SQL

Explicación de Práctica

Resumen

- Vistas
- Triggers
- Store Procedures

VISTAS

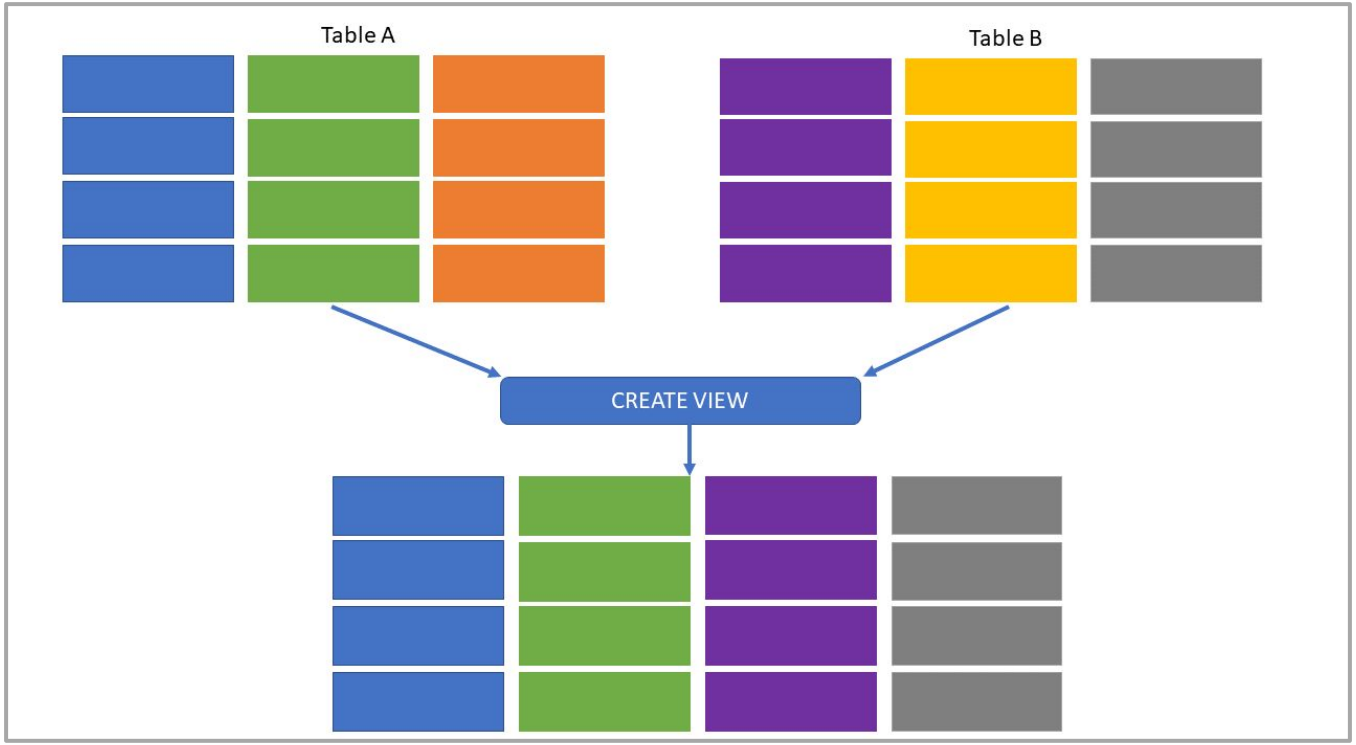
Vistas

- Normalmente el modelo físico puede complejizarse.
- Muchas veces no se quiere que un usuario acceda a todo este modelo.
- Puede darsele al usuario una versión alternativa del modelo que se ajuste a sus necesidades de consulta.

Vistas

- Una vista es una representación lógica de los datos almacenados en una o varias tablas.
- No almacena datos físicos en sí misma sino que es una consulta predefinida que proporciona una vista virtual a los datos existentes en las tablas.
- Son el resultado de una consulta que puede involucrar una o varias tablas.
- No se pueden modificar los datos, sólo pueden realizarse consultas.

Vistas



Vistas

```
CREATE VIEW <nombre> AS <consulta>
```

Estructura de la consulta para crear una vista

```
CREATE VIEW productos_por_categoria
AS
SELECT c.id_categoria as id_categoria, c.nombren as nombre, COUNT(*) as cant
FROM producto p INNER JOIN categoria c ON p.id_categoria == c.id_categoria
GROUP BY c.id_categoria, c.nombre
```

Esta vista muestra las categorías, su nombre y la cantidad de productos

Vistas - Ejercicios

Dado el siguiente esquema veamos algunas posibles vistas que se puedan generar:

PATIENT (patient_id , patient_name, patient_address, patient_city, primary_phone, secondary_phone)

DOCTOR (doctor_id, doctor_name, doctor_address, doctor_city, doctor_speciality)

APPOINTMENT (patient_id, appointment_date, appointment_duration, contact_phone, observations, payment_card)

MEDICAL_REVIEW (patient_id, appointment_date, doctor_id)

PRESCRIBED_MEDICATION (patient_id, appointment_date, medication_name)

Vistas - Ejercicios

Crear una vista que muestre revisiones médicas completas con nombre del paciente, nombre del doctor, fecha y especialidad.

```
CREATE VIEW complete_medical_reviews AS
SELECT
    p.patient_name,
    d.doctor_name,
    d.doctor_speciality,
    mr.appointment_date
FROM
    medical_review mr
INNER JOIN
    patient p ON mr.patient_id = p.patient_id
INNER JOIN
    doctor d ON mr.doctor_id = d.doctor_id;
```

Vistas - Ejercicios

La vista creada se puede utilizar para realizar consultas de la misma manera que si se tratara de una tabla.

Obtenga todos los pacientes que han sido atendidos por el doctor ‘Juan’ y sobre la especialidad ‘Cardiologia’

```
SELECT DISTINCT
   patient_name
FROM
   complete_medical_reviews
WHERE
   doctor_name = 'Juan' AND
   doctor_speciality = 'Cardiologia'
```

Triggers

Triggers

- Un trigger es un objeto de la base de datos que se asocia a una tabla y funciona como un disparador de una acción ante un evento.
- Permite automatizar acciones o aplicar una lógica cuando se producen cambios en una tabla.

```
CREATE TRIGGER nombre_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nombre_tabla FOR EACH ROW
BEGIN
    ...
END;
```

Triggers

- {BEFORE | AFTER} indica cuándo se activa el trigger, antes o después de la operación
- {INSERT | UPDATE | DELETE} es la operación que dispara el trigger
- FOR EACH ROW indica que se ejecutará una vez por cada tupla afectada
- BEGIN y END delimitan el cuerpo del trigger con las operaciones

```
CREATE TRIGGER nombre_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nombre_tabla FOR EACH ROW
BEGIN
    ...
END;
```

Triggers

Cuando el evento disparador es UPDATE, se puede usar:

- *OLD.nombreColumna*: hace referencia al valor antes de actualizarse de la columna en cuestión)
- *NEW.nombreColumna*: hace referencia al valor después de actualizarse de la columna en cuestión)

Triggers

Cuando el evento disparador es INSERT, se puede usar:

- *NEW.nombreColumna*

Cuando el evento disparador es DELETE, se puede usar:

- *OLD.nombreColumna*

Triggers - Ejercicios

Dado el siguiente esquema veamos alguns posibles triggers que se puedan generar:

PATIENT (patient_id , patient_name, patient_address, patient_city, primary_phone, secondary_phone)

DOCTOR (doctor_id, doctor_name, doctor_address, doctor_city, doctor_speciality)

APPOINTMENT (patient_id, appointment_date, appointment_duration, contact_phone, observations, payment_card)

MEDICAL_REVIEW (patient_id, appointment_date, doctor_id)

PRESCRIBED_MEDICATION (patient_id, appointment_date, medication_name)

Vistas - Ejercicios

Si al insertar una nueva cita (APPOINTMENT) no se proporciona un contact_phone, el trigger debe tomar el primary_phone del paciente y colocarlo como contacto.

```
CREATE TRIGGER trg_set_contact_phone BEFORE INSERT ON appointment
FOR EACH ROW
BEGIN
    IF NEW.contact_phone IS NULL THEN
        DECLARE sec_phone VARCHAR(20);

        SELECT secondary_phone INTO sec_phone
        FROM patient
        WHERE patient_id = NEW.patient_id;

        SET NEW.contact_phone = sec_phone;
    END IF;
END$$
```

Store Procedures

Store Procedures

- Un Stored Procedure es un bloque de código SQL que se almacena en la base de datos y puede ser ejecutado repetidamente.
- Se utilizan para realizar tareas específicas, como consultas complejas, operaciones tupla a tupla, etc.
- Se almacenan en el servidor, de manera pre-compilada, y se pueden invocar desde aplicaciones externas o directamente desde el servidor.

Store Procedures

- Se crea indicando nombre, parámetros y cuerpo del procedimiento.
- Es necesario redefinir el delimitador para evitar confusiones entre los delimitadores del SP y el estándar.
- Se lo invoca mediante la sentencia CALL con su nombre y argumentos

```
DELIMITER //
CREATE PROCEDURE <nombre> ( <lista-de-parametros> )
BEGIN
  <lista-de-sentencias-sql>;
END
//
DELIMITER ;
```

```
CALL <nombre> ( <lista-de-argumentos> )
```

Store Procedures

- El contenido de un store procedure puede incluir:
  - Sentencias SQL
  - Declaración de variables locales
  - Uso de estructuras de control: condicionales, repetitivas, manejo de excepciones
  - Llamadas a otros procedimientos
  - Manejo de transacciones.

Store Procedures

- Se utiliza DECLARE para crear una variable, seguido de un identificador y tipo, y se puede dar un valor inicial con DEFAULT.

```
DECLARE <nombre> <tipo> DEFAULT <valor-inicial>;
```

```
DECLARE precio_minimo DECIMAL(3, 2) DEFAULT 100.00;
```

Store Procedures

- Los Stored Procedures pueden recibir parámetros de tres tipos:
  - Entrada (IN): se puede usar y modificar su valor dentro del SP, pero los cambios no se verán reflejados fuera de este
  - Salida (OUT): Se debe asignar un valor dentro del SP, y puede usarse dentro del mismo. Los cambios se verán reflejados fuera del SP
  - Entrada/Salidad (INOUT): Se puede usar y modificar su valor dentro del SP, y los cambios se verán reflejados fuera de este
- Todo parámetro será una variable local en el cuerpo del SP.

Store Procedures

- Los Stored Procedures también poseen:
  - Estructuras de control condicionales como IF y CASE
  - Estructuras de control repetitivas como LOOP, WHILE o REPEAT-UNTIL
  - Manejadores de excepciones
  - Llamados a funciones, definidas por el usuario o bien predefinidas por el DBMS como LAST_INSERT_ID() o NOW()
  - Cursores
  - Manejo de transacciones

Store Procedure - Ejercicios

Dado el siguiente esquema veamos alguns posibles triggers que se puedan generar:

PATIENT (patient_id , patient_name, patient_address, patient_city, primary_phone, secondary_phone)

DOCTOR (doctor_id, doctor_name, doctor_address, doctor_city, doctor_speciality)

APPOINTMENT (patient_id, appointment_date, appointment_duration, contact_phone, observations, payment_card)

MEDICAL_REVIEW (patient_id, appointment_date, doctor_id)

PRESCRIBED_MEDICATION (patient_id, appointment_date, medication_name)

Store Procedures - Ejercicios

Obtener cantidad de citas de un paciente específico, pasado como parámetro al store procedure

```
DELIMITER //

CREATE PROCEDURE obtener_cantidad_de_citas (
    IN p_patient_id INT,
    OUT p_appointment_count INT
)
BEGIN
    SELECT COUNT(*)
    INTO p_appointment_count
    FROM APPOINTMENT
    WHERE patient_id = p_patient_id;
END //

DELIMITER ;
```

Store Procedures - Ejercicios

Registrar una revisión médica (MEDICAL_REVIEW) de un paciente con un determinado doctor, y al mismo tiempo agregar un medicamento prescrito (PRESCRIBED_MEDICATION)

```
CREATE PROCEDURE registrar_medical_review_con_medication (
    IN p_patient_id INT,
    IN p_doctor_id INT,
    IN p_appointment_date DATE,
    IN p_medication_name VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        ROLLBACK;

    START TRANSACTION;

    INSERT INTO MEDICAL_REVIEW (patient_id, appointment_date, doctor_id)
    VALUES (p_patient_id, p_appointment_date, p_doctor_id);

    INSERT INTO PRESCRIBED_MEDICATION (patient_id, appointment_date,
                                                    medication_name)
    VALUES (p_patient_id, p_appointment_date, p_medication_name);

    COMMIT;
END //
```