40. Internetový obchod

Projektová dokumentace

Oleksandr Turytsia *(xturyt00)*

Zdebska Kateryna *(xzdebs00)*

1 Zadaní

Cílem je vytvoření jednoduché aplikace pro internetový obchod s určitým druhem zboží např. knihkupectví. Návštěvníci WWW stránek mají možnost prohlížet si veškerý sortiment obchodu, který je členěn do kategorií, ať už daný produkt je skladem či nikoliv. Pokud má návštěvník zájem o určitý produkt, může si jej vybrat (vložení do nákupního košíku). Vybrané zboží si může objednat po zadání potřebných údajů (kontakt, doprava, ...). Zboží si může objednat pouze registrovaný uživatel, pokud uživatel nakupuje poprvé, musí se zaregistrovat a získá přihlašovací jméno a heslo. Tyto údaje může použit k modifikaci osobních informací. Po zaplacení zákazníkem za zboží je považována obchodní transakce za vyřízenou a zaměstnanec obchodu vyexpeduje zboží podle objednávky. Vedení obchodu má informace o celkových tržbách, oblíbenosti zboží, jeho kapacitě, o objednávkách, kdo ji vyřizoval atd.

2 SQL skript pro vytvoření pokročilých objektů schématu databáze

2.1 Triggery

Implementovali jsme 2 databazové triggery

2.1.1 UPDATE_ITEMS_ON_ORDER_CREATE

Jakmile uživatel vytvoří objednávku, všechny položky z jeho košíku (ITEMS) ztratí spojení s košíkem a získají spojení s objednávkou. Pak se vytvoří automaticky platba, která se napojí na uživatele, který objednávku vytvořil.

```sql
CREATE OR REPLACE TRIGGER UPDATE_ITEMS_ON_ORDER_CREATE AFTER
    INSERT ON ORDERS FOR EACH ROW
DECLARE
PAYMENT_ID PAYMENTS.PAYMENT_ID%TYPE;
BEGIN

UPDATE ITEMS
SET
    ORDER_ID = :NEW.ORDER_ID,
    CART_ID = NULL
WHERE
    ITEM_ID = (
        SELECT ITEM_ID FROM CARTS WHERE CARTS.CART_ID = (SELECT CART_ID
FROM CUSTOMERS WHERE CUSTOMERS.CUSTOMER_LOGIN = :NEW.CUSTOMER_LOGIN)
    ) AND ORDER_ID IS NULL;

INSERT INTO PAYMENTS (
    ASSIGNED_TO,
    ACCOUNT_NUMBER,
    IS_PAID,
    ORDER_ID,
    TOTAL_PRICE
) VALUES (
    :NEW.CUSTOMER_LOGIN,
    '1234 3456 3456 3454',
    0,
    :NEW.ORDER_ID,
    :NEW.TOTAL_PRICE
);

END;
```

Kód 1: Trigger UPDATE_ITEMS_ON_ORDER_CREATE

2.1.2 PAYMENT_EXPIRED_CHECK

Jestli platba se změnila, hned spustí se trigger předtím, než změny budou uložené do tabulky. Provede se validace atributu EXPIRED_AT. Pokud dnešní datum je větší než EXPIRED_AT nastane chyba. Kterákoliv změna po platnosti objednávky je chybová.

```
CREATE OR REPLACE TRIGGER PAYMENT_EXPIRED_CHECK BEFORE
    UPDATE ON PAYMENTS FOR EACH ROW
BEGIN
    IF (:NEW.EXPIRES_AT < CURRENT_TIMESTAMP) THEN
        RAISE_APPLICATION_ERROR(-20000, 'Payment token has expired');
    END IF;
END;
```

Kód 2: Trigger Payment_Expired_Check

2.2 Procedury

Vytvořili jsme 2 procedury

2.2.1 AVERAGE_PRODUCT_RATING

Procedura vypočítá průměrné hodnocení produktu na základě jeho recenzí.

Na vstupu procedura bere id produktu, který je třeba vyhodnotit. Pokud tento produkt neměl žádnou recenze, tak vypíše “no feedback was found“. V opačném případě bude výsledkem procedury průměrné hodnocení produktu

V tomto příkladu použili jsme CURSOR pro FEEDBACKy.

```sql
CREATE OR REPLACE PROCEDURE AVERAGE_PRODUCT_RATING (
    P_ID IN PRODUCTS.PRODUCT_ID%TYPE
) AS
    PRODUCTS_COUNT NUMBER;
    SUM_RATING PRODUCTS.TOTAL_RATING%TYPE;
    AVG_RATING PRODUCTS.TOTAL_RATING%TYPE;
    FEEDBACKS_DONT_EXIST EXCEPTION;

    CURSOR CURSOR_FEEDBACKS IS
        SELECT RATING
        FROM FEEDBACKS
        WHERE FEEDBACKS.PRODUCT_ID = P_ID;
BEGIN
    SUM_RATING := 0;
    PRODUCTS_COUNT := 0;

    FOR FEEDBACK IN CURSOR_FEEDBACKS LOOP
        SUM_RATING := SUM_RATING + FEEDBACK.RATING;
        PRODUCTS_COUNT := PRODUCTS_COUNT + 1;
    END LOOP;

    IF PRODUCTS_COUNT = 0 THEN
        RAISE FEEDBACKS_DONT_EXIST;
    END IF;

    AVG_RATING := SUM_RATING / PRODUCTS_COUNT;

    DBMS_OUTPUT.PUT_LINE('AVG rating: ' || AVG_RATING);

    EXCEPTION
        WHEN FEEDBACKS_DONT_EXIST THEN
            DBMS_OUTPUT.PUT_LINE('No feedback was found');
END;
```

Kód 3: Procedura Average_Product_Rating

2.2.2 SHIP_ORDER

SHIP_ORDER procedura je určena pro zaměstnance, aby mohli snadno ověřit objednávku a odeslat ji zákazníkovi.

Procedura ověří zda zákazník zaplatil za objednávku a, že objednávka je zpracovaná. Jestli tyto údaje objednávky jsou správné, změní se stav objednávky na “shipped“ a přiradí zaměstnance, který využil proceduru, a atributu SHIPPED_BY.

```
CREATE OR REPLACE PROCEDURE SHIP_ORDER (
    O_ID IN ORDERS.ORDER_ID%TYPE,
    E_LOGIN IN EMPLOYEES.EMPLOYEE_LOGIN%TYPE
) AS
    STATUS ORDERS.STATUS%TYPE;
    PAYMENT_ID PAYMENTS.PAYMENT_ID%TYPE;
    IS_PAID PAYMENTS.IS_PAID%TYPE;

    INVALID_STATUS EXCEPTION;
    NOT_PAID EXCEPTION;
BEGIN

    SELECT PAYMENT_ID, STATUS
    INTO PAYMENT_ID, STATUS
    FROM ORDERS
    WHERE ORDERS.ORDER_ID = O_ID;

    IF STATUS != 'processing' THEN
        RAISE INVALID_STATUS;
    END IF;

    SELECT IS_PAID
    INTO IS_PAID
    FROM PAYMENTS
    WHERE PAYMENTS.PAYMENT_ID = PAYMENT_ID;

    IF IS_PAID != 1 THEN
        RAISE NOT_PAID;
    END IF;

    UPDATE ORDERS SET STATUS = 'shipped', SHIPPED_BY = E_LOGIN WHERE
ORDERS.ORDER_ID = O_ID;

    EXCEPTION
        WHEN INVALID_STATUS THEN
            DBMS_OUTPUT.PUT_LINE('Order has invalid status of ' || STATUS
|| '. Expected "processing"');
        WHEN NOT_PAID THEN
            DBMS_OUTPUT.PUT_LINE('Order is not paid');
END;
```

Kód 4: Procedura Ship_Order

2.3 Indexování a EXPLAIN PLAN

Rozhodli jsme se, že bude nejlepší indexy zkombinovat s EXPLAIN PLAN, aby bylo vidět zlepšení efektivity selektu.

Vytvořili jsme tento testovací selekt a spustili ho předtím, než se vytvořil index. Tento testovací selekt musí vybrat lidí které nezaplatili za objednávku v intervalu od 2022.01.01 do 2024.01.01. Ve výsledku vypíše se tabulka, ve které je vidět jméno uživatele a celkovou cenu všech objednávek v tomto rozmezí.

```
EXPLAIN PLAN FOR
SELECT FIRST_NAME,
    SUM(TOTAL_PRICE)
FROM PAYMENTS
    INNER JOIN CUSTOMERS ON
        (PAYMENTS.ASSIGNED_TO = CUSTOMERS.CUSTOMER_LOGIN AND
            PAYMENTS.EXPIRES_AT BETWEEN DATE '2022-01-01' AND DATE '2024-
01-01')
GROUP BY FIRST_NAME
ORDER BY FIRST_NAME;
```

Kód 5: Explain plan

Dostali jsme tento výsledek.

Plan hash value: 564673902

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3	1227	7 (15)	00:00:01
1	SORT GROUP BY		3	1227	7 (15)	00:00:01
2	NESTED LOOPS		3	1227	6 (0)	00:00:01
3	NESTED LOOPS		3	1227	6 (0)	00:00:01
* 4	TABLE ACCESS FULL	PAYMENTS	3	453	3 (0)	00:00:01

Tady je vidět, že *CPU per row* v každém řádku je 7, 7, 6, 6, 3. Poslední řádek ukazuje, jakým způsobem fungovalo prohledávání tabulky. **TABLE ACCESS FULL** znamená, že iterovalo se přes každý řádek.

Pak vytvořili jsme index pro datový atribut EXPIRES_AT a spustili stejný select znovu.

```
CREATE INDEX PAYMENT_INDEX ON PAYMENTS(
    EXPIRES_AT
);
```

Kód 6: Index PAYMENT_INDEX

Po spuštění dostali jsme tuto informace.

Plan hash value: 1970997931

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3	1227	6 (17)	00:00:01
1	SORT GROUP BY		3	1227	6 (17)	00:00:01
2	NESTED LOOPS		3	1227	5 (0)	00:00:01
3	NESTED LOOPS		3	1227	5 (0)	00:00:01
4	TABLE ACCESS BY INDEX ROWID BATCHED	PAYMENTS	3	453	2 (0)	00:00:01

CPU per row pro každý řádek je o 1 menší, než byl předtím. Je to proto, že vytvořili index pro atribut EXPIRES_AT, který je využit v podmínce našeho testovacího selektu.

2.4 Přístupová práva k databázovým objektům

Přístupové pravá byly přidané vedoucím týmu XTURYT00 pro druhého člena XZDEBS00 na každou tabulku a proceduru pomocí příkazu **GRANT ALL ON** a **GRANT EXECUTE ON**:

```
GRANT ALL ON XTURYT00.ADDRESSES TO XZDEBS00;
GRANT ALL ON XTURYT00.ADMINS TO XZDEBS00;
GRANT ALL ON XTURYT00.CARTS TO XZDEBS00;
GRANT ALL ON XTURYT00.CUSTOMERS TO XZDEBS00;
GRANT ALL ON XTURYT00.EMPLOYEES TO XZDEBS00;
GRANT ALL ON XTURYT00.FEEDBACKS TO XZDEBS00;
GRANT ALL ON XTURYT00.GUESTS TO XZDEBS00;
GRANT ALL ON XTURYT00.ORDERS TO XZDEBS00;
GRANT ALL ON XTURYT00.PAYMENTS TO XZDEBS00;
GRANT ALL ON XTURYT00.PRODUCTS TO XZDEBS00;
GRANT ALL ON XTURYT00.ITEMS TO XZDEBS00;
GRANT EXECUTE ON AVERAGE_PRODUCT_RATING TO XZDEBS00;
GRANT EXECUTE ON SHIP_ORDER TO XZDEBS00;
```

Kód 7: Přístupová práva

2.5 Materializovaný pohled

Materializovaný pohled je speciální příkaz, který uchovává v sobě výsledek příkazu SELECT.

```sql
CREATE MATERIALIZED VIEW PRODUCT_CATEGORY_COUNT REFRESH ON COMMIT AS
    SELECT CATEGORY,
        COUNT(PRODUCT_ID)
    FROM PRODUCTS
    GROUP BY CATEGORY;
```

Kód 8: Materializovaný pohled

Tento pohled by měl zavolat SELECT který zobrazuje počet jednotlivých produktů které patří do kategorie.

Tímto příkazem vedoucí týmu přidal přístupová práva pro druhého člena týmu

```sql
GRANT ALL ON XTURYT00.PRODUCT_CATEGORY_COUNT TO XZDEBS00;
```

Kód 9: Přístup do materializovaného pohledu

2.6 WITH a CASE

```sql
WITH PRODUCT_AMOUNT_NAME AS (
    SELECT PRODUCT_NAME, PRODUCT_DESC, PRODUCT_IMG, CATEGORY, UNIT_PRICE,
TOTAL_RATING,
        CASE
            WHEN STOCK < 10000 THEN
                'Almost Out of Stock'
            WHEN STOCK >= 10000 AND STOCK <= 20000 THEN
                'Limited Stock'
            WHEN STOCK >= 20000 AND STOCK <= 40000 THEN
                'Sufficient Stock'
            ELSE
                'HIGH STOCK'
        END AS STOCK_NAME
    FROM PRODUCTS
)
SELECT *
FROM PRODUCT_AMOUNT_NAME;
```

Kód 10: Select WITH a CASE

Tento jednoduchý select vypíše jednotlivé produkty: Jméno, popis, obrázek, kategorie, cena za jednotku, hodnocení a počet produktů ve formátu text.