

4. **A base original do Chinook possui uma coluna Total na tabela Invoice representada de forma redundante com as informações contidas nas colunas UnitPrice e Quantity na tabela InvoiceLine. Podemos identificar nesse caso uma regra semântica onde o**

valor Total de um Invoice deve ser igual à soma de UnitPrice * Quantity de todos os registros de InvoiceLine relacionados a um Invoice. Implementar uma solução que garanta a integridade dessa regra

Solução.

```
-- Q2
USE Chinook;

DROP TRIGGER IF EXISTS invoiceLineInsert;
CREATE TRIGGER invoiceLineInsert
    AFTER INSERT
    ON InvoiceLine
    FOR EACH row
BEGIN
    UPDATE Invoice
    SET Total = Total + (NEW.UnitPrice * NEW.Quantity)
    WHERE NEW.InvoiceId = InvoiceId;
END;

DROP TRIGGER IF EXISTS invoiceLineDelete;

delimiter //

CREATE TRIGGER invoiceLineDelete
    AFTER DELETE
    ON InvoiceLine
    FOR EACH row
BEGIN
    UPDATE Invoice
    SET Total = Total - (OLD.UnitPrice * OLD.Quantity)
    WHERE OLD.InvoiceId = InvoiceId;
END;

DROP TRIGGER IF EXISTS invoiceLineUpdate;
CREATE TRIGGER invoiceLineUpdate
    AFTER UPDATE
    ON InvoiceLine
    FOR EACH row
BEGIN
    UPDATE Invoice
    SET Total = Total - (OLD.UnitPrice * OLD.Quantity)
    + (NEW.UnitPrice * NEW.Quantity)
    WHERE NEW.InvoiceId = InvoiceId;
END;
```

5. **Implemente uma solução para geração de números sequenciais sem buracos na numeração ao considerar apenas as transações concluídas com sucesso que solicitaram**

um número. Atenção: não é possível obter o comportamento exigido usando sequences. Esta solução deve se basear nos conceitos apresentados sobre controle de concorrência para obter o resultado solicitado. Dica: pesquisar o funcionamento do SELECT ... FOR UPDATE no SGBD escolhido. Junto da solução deverá ser entregue um cenário de teste demonstrando o funcionamento da solução

Solução.

```
#Q5 Parte 2
CREATE TABLE sequence
(
    seq_name    varchar(20) unique not null,
    seq_current int unsigned       not null
);

delimiter //
CREATE PROCEDURE currval(IN seqname varchar(20))
BEGIN
    SELECT seq_current
    FROM sequence WHERE seq_name = seqname;
end //

CREATE PROCEDURE nextval(IN seqname varchar(20))
BEGIN
    UPDATE sequence
    SET seq_current = (@next := seq_current + 1)
    WHERE seq_name = seqname;
end //

CREATE PROCEDURE newSequence(
    IN seqname varchar(20), IN startWith int)
BEGIN
    INSERT INTO sequence values (seqname, startWith);
end //

delimiter //

CALL newSequence('Prueba1', 1);
CALL newSequence('Prueba2', 1);

SELECT *
FROM sequence;

CALL currval('Prueba1');
CALL nextval('Prueba1');
CALL currval('Prueba1');
CALL nextval('Prueba2');
CALL nextval('Prueba2');
CALL nextval('Prueba2');
```

```
CALL currval('Prueba2');

SELECT *
FROM sequence;

CALL newSequence('Prueba2');
```