BÀI TẬP LỚN 1

Nguyễn Quốc Khánh
Trường Đại học Công nghệ thông tin
ĐHQG TP.HCM
20521452@gm.uit.edu.vn

Bùi Thị Thanh Ngân
Trường Đại học Công nghệ thông tin
ĐHQG TP.HCM
20521643@gm.uit.edu.vn

Đinh Thí Tú Uyên
Trường Đại học Công nghệ thông tin
ĐHQG TP.HCM
20522139@gm.uit.edu.vn

Nguyễn Ngọc Hải Sơn
Trường Đại học Công nghệ thông tin
ĐHQG TP.HCM
20521846@gm.uit.edu.vn

***Abstract*—Mục tiêu của nghiên cứu này là thiết kế một cơ sở dữ liệu phân tán trên môi trường Oracle 21c, thực hiện phân quyền, query dữ liệu cũng như tạo các function, procedure, trigger, tiến hành thực hiện và demo các mức cô lập và các vấn đề gây mất tính nhất quán, thực hiện tối ưu hóa truy vấn và cuối cùng là tìm hiểu nghiên cứu về tính năng mới của oracle 21c và demo trên môi trường phân tán. (*Abstract*)**

Keywords—oracle, query, optimization, role, database (*key words****)***

I. THIẾT KẾ CƠ SỞ DỮ LIỆU VÀ THỰC HIỆN TRUY VẤN

Lược đồ cơ sở dữ liệu của mỗi cửa hàng điện thoại như sau:

BRANCHES (BranchID, Location)
Tân từ: Mỗi chi nhánh có mã văn phòng (BranchID) là duy nhất, để phân biệt với chi nhánh khác,địa điểm (Location).

CUSTOMERS (CustomerID, CustomerName, Gender, Address, Phone, Birthday, Membership, Spending)
Tân từ: Mỗi khách hàng có mã khách hàng (CustomerID) là duy nhất, tên khách hàng (CustomerName), giới tính của khách hàng (Gender), địa chỉ (Address), số điện thoại của khách hàng (Phone), ngày sinh của khách hàng (Birthday), hạng mức thành viên (Membership), Mức chi tiêu của khách hàng (Spending)

EMPLOYEES (EmployeeID, EmployeeName, Gender, Address, Phone, BranchID, StartDate, Salary)
Tân từ: Mỗi nhân viên có mã nhân viên dung để phân biệt với nhân viên khác (EmployeeID) là duy nhất, tên nhân viên (EmployeeName), giới tính nhân viên (Gender), địa chỉ của nhân viên (Address), số điện thoại của nhân viên (Phone), chi nhánh mà nhân viên đó làm việc (BranchID), ngày vào làm (StartDate), mức lương của nhân viên (Salary).

PHONES (PhoneID, PhoneName, Brand, Color, StorageCapacity, Price, ReleaseDate)
Tân từ: Mỗi loại xe có mã xe (PhoneID) là duy nhất. tên xe (PhoneName), hãng (Brand), giá bán (Price), dung lượng lưu trữ (StorageCapacity), ngày mở bán (ReleaseDate).

PAYMENTS (PaymentID, EmployeeID, CustomerID, PaymentDate, TotalCost)
Tân từ: Mỗi hóa đơn thanh toán có mã thanh toán (PaymentID) là duy nhất để phân biệt với các đơn thanh toán khác, mỗi khách hàng khi thực hiện thanh toán sẽ lưu lại mã khách hàng (CustomerID), mã nhân viên (EmployeeID) tại hóa đơn thanh toán, ngày thanh toán (PaymentDate), tổng số tiền thực hiện thanh toán (TotalCost).

PAYMENT_DETAILS (PaymentID, PhoneID, Quantity)
Tân từ: Mỗi khách hàng khi thực hiện thanh toán xe sẽ lưu lại trong chi tiết hóa đơn thanh toán bao gồm mã thanh toán (PaymentID), mã xe (PhoneID), số lượng điện thoại bán (Quantity).

WAREHOUSE_IMPORTS (BranchID, PhoneID, ImportDate, QuantityImport)
Tân từ: Mỗi mẫu điện được nhập vào một chi nhánh được mô tả thông qua các thuộc tính như mã chi nhánh (BranchID), mã điện thoại (PhoneID), ngày nhập kho (ImportDate), số lượng nhập kho (QuantityImport).

WAREHOUSE_MANAGES (BranchID, PhoneID, Status)
Tân từ: Mỗi mẫu điện thoại được quản lý ở mỗi chi nhánh được mô tả thông qua các thuộc tính như mã chi nhánh (BranchID), mã điện thoại (PhoneID), tình trạng hàng trong kho ‘Còn hàng’ (In stock) hay ‘Hết hàng’ (Out of stock) (Status).

KIẾN TRÚC PHÂN MẢNH
MÔ TẢ PHÂN MẢNH

- Quan hệ BRANCHES là phân mảnh ngang chính
- Quan hệ EMPLOYEES, PAYMENT, PAYMENT_DETAILS là phân mảnh ngang dẫn xuất
- Quan hệ WAREHOUSE được phân mảnh hỗn hợp thành WAREHOUSE_MANAGES và WAREHOUSE_IMPORTS. Trong đó:
    - Quan hệ WAREHOUSE_MANAGES quản lý trạng thái bán hàng của sản phẩm
    - Quan hệ WAREHOUSE_IMPORTS chứa các thông tin dùng để quản lý thông tin nhập hàng của sản phẩm
- Quan hệ CUSTOMERS, PHONES được nhân bản tại tất cả chi nhánh

ĐẠI SỐ QUAN HỆ

- Quan hệ Branches phân mảnh theo Location

BRANCH01 = σLocation= ‘New York’ (BRANCHES)
BRANCH02 = = σLocation= ‘LA’ (BRANCHES)

- Quan hệ Employees, Payments, Payment_Details, Warehouse_Manages, Warehouse_Imports phân mảnh ngang dẫn xuất như sau:

EMPLOYEES01=EMPLOYEES ⋉ BranchID (BRANCH01)
EMPLOYEES02= EMPLOYEES ⋉ BracnhID (BRANCH02)
PAYMENTS01=PAYMENT ⋉ EmployeeID (EMPLOYEES01)
PAYMENTS02= PAYMENT ⋉ EmployeeID (EMPLOYEES02)
PAYMENT_DETAILS01=PAYMENT_DETAILS ⋉ EmployeeID (PAYMENT01)
PAYMENT_DETAILS02= PAYMENT_DETAILS ⋉ EmployeeID(PAYMENT02)
WAREHOUSE_MANAGES01=WAREHOUSE_MANAGES⋉BranchID (BRANCH01)
WAREHOUSE_MANAGES02=WAREHOUSE_MANAGES⋉OfficeID (BRANCH02)
WAREHOUSE_IMPORTS01= WAREHOUSE_IMPORTS⋉BracnhID (BRANCH01)
WAREHOUSE_IMPORTS02= WAREHOUSE_IMPORTS⋉BranchID (BRANCH02)

- Quan hệ PHONES, CUSTOMERS được nhân bản ở cả hai chi nhánh.

XÂY DỰNG CƠ SỞ DỮ LIỆU PHÂN TÁN

- TẠI CHI NHÁNH 1:

```
CREATE TABLE Branches (
   BranchID VARCHAR(50) PRIMARY KEY,
   Location VARCHAR(255)
);
```

```
CREATE TABLE Customers (
   CustomerId VARCHAR(50) PRIMARY KEY,
   CustomerName VARCHAR(255),
   Gender VARCHAR(10),
   Address VARCHAR2(255),
   Phone VARCHAR2(15),
   Birthday DATE,
   Membership VARCHAR(255),
   Spending NUMBER
);
```

```
CREATE TABLE Employees (
   EmployeeID VARCHAR(50) PRIMARY KEY,
   EmployeeName VARCHAR(255),
   Gender VARCHAR(10),
   Address VARCHAR(255),
   Phone VARCHAR(15),
   BranchID VARCHAR(50),
   StartDate DATE,
   Salary NUMBER,
   FOREIGN KEY (BranchID) REFERENCES Branches(BranchID)
);
```

```
CREATE TABLE Phones (
   PhoneID VARCHAR(50) PRIMARY KEY,
   PhoneName VARCHAR(255),
   Brand VARCHAR(255),
   Color VARCHAR(50),
   StorageCapacity NUMBER,
   Price NUMBER,
   ReleaseDate DATE
);
```

```
CREATE TABLE Payments (
   PaymentID VARCHAR(50) PRIMARY KEY,
   EmployeeID VARCHAR(50),
   CustomerID VARCHAR(50),
   PaymentDate DATE,
   TotalCost NUMBER,
   FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);
```

```
CREATE TABLE Payment_Details (
   PaymentID VARCHAR(50),
   PhoneID VARCHAR(50),
   Quantity NUMBER,
   PRIMARY KEY (PaymentID, PhoneID),
   FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID),
   FOREIGN KEY (PhoneID) REFERENCES Phones(PhoneID)
);
```

```
CREATE TABLE Warehouse_Imports (
   BranchID VARCHAR(50),
   PhoneID VARCHAR(50),
   ImportDate DATE,
   QuantityImport NUMBER,
   PRIMARY KEY (BranchID, PhoneID),
   FOREIGN KEY (BranchID) REFERENCES Branches(BranchID),
   FOREIGN KEY (PhoneID) REFERENCES Phones(PhoneID)
);
```

```
CREATE TABLE Warehouse_Manages (
   BranchID VARCHAR(50),
   PhoneID VARCHAR(50),
   Status VARCHAR(50),
   PRIMARY KEY (BranchID, PhoneID),
   FOREIGN KEY (BranchID) REFERENCES Branches(BranchID),
   FOREIGN KEY (PhoneID) REFERENCES Phones(PhoneID)
);
```

- TẠI CHI NHÁNH 2:

```
CREATE TABLE Branches (
   BranchID VARCHAR(50) PRIMARY KEY,
```

```
    Location VARCHAR(255)
);
```

```
CREATE TABLE Customers (
    CustomerId VARCHAR(50) PRIMARY KEY,
    CustomerName VARCHAR(255),
    Gender VARCHAR(10),
    Address VARCHAR2(255),
    Phone VARCHAR2(15),
    Birthday DATE,
    Membership VARCHAR(255),
    Spending NUMBER
);
```

```
CREATE TABLE Employees (
    EmployeeID VARCHAR(50) PRIMARY KEY,
    EmployeeName VARCHAR(255),
    Gender VARCHAR(10),
    Address VARCHAR(255),
    Phone VARCHAR(15),
    BranchID VARCHAR(50),
    StartDate DATE,
    Salary NUMBER,
    FOREIGN KEY (BranchID) REFERENCES Branches(BranchID)
);
```

```
CREATE TABLE Phones (
    PhoneID VARCHAR(50) PRIMARY KEY,
    PhoneName VARCHAR(255),
    Brand VARCHAR(255),
    Color VARCHAR(50),
    StorageCapacity NUMBER,
    Price NUMBER,
    ReleaseDate DATE
);
```

```
CREATE TABLE Payments (
    PaymentID VARCHAR(50) PRIMARY KEY,
    EmployeeID VARCHAR(50),
    CustomerID VARCHAR(50),
    PaymentDate DATE,
    TotalCost NUMBER,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);
```

```
CREATE TABLE Payment_Details (
    PaymentID VARCHAR(50),
    PhoneID VARCHAR(50),
    Quantity NUMBER,
    PRIMARY KEY (PaymentID, PhoneID),
    FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID),
    FOREIGN KEY (PhoneID) REFERENCES Phones(PhoneID)
);
```

```
CREATE TABLE Warehouse_Imports (
    BranchID VARCHAR(50),
    PhoneID VARCHAR(50),
    ImportDate DATE,
    QuantityImport NUMBER,
    PRIMARY KEY (BranchID, PhoneID),
    FOREIGN KEY (BranchID) REFERENCES Branches(BranchID),
    FOREIGN KEY (PhoneID) REFERENCES Phones(PhoneID)
);
```

```
CREATE TABLE Warehouse_Manages (
    BranchID VARCHAR(50),
    PhoneID VARCHAR(50),
    Status VARCHAR(50),
    PRIMARY KEY (BranchID, PhoneID),
    FOREIGN KEY (BranchID) REFERENCES Branches(BranchID),
    FOREIGN KEY (PhoneID) REFERENCES Phones(PhoneID)
);
```

KIẾN TRÚC PHÂN MẢNH

User	Role
Director	CONNECT, DBA
Manager	CONNECT
Staff	CONNECT

❖ Chi nhánh 1:
- Director(Giám đốc): Xem được toàn bộ thông tin của cả hai chi nhánh.
- Manager(Quản lý): Xem được thông tin CUSTOMERS, PHONES , PAYMENTS, PAYMENT_DETAILS của cả hai chi nhánh. Xem được thông tin WAREHOUSE_MANAGES, WAREHOUSE_IMPORTS, EMPLOYEES của chi nhánh mình quản lí.
- Staff(Nhân viên): Xem được thông tin WAREHOUSE_MANAGES, PHONES của chi nhánh mình làm việc.

❖ Chi nhánh 2:
- Manager(Quản lý): Xem được thông tin CUSTOMERS, PHONES, PAYMENTS, PAYMENT_DETAILS của cả hai chi nhánh. Xem được thông tin WAREHOUSE_MANAGES, WAREHOUSE_IMPORTS, EMPLOYEES của chi nhánh mình quản lí.
- Staff(Nhân viên): Xem được thông tin WAREHOUSE_MANAGES, PHONES của chi nhánh mình làm việc

CÂU TRUY VẤN

- **TẠI CHI NHÁNH 1**

Query1: Tài khoản Director: Tìm top 3 nhân viên bán được nhiều điện thoại nhất của cả 2 chi nhánh.

```
(SELECT E1.EmployeeID, E1.EmployeeName, SUM(PD1.Quantity) AS DoanhSo
FROM BRANCH01.Employees E1, BRANCH01.Payments P1, BRANCH01.Payment_Details PD1
WHERE
E1.EmployeeID=P1.EmployeeID
AND P1.PaymentID=PD1.PaymentID
GROUP BY E1.EmployeeID, E1.EmployeeName)
UNION
(SELECT E2.EmployeeID, E2.EmployeeName, SUM(PD2.Quantity) AS DoanhSo
FROM BRANCH02.Employees@DBL_M02 E2, BRANCH02.Payments@DBL_M02 P2, BRANCH02.Payment_details@DBL_M02 PD2
WHERE
E2.EmployeeID=P2.EmployeeID
AND P2.PaymentID=PD2.PaymentID
GROUP BY E2.EmployeeID, E2.EmployeeName)
ORDER BY DOANHSO DESC
    FETCH FIRST 3 ROWS ONLY;
```

Query2: *Tài khoản Director: Giám đốc tìm các khách hàng đã mua tất cả điện thoại có Brand là Apple ở cả hai chi nhánh.*

```
SELECT C1.CustomerID, C1.CustomerName
FROM BRANCH01.CUSTOMERS C1
WHERE NOT EXISTS (SELECT *
 FROM BRANCH01.PHONES PH1
 WHERE PH1.Brand = 'Apple'
 AND NOT EXISTS (SELECT * FROM (
 (SELECT * FROM BRANCH01.PAYMENTS P1, BRANCH01.PAYMENT_DETAILS PD1
 WHERE P1.CustomerID = C1.CustomerID
 AND PD1.PaymentID = P1.PaymentID
 AND PD1.PhoneID = PH1.PhoneID)
 UNION
 (SELECT * FROM
 BRANCH02.PAYMENTS@DBL_M02 P2, BRANCH02.PAYMENT_DETAILS@DBL_M02 PD2
 WHERE P2.CustomerID = C1.CustomerID
 AND PD2.PaymentID = P2.PaymentID
 AND PD2.PhoneID = PH1.PhoneID))));
```

Query3: *Tài khoản Manager: Manager xem số lượng nhập xuất của từng hãng điện thoại tại chi nhánh mình quản lý.*

```
SELECT P.Brand,NVL(SUM(WI.QUANTITYIMPORT), 0) AS SLNhap,
NVL(SUM(PD2.QUANTITY), 0) AS SLXuat
FROM PHONES P
LEFT JOIN WAREHOUSE_IMPORTS WI ON P. PHONEID=WI.PHONEID
LEFT JOIN PAYMENT_DETAILS PD2 ON P. PHONEID=PD2.PHONEID
   GROUP BY P. Brand;
```

Query4: *Tài khoản Director: Tiến hành xem số lượng nhân viên làm việc ở cả chi nhánh 1 và chi nhánh 2*

```
SELECT B1.BRANCHID, COUNT(E1.EMPLOYEEID) AS NUMBEROFEMPLOYEE
FROM BRANCHES B1, EMPLOYEES E1
WHERE B1.BRANCHID = E1.BRANCHID
GROUP BY B1.BRANCHID
UNION
SELECT B2.BRANCHID, COUNT(E2.EMPLOYEEID) AS NUMBEROFEMPLOYEE
FROM BRANCH02.BRANCHES@DBL_M02 B2, BRANCH02.EMPLOYEES@DBL_M02 E2
WHERE B2.BRANCHID = E2.BRANCHID
   GROUP BY B2.BRANCHID;
```

Query5*: Tài khoản Director: Liệt kê danh sách chi nhánh và từng số lượng điện thoại có trong kho của từng chi nhánh (Sử dụng CROSS JOIN , phép nhân)*

```
SELECT B1.BranchID, B1.Location, SUM(WI1.QuantityImport) AS TotalPhonesInStock
FROM Branches B1
CROSS JOIN Phones PH1
LEFT JOIN Warehouse_Imports WI1 ON B1.BranchID = WI1.BranchID AND PH1.PhoneID = WI1.PhoneID
GROUP BY B1.BranchID, B1.Location
UNION
SELECT B2.BranchID, B2.Location, SUM(WI2.QuantityImport) AS TotalPhonesInStock
FROM BRANCH02.Branches@DBL_M02 B2
CROSS JOIN BRANCH02.Phones@DBL_M02 PH2
LEFT JOIN BRANCH02.Warehouse_Imports@DBL_M02 WI2 ON B2.BranchID = WI2.BranchID AND PH2.PhoneID = WI2.PhoneID
   GROUP BY B2.BranchID, B2.Location
```

- **TẠI CHI NHÁNH 2**

Query6**:** *Tài khoản Director: Liệt kê nhân viên và mức lương của nhân viên đó ở cả hai chi nhánh (Phép Hội)*

```
(  SELECT EmployeeId, EmployeeName, Salary
FROM BRANCH01.EMPLOYEES@DBL_M01)
UNION
(SELECT EmployeeId, EmployeeName, Salary
FROM BRANCH02.EMPLOYEES);
```

Query7: Tài khoản Director: Liệt kê những khách hàng mua hàng ở cả hai chi nhánh (Phép giao)

```
SELECT C1.CustomerID, C1.CustomerName
FROM BRANCH01.CUSTOMERS@DBL_M01 C1, BRANCH01.PAYMENTS@DBL_M01 P1
WHERE C1.CustomerID = P1.CustomerID
INTERSECT
SELECT C2.CustomerID, C2.CustomerName
FROM BRANCH02.CUSTOMERS C2, BRANCH02.PAYMENTS P2
WHERE C2.CustomerID = P2.CustomerID;
```

Query8*: Tài khoản Director: Tìm khách hàng chỉ mua điện thoại ở chi nhánh 1 (Phép trừ)*

```
SELECT  C.CustomerID, C.CustomerName
FROM CUSTOMERS C, BRANCH01.PAYMENTS@DBL_M01 P1
WHERE C.CustomerID=P1.CustomerID
MINUS
SELECT C.CustomerID, C.CustomerName
FROM CUSTOMERS C, BRANCH02.PAYMENTS P2
WHERE C.CustomerID=P2.CustomerID;
```

Query9: *Tài khoản "MANAGER": Top 5 sản phẩm điện thoại bán chạy ở chi nhánh 2*

```
SELECT B2.BranchID,P2.PhoneID, P2.PhoneName, SUM(PD2.Quantity) AS TONGBANDUOC
FROM BRANCH02.PHONES P2, BRANCH02.PAYMENT_DETAILS PD2, BRANCH02.BRANCHES B2
WHERE P2.PhoneID = PD2.PhoneID
```

```
GROUP BY P2.PhoneID, P2.PhoneName, B2.BranchID
ORDER BY TONGBANDUOC DESC
FETCH NEXT 5 ROWS ONLY;
```

Query10: *Tài khoản Director: Kiểm tra sản phẩm nào thuộc hãng điện thoại Samsung và có từ 3 sản phẩm được bán của chi nhánh 1 và chi nhánh 2*

```
SELECT P2.PhoneID, P2.PhoneName, P2.Brand, SUM(PD2.Quantity) AS SUM_AMOUNT
FROM BRANCH01.PHONES@DBL_M01 P2, BRANCH01.PAYMENT_DETAILS@DBL_M01 PD2
WHERE P2.Brand = 'Samsung' AND P2.PhoneID = PD2.PhoneID
HAVING SUM(PD2.Quantity)>=3
GROUP BY P2.PhoneID, P2.PhoneName, P2.Brand
UNION
SELECT P2.PhoneID, P2.PhoneName, P2.Brand, SUM(PD2.Quantity) AS SUM_AMOUNT
FROM BRANCH02.PHONES P2, BRANCH02.PAYMENT_DETAILS PD2
WHERE P2.Brand = 'Samsung' AND P2.PhoneID = PD2.PhoneID
HAVING SUM(PD2.Quantity)>=3
GROUP BY P2.PhoneID, P2.PhoneName, P2.Brand;
```

II. VIẾT HÀM, THỦ TỤC VÀ RÀNG BUỘC TOÀN VẸN TRÊN MÔI TRƯỜNG PHÂN TÁN

A. Hàm (Function):

Nhập vào mã khách hàng (CustomerID), thống kê tổng số tiền chi tiêu của khách hàng đó trên toàn bộ hệ thống.

Nội Dung:

- Hàm CalculateAverageOrderValue tính tổng chi tiêu của một khách hàng giữa hai chi nhánh.
- Hàm này truy vấn cả bảng Payments từ chi nhánh địa phương và chi nhánh xa, trả về tổng chi tiêu hoặc 0 nếu không tìm thấy dữ liệu.

```
CREATE OR REPLACE FUNCTION CalculateAverageOrderValue(CUSID VARCHAR2)
  RETURN NUMBER
  AS
    V_TONGTIEN NUMBER;
  BEGIN
    -- Calculate total cost from local and remote Payments table
    SELECT SUM(TOTAL_COST) INTO V_TONGTIEN
    FROM(
      SELECT SUM(P.TotalCost) AS TOTAL_COST
      FROM BRANCH02.Payments P
      WHERE P.CustomerID = CUSID
      UNION ALL
      SELECT SUM(P.TotalCost) AS TOTAL_COST
      FROM BRANCH01.Payments@DBL_M01 P
      WHERE P.CustomerID = CUSID
    );
    RETURN V_TONGTIEN;
  EXCEPTION
    -- Return 0 if no data found
    WHEN NO_DATA_FOUND THEN
      RETURN 0;
    -- Optionally, handle other exceptions
    WHEN OTHERS THEN
      -- Handle other exceptions or re-raise
      RAISE;
  END;
```

B. Thủ tục lưu trữ (Stored Procedure):

Nhập vào mã nhân viên (EmployeeID) và mức lương (Salary) mới , cập nhật mức lương của nhân viên đó theo mức lương được nhập vào.

```
CREATE OR REPLACE PROCEDURE updateEmployeeSalary(empID IN VARCHAR2, sal IN NUMBER) IS
  empExists NUMBER;
BEGIN
  -- Check in Branch02 first
  SELECT COUNT(*) INTO empExists FROM Branch02.Employees WHERE EmployeeID = empID;
  IF empExists > 0 THEN
    -- Employee found in Branch02, update salary
    UPDATE Branch02.Employees SET Salary = sal WHERE EmployeeID = empID;
  ELSE
    -- If not found in Branch02, check in Branch01
    SELECT COUNT(*) INTO empExists FROM Branch01.Employees@DBL_M01 WHERE EmployeeID = empID;
    IF empExists > 0 THEN
      -- Employee found in Branch01, update salary
      UPDATE Branch01.Employees@DBL_M01 SET Salary = sal WHERE EmployeeID = empID;
    END IF;
  END IF;
  -- Only commit if an update has occurred
  IF empExists > 0 THEN
    COMMIT;
  END IF;
END;
```

C. Ràng Buộc Toàn Vẹn (Referential Integrity Constraint):

Trigger 1: Kiểm tra khách hàng phải đủ 15 tuổi khi thực hiện mua hàng trên toàn bộ chi nhánh.

Bối cảnh: Khi INSERT hoặc UPDATE vào bảng CUSTOMERS.

Nội dung:

- Trigger trg_CUS_insert_update được định nghĩa để hoạt động sau khi có lệnh INSERT hoặc UPDATE trên bảng Customers.
- Nếu ngày sinh của khách hàng mới được thêm vào hoặc được cập nhật cho thấy họ chưa đủ 15 tuổi, trigger sẽ ngăn chặn việc thêm hoặc cập nhật thông tin đó và phát sinh một lỗi với mã -20100.

Bảng tầm hưởng:

	Thêm	Xóa	Sửa
CUSTOMER	+(Birthday)	-	+(Birthday)

Khai báo:

```
CREATE OR REPLACE TRIGGER trg_CUS_insert_update
AFTER INSERT OR UPDATE ON Customers
FOR EACH ROW
DECLARE
  today DATE;
BEGIN
  SELECT SYSDATE INTO today FROM DUAL;
      IF EXTRACT(YEAR FROM today) - EXTRACT(YEAR FROM :NEW.Birthday) < 15 THEN
    RAISE_APPLICATION_ERROR(-20100, 'Customers must be at least 15 years old to conduct transactions');
  END IF;
END;
```

Trigger 2: Kiểm tra ngày bắt đầu làm việc của nhân viên phải lớn hơn ngày trong hóa đơn.

Bối cảnh: Trigger CheckDateConstraintPayments được thiết kế để đảm bảo rằng ngày thanh toán (PaymentDate) trong mỗi giao dịch mới hoặc cập nhật giao dịch không xảy ra trước ngày bắt đầu làm việc (StartDate) của nhân viên xử lý giao dịch đó.

Nội dung:

- Trigger CheckDateConstraintPayments đảm bảo ngày thanh toán không trước ngày nhân viên bắt đầu làm việc.
- Áp dụng cho thao tác INSERT và UPDATE trên bảng Payments, tạo lỗi khi vi phạm ràng buộc ngày.

Bảng tầm hưởng:

	Thêm	Xóa	Sửa
Employees	-	-	-
Payments	+(PaymentDate)	-	+(PaymentDate)

Khai báo:

```
REATE OR REPLACE TRIGGER CheckDateConstraintPayments
BEFORE INSERT OR UPDATE ON Payments
FOR EACH ROW
DECLARE
  v_StartDate DATE;
BEGIN
  SELECT StartDate
  INTO v_StartDate
  FROM Employees
  WHERE EmployeeID = :NEW.EmployeeID;
  IF (:NEW.PaymentDate IS NOT NULL AND v_StartDate IS NOT NULL AND :NEW.PaymentDate < v_StartDate) THEN
      RAISE_APPLICATION_ERROR(-20002, 'Payment date must be after the employee''s start date');
  END IF;
END;
```

Trigger 3: Kiểm tra số lượng khi nhập hàng vào kho phải lớn hơn 0.

Bối cảnh: Khi INSERT HOẶC UPDATE vào bảng Warehouse_Imports (Chi nhánh tiến hành nhập hàng)

Nội dung:

- Trigger TRG_VALID_WAREHOUSE_INSERT_UPDATE được thiết lập để hoạt động trước các thao tác INSERT hoặc UPDATE trên bảng Warehouse_Imports, kiểm tra mỗi bản ghi mới hoặc cập nhật để đảm bảo rằng trường QuantityImport luôn lớn hơn 0.
- Nếu QuantityImport không hợp lệ (nhỏ hơn hoặc bằng 0), trigger sẽ ngăn chặn thao tác cập nhật dữ liệu bằng cách phát sinh lỗi với mã -20100 và thông báo "The import quantity is not valid".

Bảng tầm hưởng:

	Thêm	Xóa	Sửa
Warehouse_Imports	+(QuantityImport)	-	+(QuantityImport

Khai báo:

```
CREATE OR REPLACE TRIGGER TRG_VALID_WAREHOUSE_INSERT_UPDATE
BEFORE INSERT OR UPDATE ON Warehouse_Imports
FOR EACH ROW
BEGIN
  IF(:NEW.QuantityImport <= 0)
  THEN
  RAISE_APPLICATION_ERROR(-20100, 'The import quantity is not valid');
  END IF;
END;
```

Trigger 4: Kiểm tra một nhân viên trước khi bị xóa đã từng thực hiện phụ trách hóa đơn cho khách hàng chưa.

Bối cảnh: Khi thực hiện thao tác xóa records từ bảng Employees, phải kiểm tra nhân viên đó đã từng phụ trách giao dịch của khách hàng nào chưa, nếu có tồn thì không được xóa

Nội dung:

- Trigger TRG_CHECK_EMPLOYEE_DELETE sẽ chạy trước khi xóa (BEFORE DELETE) một bản ghi từ bảng Employees, kiểm tra từng nhân viên cụ thể (FOR EACH ROW).
- Nếu nhân viên đó đã có giao dịch trong bảng Payments (SELECT COUNT(*) ... > 0), trigger sẽ không cho phép xóa và sẽ phát sinh một lỗi (RAISE_APPLICATION_ERROR) với thông báo "Cannot delete employee who has existing transactions".

Bảng tầm hưởng:

	Thêm	Xóa	Sửa
Employees	-	+(Kiểm tra)	-
Payments	-	-	-

Khai báo:

```
CREATE OR REPLACE TRIGGER
TRG_CHECK_EMPLOYEE_DELETE
BEFORE DELETE ON Employees FOR EACH ROW
DECLARE
   v_payment_count NUMBER;
BEGIN
   -- Kiểm tra xem nhân viên có liên quan đến giao dịch nào
không
   SELECT COUNT(*)
   INTO v_payment_count
   FROM Payments
   WHERE EmployeeID = :OLD.EmployeeID;
   -- Nếu nhân viên đã tham gia vào giao dịch, không cho
phép xóa
   IF v_payment_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20100, 'Cannot
delete employee who has existing transactions');
   END IF;
END;
```

III. DEMO CÁC MỨC CÔ LẬP TRONG MÔI TRƯỜNG PHÂN TÁN

A. Lost Update

Nhân viên 1 đang thay đổi thông tin của khách hàng thì có nhân viên 2 đến thay đổi thông tin cũng của chính khách hàng đó nhưng với dữ liệu khác. Từ đó, dẫn đến việc thông tin của nhân viên 2 ghi đè lên trên thông tin của nhân viên 1.

Thực thi:

MÁY 1	MÁY 2
	- Tiến hành kiểm tra dữ liệu trên máy SELECT * FROM BRANCH01.CUSTOMERS @DBL_M01
	- May2 tiến hành update tên của khách hàng có mã 01 thành Tu Uyen và commit UPDATE BRANCH01.CUSTOMERS @DBL_M01 SET CustomerName='Tu Uyen' WHERE CUSTOMERID='Cus01';
	COMMIT;
- May1 tiến hành update lại tên khách hàng có mã 01 thành Thanh Ngan và commit UPDATE BRANCH01.CUSTOMERS SET CustomerName='Thanh Ngan' WHERE CustomerID='Cus01';	- Tiến hành kiểm tra dữ liệu trên máy1 SELECT * FROM BRANCH01.CUSTOMERS @DBL_M01
COMMIT;	
Xem lại dữ liệu ta thấy dữ liệu cập nhật ở máy 1 đã mất. Để giải quyết cần phải set mức cô lập serializable cho cả hai máy	

Cách ngăn chặn:

MÁY 1	MÁY 2
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
	- Sau khi set mức cô lập ta thực hiện lại thao tác Update trên máy UPDATE BRANCH01.CUSTOMERS @DBL_M01 SET CustomerName='Tu Uyen' WHERE CustomerID='Cus01'
	COMMIT;
UPDATE BRANCH01.CUSTOMERS SET CustomerName='Thanh Ngan' WHERE CustomerID='Cus01'	
Tại máy 1 sẽ xuất hiện error can't serialize access for this transaction. Ta cần tiến hành COMMIT lại	
SELECT * FROM BRANCH01.CUSTOMERS;	
Tiến hành xem lại dữ liệu ta thấy dữ liệu ở máy 1 cập nhật đã có và có thể tiến hành cập nhật lại dữ liệu ở máy 2 nếu muốn →Vấn đề Lost Update đã được giải quyết	

B. Unrepeatable Data

Mô tả tình huống: Nhân viên 1 xem thông tin khách hàng lần 1 hoàn tất thì nhân viên 2 truy cập vào hệ thống để thay đổi thông tin khách hàng. Sau đó, nhân viên 1 quay lại để kiểm tra thông tin thì nhận thấy có sự thay đổi so với lần xem đầu tiên.

Thực thi:

MÁY 1	MÁY 2
- Đọc dữ liệu khách hàng SELECT * FROM BRANCH01.CUSTOMERS	
	- Cập nhật dữ liệu cho khách hàng có mã KH02 UPDATE BRANCH01.CUSTOMERS @DBL_M01 SET CustomerName='Thanh Ngan' WHERE CustomerID='Cus02';
	COMMIT;
Tiến hành đọc lại dữ liệu ở máy 1 thì ta thấy dữ liệu đã bị thay đổi do cập nhật từ máy 2. Để giải quyết vấn đề này ta cần set mức cô lập thành Serializable	

Cách ngăn chặn:

MÁY 1	MÁY 2
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
	- Tiến hành Update dữ liệu tại máy 2 một lần nữa UPDATE BRANCH01.CUSTOMERS @DBL_M01 SET CustomerName='Ngan' WHERE CustomerID='Cus02';
	COMMIT;
- Đọc dữ liệu bảng khách hàng một lần nữa và thấy dữ liệu vẫn giữ nguyên với Cus2 tên Tu	
- Tiến hành COMMIT lại và đọc lại dữ liệu SELECT * FROM BRANCH01.CUSTOMERS	
Lúc này ta đã thấy Cus02 đã được đổi thành Ngan→Vấn đề đã được giải quyết	

C. Phantom

Mô tả tình huống: Nhân viên 1 kiểm tra thông tin số lượng khách hàng của chi nhánh. Sau đó nhân viên 2 xóa thông tin của một khách hàng. Khi nhân viên 1 kiểm lại thì thấy thông tin các khách hàng bị mất một khách hàng.

MÁY 1	MÁY 2
SELECT * FROM BRANCH01.CUSTOMERS;	
	- Tiến hành xóa khách hàng có mã Cus01 DELETE FROM BRANCH01.CUSTOMERS @DBL_M01 WHERE CustomerID='Cus01';
	COMMIT;
- Đọc lại dữ liệu lần nữa SELECT * FROM BRANCH01.CUSTOMERS	
Ta thấy nhân viên có mã Cus01 đã bị xóa bởi máy 2 → Đây là vấn đề Phantom hay còn gọi là bóng ma là khi một giao tác trên một tập dữ liệu nhưng có một giao tác khác chèn thêm hoặc xóa đi dữ liệu mà giao tác kia đang quan tâm	

Cách ngăn chặn:

SET TRANSACTION ISOLATION	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
LEVEL SERIALIZABLE;	
- Đọc dữ liệu SELECT * FROM BRANCH01.CUSTOMERS;	
	- Xóa dữ liệu khách hàng có mã Cus02 DELETE FROM BRANCH01.CUSTOMERS @DBL_M01 WHERE CUSTOMERID='Cus02';
	COMMIT;
- Đọc dữ liệu lại lần nữa SELECT * FROM BRANCH01.CUSTOMERS;	
Ta thấy dữ liệu vẫn như cũ không đổi cho đến khi ta tiến hành COMMIT và đọc lại → Vấn đề đã được giải quyết	

D. Deadlock

Mô tả tình huống: Máy 1 tiến hành sửa thông tin của 1 nhân viên bất kỳ (nhân viên 01). Sau đó máy 2 cũng sửa thông tin của nhân viên đó (nhân viên 02) và sửa thông tin của 1 nhân viên khác (nhân viên 02) . Sau đó máy 1 lại tiếp tục tiến hành sửa thông tin của nhân viên 02 .Khi có quá nhiều thao tác thực hiện cùng lúc dẫn đến xuất hiện deadlock.

Thực thi và cách giải quyết deadlock

MÁY 1	MÁY 2
Máy 1: Update thông tin Employee01 đổi EmployeeName thành Quốc Khánh UPDATE BRANCH01.Employees SET EmployeeName='Quoc Khanh' WHERE EmployeeID='Emp01';	
	Máy 2: Update thông tin Employee01 đổi tên thành Hai Son UPDATE BRANCH01.Employees@DBL_M01 SET EmployeeName='Hai Son' WHERE EmployeeID='Emp02';
	Máy 2: Update thông tin Emp01 đổi tên thành Tú Uyên UPDATE BRANCH01.Employees@DBL_M01 SET EmployeeName='Tu Uyen'

	WHERE EmployeeID='Emp01';
Máy 1: Update thông tin Emp02 đổi tên thành Thanh Ngân. UPDATE BRANCH01.Employees SET EmployeeName='Thanh Ngan' WHERE EmployeeID='Emp02';	
Xuất hiện Deadlock do quá nhiều thao tác thực hiện cùng lúc	
Để giải quyết ta tiến hành COMMIT. Xem lại dữ liệu: Ta nhận thấy thao tác cuối ở máy xảy ra deadlock sẽ không được thực hiện	

IV. THỰC HIỆN TỐI ƯU HÓA TRUY VẤN TRÊN MÔI TRƯỜNG PHÂN TÁN MỘT CÂU TRUY VẤN ĐƠN GIẢN

A. Câu truy vấn cần tối ưu:

Thống kê số lượng điện thoại đã bán được của từng nhân viên ở Branch01, giảm dần theo số lượng.

```
SELECT E.EmployeeID, E.EmployeeName,
SUM(PD.Quantity) AS DoanhSo
FROM BRANCHES B, EMPLOYEES E, PAYMENTS
P, PAYMENT_DETAILS PD
WHERE B.BRANCHID=E.BRANCHID AND
E.EmployeeID=P.EmployeeID
   AND P.PaymentID=PD.PaymentID
   AND EXTRACT(YEAR FROM
PAYMENTDATE)=2023
   AND B.BRANCHID ='Branch01'
GROUP BY E.EmployeeID, EmployeeName ORDER
BY DoanhSo DESC;
```

Kết quả câu truy vấn ban đầu:

	EMPLOYEEID	EMPLOYEENAME	DOANHSO
1	Emp11	Ava Robinson	5
2	Emp07	William Taylor	4
3	Emp01	John Williams	4
4	Emp02	Sarah Brown	4
5	Emp05	Daniel White	3
6	Emp04	Emily Davis	3
7	Emp09	Emma Miller	2
8	Emp19	Grace Carter	2
9	Emp18	Joseph Baker	2
10	Emp20	Samuel Coleman	1
11	Emp08	James Martinez	1
12	Emp03	Michael Johnson	1
13	Emp15	Abigail Scott	1
14	Emp14	Ethan King	1
15	Emp06	Olivia Thomas	1
16	Emp16	Daniel Adams	1
17	Emp13	Mia Hall	1
18	Emp17	Madison Allen	1
19	Emp10	Benjamin Garcia	1
20	Emp12	Jackson Wright	1

Thực hiện explain query:

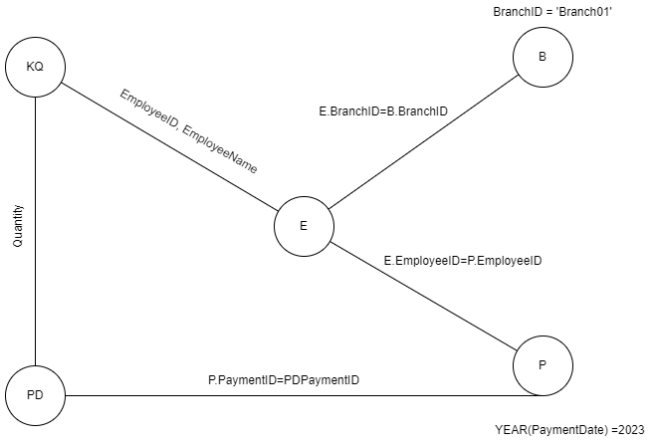
```
EXPLAIN PLAN FOR
SELECT E.EmployeeID, E.EmployeeName,
SUM(PD.Quantity) AS DoanhSo

FROM BRANCHES B, EMPLOYEES E, PAYMENTS
P, PAYMENT_DETAILS PD
WHERE B.BRANCHID=E.BRANCHID AND
E.EmployeeID=P.EmployeeID

   AND P.PaymentID=PD.PaymentID

   AND EXTRACT(YEAR FROM
PAYMENTDATE)=2023

   AND B.BRANCHID ='Branch01'

GROUP BY E.EmployeeID, EmployeeName ORDER
BY DoanhSo DESC;


select * from table(dbms_xplan.display);
```

Kết quả khi Explain câu truy vấn:

```
PLAN_TABLE_OUTPUT
Plan hash value: 3162408958

-----------------------------------------------------------------------------------------
| Id  | Operation             | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                 |    35 | 10010 |    11  (19)| 00:00:01 |
|   1 |  SORT ORDER BY        |                 |    35 | 10010 |    11  (19)| 00:00:01 |
|   2 |   HASH GROUP BY       |                 |    35 | 10010 |    11  (19)| 00:00:01 |
|*  3 |    HASH JOIN          |                 |    35 | 10010 |     9   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL | PAYMENT_DETAILS |    35 |  1400 |     3   (0)| 00:00:01 |
|*  5 |     HASH JOIN         |                 |    35 |  8610 |     6   (0)| 00:00:01 |
|*  6 |      TABLE ACCESS FULL| EMPLOYEES       |    20 |  3660 |     3   (0)| 00:00:01 |
|*  7 |      TABLE ACCESS FULL| PAYMENTS        |    35 |  2205 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("P"."PAYMENTID"="PD"."PAYMENTID")
   5 - access("E"."EMPLOYEEID"="P"."EMPLOYEEID")
   6 - filter("E"."BRANCHID"='Branch01')
   7 - filter(EXTRACT(YEAR FROM INTERNAL_FUNCTION("PAYMENTDATE"))=2023)
```

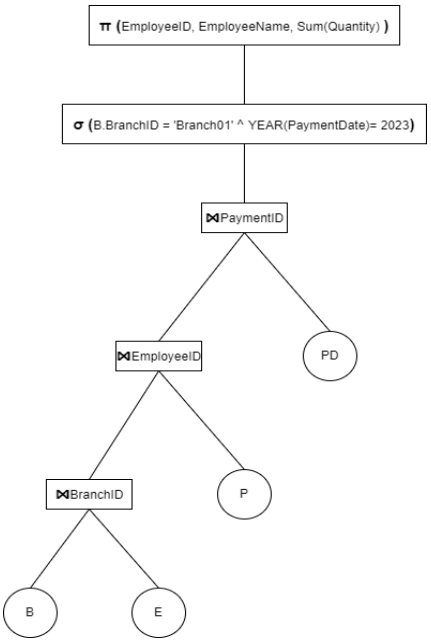
B. Tối ưu hóa câu truy vấn

Kiểm tra câu truy vấn có đúng ngữ nghĩa hay không

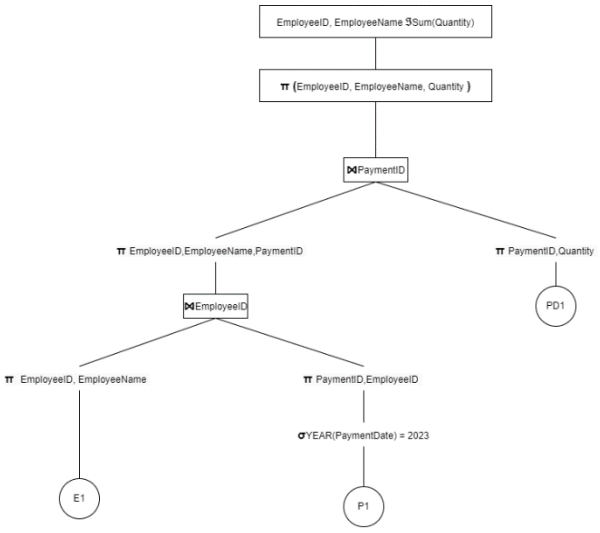


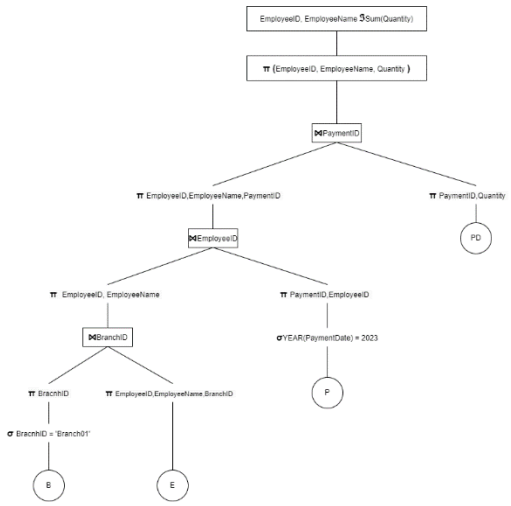
=>Đồ thị liên thông nên câu truy vấn này đúng ngữ nghĩa

Phân rã câu truy vấn để tối ưu hóa toàn cục:
Cây truy vấn quan hệ ban đầu:

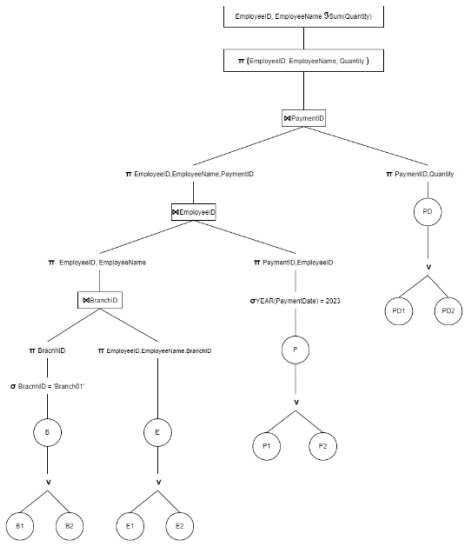


Cây truy vấn sau khi tối ưu hóa toàn cục





Cây truy vấn dựa vào lược đồ phân mảnh:



Cây truy vấn sau khi tối ưu hóa:

C. Câu truy vấn và kết quả sau khi tối ưu hóa

```
SELECT EmployeeID, EmployeeName, SUM(Quantity)
AS SL
FROM(
   SELECT EmployeeID, EmployeeName, Quantity
   FROM
      (SELECT E1.EmployeeID, EmployeeName,
PaymentID
      FROM
         (SELECT EmployeeID, EmployeeName
         FROM EMPLOYEES) E1
         JOIN
         (SELECT PaymentID, EmployeeID
         FROM PAYMENTS
         WHERE EXTRACT(YEAR FROM
PaymentDate)=2023) P1
         ON E1.EmployeeID=P1.EmployeeID
         ) EP1
   JOIN
   (SELECT PAYMENTID, Quantity
   FROM PAYMENT_DETAILS
   ) PD1
   ON EP1.PaymentID=PD1.PaymentID)
GROUP BY EmployeeID, EmployeeName;
```

Explain query sau khi tối ưu hóa:

```
PLAN_TABLE_OUTPUT
Plan hash value: 2679838080

-------------------------------------------------------------------------------...
| Id  | Operation                     | Name            | Rows  | Bytes | Cost (%CPU...
-------------------------------------------------------------------------------...
|   0 | SELECT STATEMENT              |                 |     1 |    57 |     6  (17...
|   1 |  HASH GROUP BY                |                 |     1 |    57 |     6  (17...
|   2 |   NESTED LOOPS                |                 |     1 |    57 |     5   (0...
|   3 |    NESTED LOOPS               |                 |     1 |    57 |     5   (0...
|   4 |     NESTED LOOPS              |                 |     1 |    44 |     4   (0...
|*  5 |      TABLE ACCESS FULL        | PAYMENTS        |     1 |    24 |     3   (0...
|   6 |      TABLE ACCESS BY INDEX ROWID| EMPLOYEES     |     1 |    20 |     1   (0...
|*  7 |       INDEX UNIQUE SCAN       | SYS_C008409     |     1 |       |     0   (0...
|*  8 |     INDEX UNIQUE SCAN         | SYS_C008411     |     1 |       |     0   (0...
|   9 |    TABLE ACCESS BY INDEX ROWID| PAYMENT_DETAILS |     1 |    13 |     1   (0...
-------------------------------------------------------------------------------...

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - filter(EXTRACT(YEAR FROM INTERNAL_FUNCTION("PAYMENTDATE"))=2023)
   7 - access("EMPLOYEEID"="EMPLOYEEID")
   8 - access("PAYMENTID"="PAYMENTID")
```

V. CÁC ĐẶC ĐIỂM MỚI CỦA ORACLE VÀ ỨNG DỤNG PHÂN TÁN TRONG ĐẶC ĐIỂM MỚI

A. Đặc điểm mới của hệ quản trị Oracle 21c

Phiên bản gần như mới nhất và ổn định nhất của cơ sở dữ liệu phổ biến nhất thế giới, Oracle Database 21c (thời điểm hiện tại đã cho ra mắt **Oracle Database 23c vào ngày 19/09/2023),** được phát hành theo tiêu chí ưu tiên cho điện toán đám mây – "cloud first". Phiên bản chạy trên nền tảng Cloud của Oracle được ra mắt vào Tháng Mười Hai 2020, và bản hoạt động trên máy chủ on-premise được phát hành muộn hơn, vào Tháng Tám 2021.

Oracle đã nhất quán áp dụng phương pháp lưu trữ và quản lý dữ liệu trong một CSDL hội tụ, thay vì chia nhỏ thành nhiều thành phần chức năng đơn dụng. Hay nói cách khác, CSDL hội tụ của Oracle là CSDL đa mô hình (multi-model), phục vụ nhiều đối tượng (multi-tenant) và nhiều loại công việc khác nhau (multi-workload). Cơ sở dữ liệu Oracle hỗ trợ đầy đủ nhiều mô hình dữ liệu và phương pháp truy cập, đơn giản hóa việc hợp nhất trong khi đảm bảo tính cô lập và vượt trội trong các trường hợp sử dụng khối lượng công việc cơ sở dữ liệu điển hình – cả hoạt động và phân tích.

Tính năng mới trên Oracle 21c mà nhóm chọn tìm hiểu và áp dụng

SQL Macros

Để xử lý các câu lệnh SQL phức tạp, truy xuất và liên kết nhiều bảng với nhau, các lập trình viên thường sử dụng các hàm và thủ tục để đơn giản hóa. Nhưng điều này có thể khiến hiệu năng xử lý kém đi. Oracle Database 21c giải quyết vấn đề này bằng việc cho phép hàm có thể trả về một SQL Macro thay vì giá trị đơn lẻ. để tạo ra các phần của câu lệnh SQL một cách động.

Điểm đặc biệt của SQL Macros là khả năng của chúng trong việc tạo ra các phần mẫu của câu lệnh SQL, điều này giúp tăng cường khả năng tái sử dụng và giảm thiểu lặp code trong các truy vấn SQL phức tạp.

Tính năng này giảm thiểu việc gọi hàm nhiều lần, đồng thời giảm độ phức tạp của câu lệnh phải viết, cũng như tăng tốc độ xử lý chung, tương tự với việc sử dụng các View Parameterized và bảng Polymorphic.

Có hai loại SQL Macros: Scalar SQL Macros và Table SQL Macros.

1) Scalar SQL Macros
2) Scalar SQL Macros là những hàm mà trả về một giá trị đơn lẻ. Chúng có thể được sử dụng ở mọi nơi trong câu lệnh SQL nơi mà một giá trị scalar có thể được sử dụng.

Ví dụ:

Giả sử ta muốn tính giá sau thuế cho một số lượng lớn các sản phẩm khác nhau, bạn có thể tạo một Scalar SQL Macro như sau:

```
CREATE FUNCTION calculate_tax(sale_price
NUMBER, tax_rate NUMBER)
RETURN NUMBER SQL_MACRO(SCALAR) IS
BEGIN
  RETURN sale_price * (1 + tax_rate);
END;
```

Để sử dụng macro này trong một truy vấn:

```
SELECT product_id, calculate_tax(sale_price, 0.1) AS
price_after_tax
FROM product_sales;
```

Macro này sẽ trả về giá sau thuế cho mỗi sản phẩm.

Table SQL Macros

Table SQL Macros là những hàm mà trả về một bảng (hoặc một tập hợp các dòng). Chúng có thể được sử dụng trong câu lệnh FROM.

Ví dụ:

Ta muốn tạo một bảng macro để chọn top 10 đơn hàng có giá trị cao nhất từ một bảng bán hàng:

```
CREATE FUNCTION top_orders(top_n IN NUMBER)
RETURN VARCHAR2 SQL_MACRO(TABLE) IS
BEGIN
  RETURN 'SELECT * FROM (SELECT * FROM
orders ORDER BY total_amount DESC) WHERE
ROWNUM <= :top_n';
END;
```

Để sử dụng macro này trong một truy vấn:

```
SELECT * FROM TABLE(top_orders(10));
```

Macro này sẽ trả về một bảng chứa 10 đơn hàng có giá trị cao nhất.

Lợi Ích của SQL Macros

1) **Tái Sử Dụng Mã**: Macros giúp tái sử dụng mã SQL và giảm bớt sự trùng lặp.
2) **Bảo Trì Dễ Dàng**: Cập nhật macro tại một nơi duy nhất và thay đổi sẽ được phản ánh khắp nơi macro được gọi.
3) **Hiệu Suất**: SQL Macros được mở rộng tại thời điểm biên dịch câu lệnh SQL, không tạo ra chi phí thêm khi thực thi.
4) **Tính Linh Hoạt**: Bạn có thể truyền tham số vào macros, tạo ra các truy vấn động dựa trên đầu vào.

B. DEMO TRÊN MÔI TRƯỜNG PHÂN TÁN:

a) Macro Tính Tổng Chi Tiêu Khách Hàng (Dựa Trên Hàm CalculateAverageOrderValue): Tạo một SQL Macro để tính tổng chi tiêu của một khách hàng trên cả hai chi nhánh. Macro này sẽ thực thi truy vấn SQL đã được xác định trong hàm CalculateAverageOrderValue và trả về kết quả dưới dạng một câu truy vấn:

- Macro này dùng để tính tổng chi tiêu của một khách hàng dựa trên dữ liệu từ cả hai chi nhánh (local và remote).
- Khi Macro được gọi, nó sẽ tạo ra một câu truy vấn SQL động, kết hợp dữ liệu từ hai bảng Payments ở hai chi nhánh khác nhau.

```
CREATE OR REPLACE FUNCTION
TotalSpendingMacro(customer_id IN VARCHAR2)
RETURN VARCHAR2 SQL_MACRO IS
BEGIN
  RETURN 'SELECT SUM(TotalCost) as
TOTAL_SPEND FROM ('
      || 'SELECT SUM(TotalCost) TotalCost FROM
branch02.Payments p2 WHERE p2.CustomerID = ''' ||
customer_id || ''' '
      || 'UNION ALL '
      || 'SELECT SUM(TotalCost) TotalCost FROM
branch01.Payments@DBL_M01 p1 WHERE
p1.CustomerID = ''' || customer_id || ''''
```

```
        || ')';
END;
-- Thuc thi
DECLARE
  v_SQL VARCHAR2(1000);
  v_TotalSpend NUMBER;
BEGIN
  -- Lấy câu truy vấn SQL từ Macro
  v_SQL := TotalSpendingMacro('Cus01');
  EXECUTE IMMEDIATE v_SQL INTO v_TotalSpend;
  DBMS_OUTPUT.PUT_LINE('Total Spending for Cus01: ' || TO_CHAR(v_TotalSpend));
END;
```

b) Macro Cập Nhật Lương Nhân Viên (Dựa Trên Thủ Tục updateEmployeeSalary): Tạo một SQL Macro cho việc cập nhật lương nhân viên, cho phép thực thi logic của thủ tục updateEmployeeSalary dưới dạng một câu truy vấn SQL:

- Macro này được thiết kế để cập nhật lương của nhân viên dựa trên ID của họ, bất kể họ làm việc ở chi nhánh nào.
- Nó tạo ra hai câu lệnh UPDATE, mỗi câu lệnh tương ứng với một chi nhánh.
- Trong trường hợp nhân viên không tồn tại ở chi nhánh đầu tiên, Macro vẫn tạo ra câu lệnh UPDATE cho chi nhánh thứ hai. Tuy nhiên, nếu không có nhân viên nào được cập nhật, không có giao dịch nào sẽ được thực hiện.

```
CREATE OR REPLACE FUNCTION UpdateSalaryMacro(emp_id VARCHAR2, new_salary NUMBER)
RETURN VARCHAR2 SQL_MACRO IS
BEGIN
  RETURN 'UPDATE Branch02.Employees SET Salary = ' || new_salary ||
         ' WHERE EmployeeID = ''' || emp_id || '''; ' ||
         'UPDATE  Branch01.Employees@DBL_M01  SET Salary = ' || new_salary ||
         ' WHERE EmployeeID = ''' || emp_id || '''';
END;
```

REFERENCES

[1] https://docs.oracle.com/en/database/oracle/oracle-database/21/whats-new.html

[2] https://jprotech.com.vn/2022/04/20/cong-nghe-hoi-tu-tren-phien-ban-oracle-database-21c/

[3] https://docs.oracle.com/cd/B13789_01/appdev.101/b10807/13_elems017.htm

[4] https://oracle-base.com/articles/21c/sql-macros-21c#basics

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.