TRƯỜNG ĐH CÔNG NGHỆ THÔNG TIN TP.HCM
Khoa Hệ Thống Thông Tin
Chương trình đào tạo: Kỹ sư Công nghệ thông tin

Bài Thực hành số 5

Tên môn học: HQTCSDL Oracle

TRANSACTION

Bài thực hành này giúp sinh viên cài đặt PL / SQL phần kiến thức liên quan đến Transaction.

I. Tóm tắt bài thực hành

1.1. Yêu cầu lý thuyết

Sinh viên đã được trang bị kiến thức:

- Các kiến thức cơ bản ngôn ngữ PL / SQL.
- Các khái niệm: Procedure, Function và Trigger trong Oracle.

1.2. Nội dung

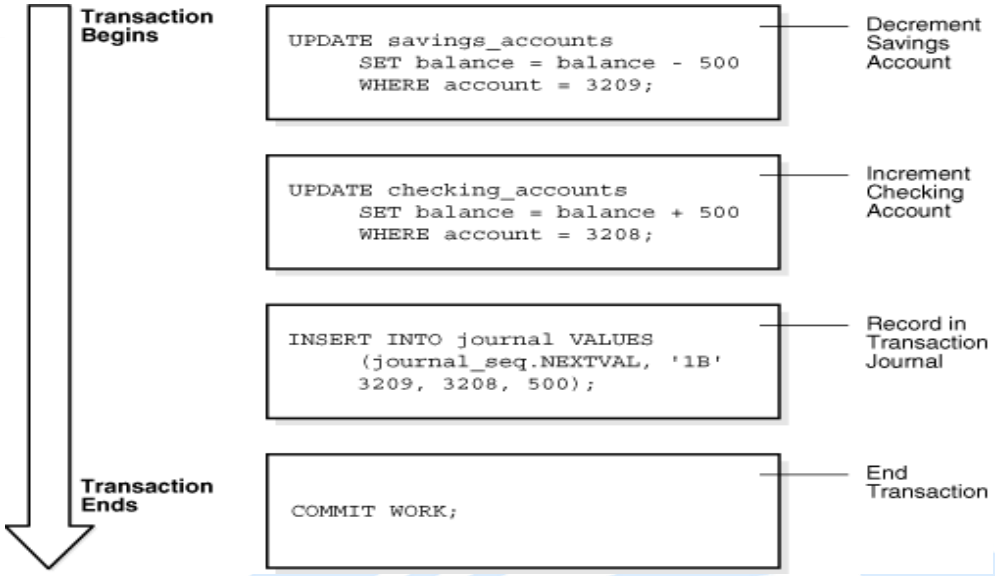
- ❖ Giới thiệu bài toán chuyển tiền ngân hàng
- ❖ Cấu trúc Transaction
- ❖ Transaction Control

II. Giới thiệu bài toán chuyển tiền ngân hàng

Khi khách hàng muốn chuyển tiền từ tài khoản A sang B, thì hệ thống phải thực thi 3 công đoạn nhỏ:

- Giảm tiền tài khoản A
- Tăng tiền tài khoản B
- Ghi chú lại lịch sử giao dịch

Có thể xảy ra 2 trường hợp: tất cả 3 công đoạn đều thành công, dữ liệu lưu xuống database. Hoặc có thể xảy ra trục trặc ở một bước nào đó (do hỏng phần cứng, tài khoản chuyển không đủ tiền, sai số tài khoản …) thì lúc đó hệ thống phải phục hồi lại trạng thái ban đầu để số dư ở tất cả tài khoản đều đúng.



III. Cấu trúc Transaction

- Gồm một hay nhiều câu truy vấn, có thể bao gồm DDL, DML.
- Có bắt đầu và kết thúc.
- *Bắt đầu transaction*
  - Transaction bắt đầu khi câu lệnh SQL đầu tiên được thực thi, bao gồm DDL, DML hoặc lệnh SET TRANSACTION.
  - TRANSACTION NAME: Đặt tên cho transaction, bắt đầu transasction.
  - SET TRANSACTION NAME <Tên transaction> (trước lệnh này nên có 1 lệnh COMMIT).

```
SET TRANSACTION NAME 'ten_transaction';
```

  - Khi một transaction mới bắt đầu, hệ quản trị Oracle sẽ gán nó vào **undo data segment** (ghi nhận lại các thao tác của transaction trước khi commit, để có thể rollback khi có lỗi).
- *Kết thúc transaction*
  - Transaction có thể kết thúc trong nhiều trường hợp.
    - Gặp lệnh COMMIT hoặc ROLLBACK mà không có savepoint.
    - Gặp các câu lệnh DDL như create, drop, rename, alter.
    - User ngắt kết nối đến hệ quản trị đột ngột, transaction sẽ tự động commit.
    - Các ứng dụng đang kết nối đến hệ quản trị bị dừng đột ngột, transaction sẽ tự động rollback.

```
BEGIN
COMMIT;
SET TRANSACTION NAME 'update_salary';
UPDATE EMPLOYEE
SET salary = salary + 500000
where EmpNo = 1;
COMMIT;  -- hoặc ROLLBACK
END;
```

- 1 Transaction đang thực thi là 1 Transaction đã bắt đầu nhưng chưa được COMMIT hoặc ROLLBACK.
- Như ví dụ trên câu lệnh đầu tiên trong Transaction **'update_salary'** là cập nhật lương của nhân viên có mã nhân viên là **1**. Từ lúc thực hiện câu lệnh **update** này cho đến câu lệnh **COMMIT** kết thúc Transaction (khoảng này được gọi là phạm vi Transaction), Transaction **'update_salary'** đã được kích hoạt.

IV. Transaction Control

- Gồm các lệnh để quản lý sự thay đổi của DML lên database, gồm một số lệnh chính:
  - SAVEPOINT: Xác định một điểm trong transaction để rollback về khi có sự cố.
  - COMMIT: Kết thúc transaction, **lưu thay đổi vĩnh viễn**, xóa tất cả SAVEPOINT, mở transaction locks.
  - ROLLBACK: phục hồi lại dữ liệu trước khi thay đổi.
- ROLLBACK
  - Hoàn tác mọi thay đổi
  - Mở tất cả khóa
  - Xóa toàn bộ savepoints
  - Kết thúc transaction

```
BEGIN
   COMMIT;
   set TRANSACTION NAME 'tadda';
   INSERT INTO Employee VALUES
(10,'Name10',TO_DATE('1/1/1998','dd/mm/yyyy'),8,
3,TO_DATE('1/1/2000','dd/mm/yyyy'),2000000,1,1,'Note1','mail10@com.vn');
   INSERT INTO Employee VALUES
(12,'Name12',TO_DATE('1/1/1998','dd/mm/yyyy'),8,
3,TO_DATE('1/1/2000','dd/mm/yyyy'),2500000,1,1,'Note1','mail12@com.vn');
   EXCEPTION WHEN DUP_VAL_ON_INDEX
```

```
    THEN
        ROLLBACK;
        dbms_output.put_line('abc');
END;
```

- COMMIT

```
BEGIN
    UPDATE EMPLOYEE
    set LEVEL_EMPLOYEE = 3
    WHERE EmpNo = 3;
    COMMIT;
    ROLLBACK;
END;
```

- COMMIT lưu giá trị vĩnh viễn?
  - Trong ví dụ trên nếu **không** có lệnh COMMIT thì khi chạy đến lệnh ROLLBACK, level_employee của nhân viên có mã nhân viên = 3 sẽ bị hoàn tác lại lúc chưa cập nhật lên level_employee = 3.
  - Còn khi có lệnh COMMIT ở trước lệnh ROLLBACK thì khi chạy đến lệnh ROLLBACK thì dữ liệu vẫn không thể hoàn tác lại. Vì vậy sau khi chạy xong thì level_employee = 3.
- SAVEPOINTS
  - Savepoints là một điểm được người dùng khai báo trong phạm vi transaction.
  - Savepoints chia một Transaction thành các phần nhỏ hơn.
- ROLLBACK TO SAVEPOINT
  - Việc ROLLBACK tại Savepoint trong Transaction chưa được COMMIT có nghĩa là hoàn tác lại mọi thay đổi được thực hiện sau Savepoint, điều này không có nghĩa là sẽ ROLLBACK lại toàn bộ Transaction.

```
BEGIN
    UPDATE EMPLOYEE
    set Salary = Salary + 500000
    WHERE EmpNo = 1;
    SAVEPOINT diem_1;
    UPDATE EMPLOYEE
    set Salary = 0
    WHERE EmpNo = 1;
    SAVEPOINT diem_2;
    COMMIT;
```

```
END;
```

- Khi một Transaction Rollback lại Savepoint như ROLLBACK TO diem_1 ở ví dụ trên
  - Dữ liệu chỉ được phục hồi lại các câu lệnh ở sau savepoint diem_1, vì vậy mức lương của nhân viên 1 sẽ được hoàn tác lại **trước** khi cập nhật mức lương của nhân viên này = 0.
  - Oracle sẽ lưu trữ lại savepoint được chỉ định trong câu lệnh Rollback to savepoint trong trường hợp này là savepoint 'diem_1', nhưng tất cả savepoint tiếp theo đều bị mất (savepoint diem_2).

- HÀNG ĐỢI TRANSACTIONS: Transaction đang đợi tài nguyên bị khóa sẽ bị block, nó sẽ xếp vào hàng đợi của Transaction đang giữ tài nguyên đó. Transaction đang giữ tài nguyên phải commit hoặc rollback để Transaction bị khóa tiếp tục thực thi.

	Session 1	Session 2	Session 3	Giải thích
	UPDATE employees SET salary=7000 WHERE last_name='Banda';			Session 1 bắt đầu 1 giao tác. Session đặt 1 khóa riêng biệt lên dòng Banda (TX) và khóa bán riêng biệt lên bảng (SX)
	SAVEPOINT after_banda_sal;			Session 1 tạo một samepoints tên là after_banda_sal;
	UPDATE employees SET salary=12000 WHERE last_name= 'Greene';			Session 1 khóa dòng Greene
		UPDATE employees SET salary=14000 WHERE last_name= 'Greene';		Session 2 muốn update dòng Greene nhưng không thành công vì Session 1 đang giữ khóa. Do đó không có giao tác nào bắt đầu tại Session 2.
	ROLLBACK TO SAVEPOINT after_banda_sal;			Session 1 rollback việc cập nhật lương cho dòng Greene, khóa bảng nhận được tại t0 không được nhả. Session 2 vẫn còn bị khóa bởi Session 1 bởi vì

				Session 2 được xếp vào trong hàng đợi trên giao tác của Session 1, giao tác đó vẫn chưa được hoàn tất.
			UPDATE employees SET salary=11000 WHERE last_name= 'Greene';	Dòng Greene đã được mở khóa vì vậy Session 3 nhận khóa giúp cho việc cập nhật tại dòng Greene. Dòng lệnh này bắt đầu giao tác của Session 3.
	Commit;			Session 1 được commit, kết thúc giao tác của Session 1. Session 2 được thêm vào hàng đợi để cập nhật cho dòng Greene sau giao tác của Session 3.

- AUTONOMOUS TRANSACTION
  - AUTONOMOUS TRANSACTION là một Transaction độc lập có thể được gọi từ Transaction chính. Có thể tạm dừng Transaction chính, thực hiện các câu lệnh SQL và COMMIT hoặc ROLLBACK trong Autonomous Transaction và rồi sau đó tiếp tục Transaction chính.
  - Autonomous Transaction có những đặc điểm:
    - Autonomous Transaction không thể thấy được những thay đổi chưa được COMMIT của Transaction chính và không chia sẻ các khóa hoặc tài nguyên với Transaction chính.

```
CREATE OR REPLACE PROCEDURE test_autonomous
AS
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   UPDATE EMPLOYEE
   set EMPLOYEE.SALARY = EMPLOYEE.SALARY + 700000
   WHERE EMPLOYEE.EMPNO = 1;
   ROLLBACK;
END;

BEGIN
   UPDATE EMPLOYEE
   set EMPLOYEE.SALARY = EMPLOYEE.SALARY + 500000
```

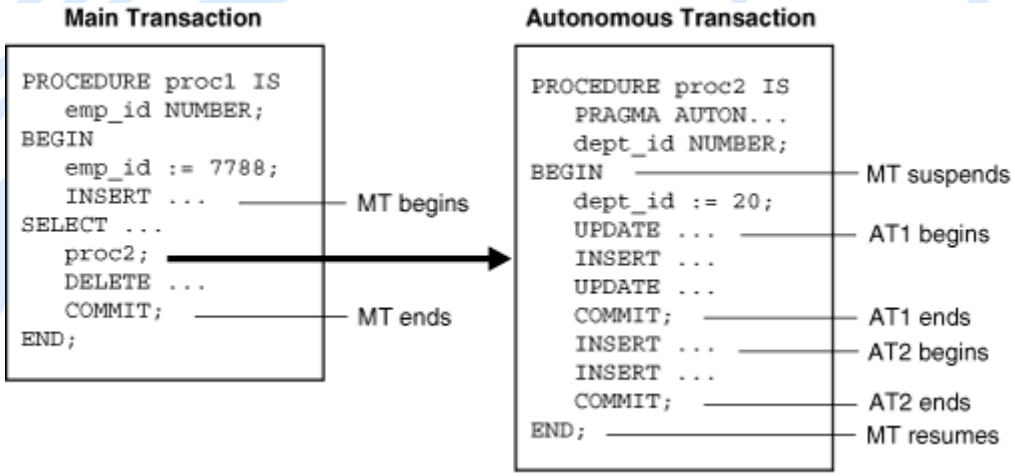
```
    WHERE EMPLOYEE.EMPNO = 1;
    test_autonomous;
    COMMIT;
END;
```

T	Session 1	Session 2	Giải thích
t0	`UPDATE EMPLOYEE set SALARY = SALARY + 500000 WHERE EMPNO = 1;`		Session 1 cập nhật lương cho nhân viên 1, Session 1 đã lock hàng nhân viên 1 này.
t1	`test_autonomous;`		Session 1 (Transaction Chính) gọi procedure *test_autonomous* (procedure này là 1 Autonomous Transaction)
t2		`UPDATE EMPLOYEE set SALARY = SALARY + 700000 WHERE EMPNO = 1;`	Session 2 (*test_autonomous*) cập nhật lương cho nhân viên 1, hiện hàng nhân viên 1 này đang bị khóa bởi Session 1.

t3	`Error at line 1`
	`ORA-00060: deadlock detected while waiting for resource`
	`ORA-06512: at "HR.TEST_AUTONOMOUS", line 5`
	`ORA-06512: at line 5`
			Kết quả là 2 Session này bị deadlock, vì khi Session 1 gọi *test_autonomous* thì Session 1 sẽ bị tạm dừng để cho *test_autonomous* thực thi, nhưng Transaction chính (S1) không chia sẻ khóa, tài nguyên với Autonomous Transaction (S2) nên S2 không có khóa để thực hiện. Cứ như vậy S1 chờ S2 thực hiện xong nhưng S2 lại chờ

			khóa từ S1 dẫn đến deadlock.

- Những thay đổi trong Autonomous Transaction khi đã COMMIT thì sẽ được hiển thị cho các Transaction khác. Do đó người dùng có thể truy cập thông tin đã cập nhật trong Autonomous Transaction mà không cần Transaction Chính (Transaction gọi Autonomous Transaction) được COMMIT.
- Autonomous Transaction có thể gọi các Autonomous Transaction khác và không giới hạn số lần gọi.
- Autonomous Transaction được khai báo bởi "***pragma AUTONOMOUS_TRANSACTION".***
- "*pragma AUTONOMOUS_TRANSACTION"* chỉ thị cho cơ sở dữ liệu rằng Transaction này khi được thực hiện sẽ được thực hiện như một Transaction mới độc lập với Transaction chính.



```
CREATE OR REPLACE PROCEDURE test_none_autonomous
AS
BEGIN
    UPDATE EMPLOYEE
    set EMPLOYEE.SALARY = EMPLOYEE.SALARY + 500000
    WHERE EMPLOYEE.EMPNO = 1;
    ROLLBACK;
END;

CREATE OR REPLACE PROCEDURE test_autonomous
AS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    UPDATE EMPLOYEE
    set EMPLOYEE.SALARY = EMPLOYEE.SALARY + 500000
    WHERE EMPLOYEE.EMPNO = 1;
    ROLLBACK;
END;

BEGIN
    UPDATE EMPLOYEE
    set EMPLOYEE.SALARY = EMPLOYEE.SALARY + 500000
    WHERE EMPLOYEE.EMPNO = 10 AND employee.STATUS = 0;
    TEST_AUTONOMOUS;
    COMMIT;
END;
```

V. Bài tập yêu cầu

1. Sinh viên xây dựng các giao tác trong bài tập thực hành.
   Bài nộp:

   - Tất cả store proc trong mục 4 bài QLTV

   - Tát cả trigger trong mục 5 bài QLTV

2. Sinh viên xây dựng các giao tác trong đồ án môn học.