**BÁO CÁO SƠ BỘ BÀI TẬP LỚN SỐ 2 MÔN HỆ CƠ SỞ DỮ LIỆU**

**HK231 - Lớp: L06**

**Nhóm**: Chủ quán net

**Danh sách thành viên**:

|  |  |  |
| --- | --- | --- |
| Stt | Họ tên | MSSV |
| 1 | Đỗ Duy Cương | 2210413 |
| 2 | Mai Hoàng Danh | 2110896 |
| 3 | Võ Tá Bảo Long | 2211911 |
| 4 | Huỳnh Nguyên Phúc | 2110451 |
| 5 | Cao Minh Quân | 2112109 |

**Chủ đề**: Quản lý gaming cyber

**1. Thiết kế ERD cập nhật mới nhất**

**2. Trigger – Thủ tục - Hàm**

**2.1 Thủ tục INSERT / UPDATE / DELETE dữ liệu vào 1 bảng dữ liệu**

Bảng dữ liệu: account(id, account\_name, password, account\_balance, time\_created, last\_login)

1. Thủ tục INSERT:

Mô tả thủ tục: Thủ tục này được dùng để thêm các dữ liệu vào bảng  account. Được sử dụng trong trường hợp cần thêm dữ liệu vào bảng account. Các thao tác chính bao gồm kiểm tra xem đã có account nào có account\_name trùng với account\_name thêm vào hay chưa, nếu đã có thì thủ tục trả về giá trị 0, nếu chưa thì thực hiện thao tác thêm dữ liệu. Mỗi tham số sẽ có một giá trị khởi tạo, trong trường hợp sử dụng thủ tục mà không gán giá trị cho tham số bất kỳ, tham số đó sẽ mang giá trị khởi tạo.

Input: @p\_account\_id, @p\_account\_name, @p\_password, @p\_account\_balance, @p\_time\_created, @p\_last\_login.

Output: 0 nếu account được thêm vào có account\_name trùng với account\_name của một account đã tồn tại.

Script:

CREATE PROCEDURE InsertIntoAccount

    @p\_account\_id uuid NEWID(),

    @p\_account\_name VARCHAR(60),

    @p\_password VARCHAR(60),

    @p\_account\_balance integer = 0,

    @p\_time\_created DATE = now(),

    @p\_last\_login DATE = now()

AS

BEGIN

    IF EXISTS (SELECT \* FROM account WHERE account\_name = @p\_account\_name) RETURN 0;

    INSERT INTO account (

        id,

        account\_name,

        password,

        account\_balance,

        time\_created,

        last\_login

    ) VALUES (

        @p\_account\_id,

        @p\_account\_name,

        @p\_password,

        @p\_account\_balance,

        @p\_time\_created,

        @p\_last\_login

    );

END;

EXEC InsertIntoAccount

    @p\_account\_name = 'bestdaxuaBKU',

    @p\_password = '123456',

    @p\_account\_balance = 10000;

1. Thủ tục UPDATE:

Mô tả thủ tục: Thủ tục này được dùng để cập nhật dữ liệu trong bảng  account. Được sử dụng trong trường hợp cần cập nhật dữ liệu của một account trong bảng account dựa trên account\_name của account đó. Các thao tác chính bao gồm tìm account cần cập nhật dựa trên account\_name, kiểm tra các giá trị cập nhật có phải null hay không, nếu không phải null thì tiến hành cập nhật dữ liệu, còn nếu là null thì vẫn giữ giá trị cũ.

Input:  @p\_account\_id, @p\_account\_name, @p\_password, @p\_account\_balance, @p\_time\_created, @p\_last\_login.

Output: None.

Script:

CREATE PROCEDURE UpdateAccount

    @p\_account\_id uuid,

    @p\_account\_name VARCHAR(60),

    @p\_password VARCHAR(60),

    @p\_account\_balance integer = 0,

    @p\_time\_created DATE = now(),

    @p\_last\_login DATE = now()

AS

BEGIN

 UPDATE account

 SET

  account\_name = ISNULL(@p\_new\_account\_name, account\_name),

  password = ISNULL(@p\_new\_password, password),

  account\_balance = ISNULL(@p\_new\_account\_balance, account\_balance),

  last\_login = ISNULL(@p\_new\_last\_login, last\_login)

 WHERE

  account\_name = @p\_account\_name;

END;

EXEC UpdateAccount

    @p\_account\_name = 'bestdaxuaBKU',

    @p\_password = '123456',

    @p\_account\_balance = 10000;

1. Thủ tục DELETE:

Mô tả thủ tục: Thủ tục này được dùng để xóa dữ liệu trong bảng  account. Được sử dụng trong trường hợp cần xóa dữ liệu của một account trong bảng account dựa trên account\_name của account đó. Các thao tác chính bao gồm tìm account cần xóa dựa trên account\_name và tiến hành xóa dữ liệu của account đó ra khỏi bảng.

Input: @p\_account\_name

Output: None.

Script:

CREATE PROCEDURE DeleteFromAccount

    @p\_account\_name VARCHAR(60)

AS

BEGIN

    DELETE FROM account

    WHERE account\_name = @p\_account\_name;

END;

EXEC DeleteFromAccount

    @p\_account\_name = ‘bestdaxuaBKU’;

* 1. **TRIGGER**

1. **Trigger 1**: Tính toán tổng giá trị đơn hàng

**Mô tả trigger**: Trigger này được sử dụng để tính toán lại tổng giá trị của đơn hàng khi có các thao tác thêm sản phẩm vào giỏ hàng, xóa sản phẩm khỏi giỏ hàng hoặc cập nhật số lượng sản phẩm.

CREATE OR ALTER TRIGGER calculate\_invoice\_total\_trigger

ON cart\_product

AFTER INSERT, DELETE, UPDATE AS

BEGIN

DECLARE @invoice\_id UNIQUEIDENTIFIER;

IF EXISTS (SELECT TOP 1 1 FROM inserted)

BEGIN

-- Lấy ID đơn hàng từ bảng invoice\_cart khi có thay đổi

SELECT DISTINCT @invoice\_id = i.invoice\_id

FROM invoice\_cart i

INNER JOIN inserted ins ON i.cart\_id = ins.cart\_id;

-- Cập nhật lại tổng giá trị của đơn hàng

UPDATE invoice

SET total\_value = COALESCE((

SELECT SUM(cp.price \* cp.quantity)

FROM cart\_product cp

WHERE cp.cart\_id = ins.cart\_id

), 0)

WHERE id = @invoice\_id;

END;

END;

1. **Trigger 2**: Kiểm tra giới hạn số lượng sản phẩm

**Mô tả trigger**: Trigger này kiểm tra số lượng sản phẩm trong giỏ hàng sau mỗi lần thêm hoặc cập nhật sản phẩm để đảm bảo không vượt quá giới hạn số lượng cho phép.

CREATE OR ALTER TRIGGER check\_product\_quantity\_limit\_trigger

ON cart\_product

AFTER INSERT, UPDATE

AS

BEGIN

DECLARE @max\_quantity INT;

IF EXISTS (SELECT TOP 1 1 FROM inserted)

BEGIN

-- Lấy giới hạn số lượng sản phẩm từ bảng product

SELECT @max\_quantity = p.available\_slot\_quantity

FROM product p

INNER JOIN inserted ins ON p.id = ins.product\_id;

-- Kiểm tra số lượng sản phẩm và hiển thị thông báo lỗi nếu vượt quá

IF EXISTS (

SELECT 1

FROM inserted

WHERE quantity > @max\_quantity

)

BEGIN

print(’Số lượng sản phẩm vượt quá giới hạn cho phép!’);

ROLLBACK TRANSACTION;

END;

END;

END;

1. **Trigger 3**: Cập nhật số dư tài khoản và kiểm tra số tiền trong giao dịch

**Mô tả trigger**: Trigger này cập nhật số dư tài khoản sau mỗi lần thực hiện giao dịch và kiểm tra xem số dư tài khoản có đủ để thực hiện giao dịch không.

CREATE OR ALTER TRIGGER update\_account\_balance\_trigger

ON transaction\_record

AFTER INSERT

AS

BEGIN

DECLARE @current\_balance INT;

IF EXISTS (SELECT TOP 1 1 FROM inserted)

BEGIN

-- Lấy số dư hiện tại từ bảng account

SELECT @current\_balance = a.account\_balance

FROM account a

INNER JOIN inserted ins ON a.id = ins.account\_id;

-- Tính tổng số tiền mới từ các giao dịch mới được thêm

DECLARE @new\_amount INT;

SELECT @new\_amount = SUM(ins.amount)

FROM inserted ins;

-- Kiểm tra và cập nhật số dư mới trong tài khoản

IF (@current\_balance + @new\_amount) < 0

BEGIN

print(’Số dư tài khoản không đủ để thực hiện giao dịch!’);

ROLLBACK TRANSACTION;

END

ELSE

BEGIN

-- Cập nhật số dư mới và thời gian giao dịch cuối cùng UPDATE account

SET account\_balance = @current\_balance + @new\_amount, last\_transaction\_date = GETDATE()

WHERE id IN (SELECT account\_id FROM inserted);

END;

END;

END;

**2.3 THỦ TỤC**

1. Thủ tục 1

Mô tả thủ tục: <*thủ tục này dùng để làm gì, sử dụng trong trường hợp – usecase – nào, thủ tục này bao gồm những thao tác chính nào*>

Input: <*các thông số nhập, nếu có*>

Output: <*kết quả trả về, nếu có*>

1. Thủ tục 2

*Trình bày tương tự thủ tục 1*

**2.4 HÀM**

1. **Hàm 1**

Mô tả hàm:

Hàm này được thiết kế để tính số năm làm việc dựa vào ngày bắt đầu làm việc.

Sử dụng trong trường hợp:

Hàm này được sử dụng khi cần tính số năm làm việc của một nhân viên, dựa vào ngày bắt đầu làm việc.

Tham số đầu vào (Input):

* **ngay\_bat\_dau\_lam\_viec**: Ngày bắt đầu làm việc của nhân viên.

Kết quả trả về (Output):

* **so\_nam\_lam\_viec**: Số năm làm việc của nhân viên.

Thao tác chính:

* Lấy ngày hiện tại (**ngay\_hien\_tai**).
* Tính số năm làm việc bằng cách trừ năm của **ngay\_bat\_dau\_lam\_viec** từ năm của **ngay\_hien\_tai**.

A computer screen shot of a code

Description automatically generated

DECLARE ket\_qua INT;

SET ket\_qua = TinhSoNamLamViec('2010-01-01');

SELECT ket\_qua AS SoNamLamViec;

Trong ví dụ này, hàm TinhSoNamLamViec sẽ trả về số năm làm việc dựa vào ngày bắt đầu làm việc (trong trường hợp này, là từ '2010-01-01'). Kết quả sẽ được hiển thị là SoNamLamViec.

1. **Hàm 2**

Mô tả hàm:

**Mục đích:**

Hàm này được thiết kế để tính số tháng kể từ lần bảo trì cuối cùng.

Sử dụng trong trường hợp:

Hàm này được sử dụng khi cần biết số tháng đã trôi qua từ lần bảo trì cuối cùng của một thiết bị hoặc hệ thống.

**Tham số đầu vào (Input):**

* ngay\_bao\_tri\_cuoi\_cung: Ngày thực hiện lần bảo trì cuối cùng.

**Kết quả trả về (Output):**

* so\_thang\_troi\_qua: Số tháng kể từ lần bảo trì cuối cùng đến ngày hiện tại.

**Thao tác chính:**

* Lấy ngày hiện tại (ngay\_hien\_tai).
* Tính số tháng kể từ ngay\_bao\_tri\_cuoi\_cung đến ngay\_hien\_tai.

A computer code with black text

Description automatically generated

-- Gọi hàm và lấy kết quả

DECLARE ket\_qua INT;

SET ket\_qua = TinhSoThangTuBaoTriCuoiCung('2022-01-15');

SELECT ket\_qua AS SoThangTroiQua;

Trong ví dụ này, hàm TinhSoThangTuBaoTriCuoiCung sẽ trả về số tháng kể từ lần bảo trì cuối cùng (trong trường hợp này, là từ '2022-01-15') đến ngày hiện tại. Kết quả sẽ được hiển thị là SoThangTroiQua.

1. **Hàm 3**

Mô tả hàm:

**Mục đích:**

Hàm này được thiết kế để tính tổng tiền của một giỏ hàng dựa trên mối quan hệ nhiều nhiều giữa giỏ hàng và sản phẩm.

Sử dụng trong trường hợp:

Hàm này được sử dụng khi cần lấy tổng giá trị của các sản phẩm trong một giỏ hàng cụ thể.

**Tham số đầu vào (Input):**

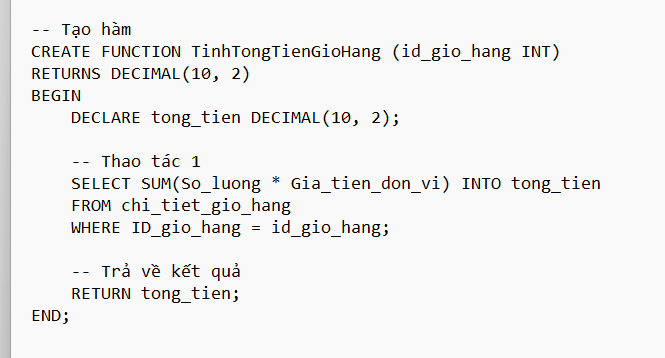
* **id\_gio\_hang**: ID của giỏ hàng cần tính tổng tiền.

**Kết quả trả về (Output):**

* **tong\_tien**: Tổng tiền của giỏ hàng.

**Thao tác chính:**

* Tính tổng tiền bằng cách sử dụng câu lệnh SQL **SELECT SUM(So\_luong \* Gia\_tien\_don\_vi) INTO tong\_tien FROM chi\_tiet\_gio\_hang WHERE ID\_gio\_hang = id\_gio\_hang;**.



-- Gọi hàm và lấy kết quả

DECLARE tong\_tien\_gio\_hang DECIMAL(10, 2);

SET tong\_tien\_gio\_hang = TinhTongTienGioHang(1); -- Thay đổi ID\_gio\_hang tùy thuộc vào giỏ hàng bạn muốn tính tổng tiền.

SELECT tong\_tien\_gio\_hang AS TongTienGioHang;

Trong ví dụ này, hàm **TinhTongTienGioHang** tính tổng tiền của giỏ hàng có **ID\_gio\_hang** bằng 1, và kết quả sẽ được hiển thị là **TongTienGioHang**.

1. **Hàm 4**

Mô tả hàm:

**Mục đích:**

Hàm này được thiết kế để tính số lượng đã bán của một sản phẩm dựa trên mối quan hệ nhiều nhiều giữa giỏ hàng và sản phẩm.

Sử dụng trong trường hợp:

Hàm này được sử dụng khi cần biết số lượng đã bán của mỗi sản phẩm trong hệ thống.

**Tham số đầu vào (Input):**

* **id\_san\_pham**: ID của sản phẩm cần tính số lượng đã bán.

**Kết quả trả về (Output):**

* **so\_luong\_da\_ban**: Số lượng đã bán của sản phẩm.

**Thao tác chính:**

* Tính tổng số lượng đã bán bằng cách sử dụng câu lệnh SQL **SELECT SUM(So\_luong) INTO so\_luong\_da\_ban FROM chi\_tiet\_gio\_hang WHERE ID\_san\_pham = id\_san\_pham;**

A white text with black text

Description automatically generated

-- Gọi hàm và lấy kết quả

DECLARE so\_luong\_da\_ban INT;

SET so\_luong\_da\_ban = TinhSoLuongDaBan(101); -- Thay đổi ID\_san\_pham tùy thuộc vào sản phẩm bạn muốn kiểm tra.

SELECT so\_luong\_da\_ban AS SoLuongDaBan;

Trong ví dụ này, hàm **TinhSoLuongDaBan** tính số lượng đã bán của sản phẩm có **ID\_san\_pham** bằng 101, và kết quả sẽ được hiển thị là **SoLuongDaBan**.