**Exercise 1: Control Structures (MANDATORY)**

create table customers (customerid number primary key, name varchar2(100), dob date, balance number, lastmodified date);

create table accounts (accountid number primary key, customerid number, accounttype varchar2(20), balance number, lastmodified date, foreign key (customerid) references customers(customerid));

create table transactions (transactionid number primary key, accountid number, transactiondate date, amount number, transactiontype varchar2(10), foreign key (accountid) references accounts(accountid));

create table loans (loanid number primary key, customerid number, loanamount number, interestrate number, startdate date, enddate date, foreign key (customerid) references customers(customerid));

create table employees (employeeid number primary key, name varchar2(100), position varchar2(50), salary number, department varchar2(50), hiredate date);

insert into customers values (1, 'John Doe', to\_date('1985-05-15', 'yyyy-mm-dd'), 1000, sysdate);

insert into customers values (2, 'Jane Smith', to\_date('1990-07-20', 'yyyy-mm-dd'), 1500, sysdate);

insert into accounts values (1, 1, 'Savings', 1000, sysdate);

insert into accounts values (2, 2, 'Checking', 1500, sysdate);

insert into transactions values (1, 1, sysdate, 200, 'Deposit');

insert into transactions values (2, 2, sysdate, 300, 'Withdrawal');

insert into loans values (1, 1, 5000, 8, sysdate, add\_months(sysdate, 60));

insert into loans values (2, 2, 4000, 7, sysdate, add\_months(sysdate, 60));

insert into employees values (1, 'Alice Johnson', 'Manager', 70000, 'HR', to\_date('2015-06-15', 'yyyy-mm-dd'));

insert into employees values (2, 'Bob Brown', 'Developer', 60000, 'IT', to\_date('2017-03-20', 'yyyy-mm-dd'));

commit;

begin

for rec in (select l.loanid, c.dob from customers c join loans l on c.customerid = l.customerid) loop

if floor(months\_between(sysdate, rec.dob) / 12) > 60 then

update loans set interestrate = interestrate - 1 where loanid = rec.loanid;

end if;

end loop;

commit;

end;

/

begin

for rec in (select c.customerid from customers c join accounts a on c.customerid = a.customerid group by c.customerid having sum(a.balance) > 10000) loop

dbms\_output.put\_line('Customer ID ' || rec.customerid || ' is promoted to VIP.');

end loop;

end;

/

begin

for rec in (select l.loanid, c.name from loans l join customers c on l.customerid = c.customerid where l.enddate <= sysdate + 30) loop

dbms\_output.put\_line('Reminder: Loan ID ' || rec.loanid || ' for ' || rec.name || ' is due soon.');

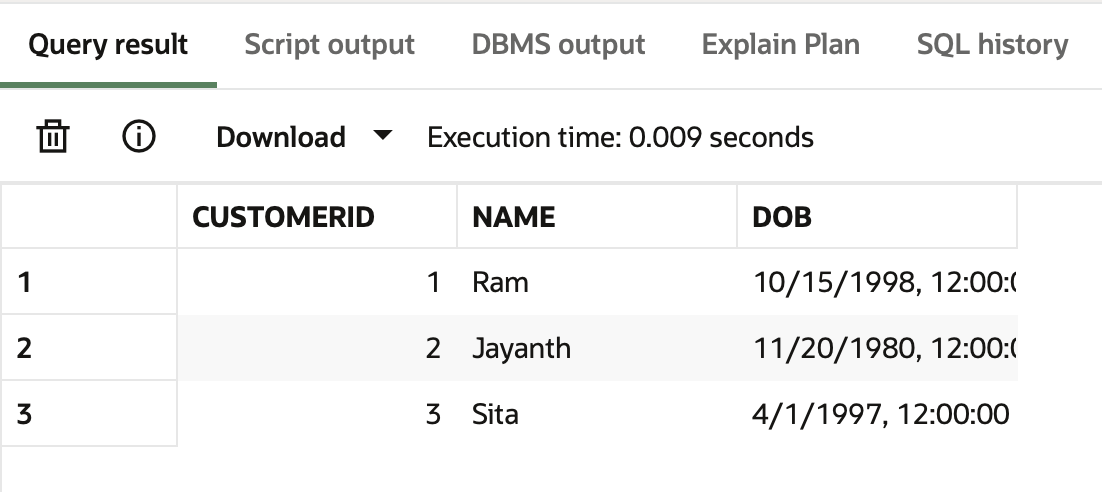
end loop;

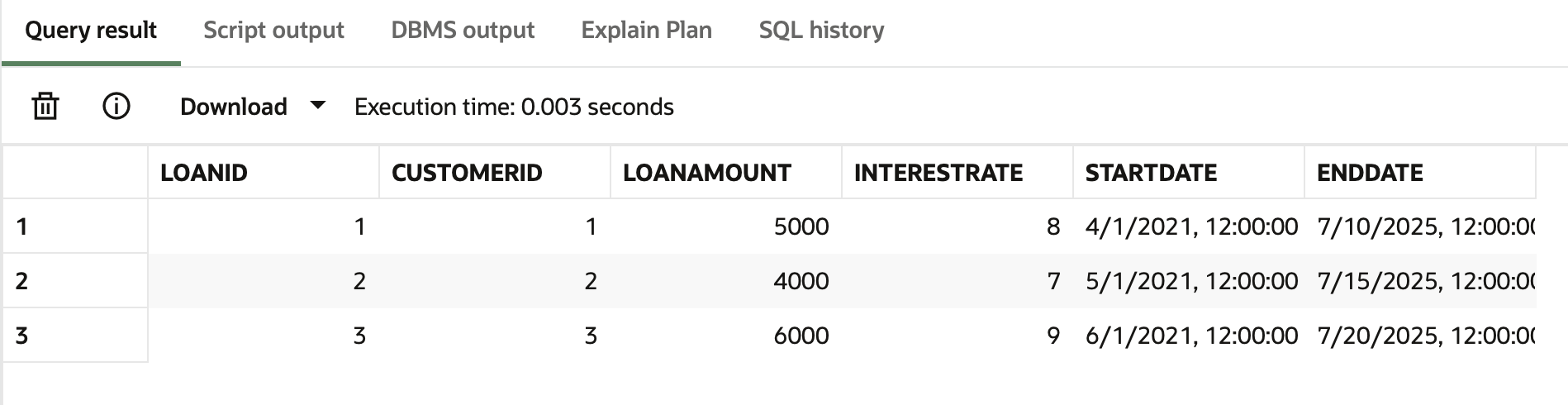
end;

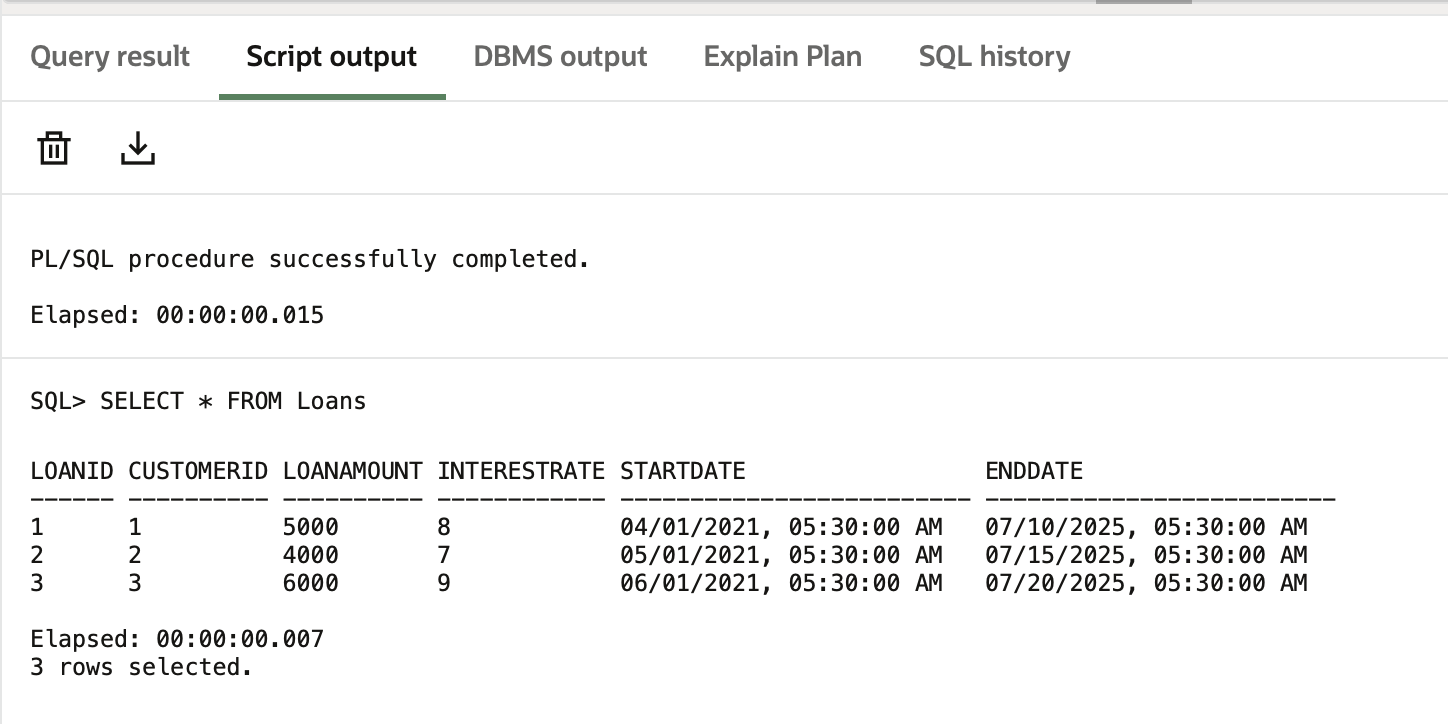
/

select \* from loans;

**Output:**

**Customers Table:**   


**Loans Table**:   


**Output for our procedure:**  


**Exercise 2: Error Handling**

create table customers (customerid number primary key, name varchar2(100), dob date, balance number, lastmodified date);

create table accounts (accountid number primary key, customerid number, accounttype varchar2(20), balance number, lastmodified date, foreign key (customerid) references customers(customerid));

create table employees (employeeid number primary key, name varchar2(100), position varchar2(50), salary number, department varchar2(50), hiredate date);

insert into customers values (1, 'John Doe', to\_date('1985-05-15', 'yyyy-mm-dd'), 1000, sysdate);

insert into customers values (2, 'Jane Smith', to\_date('1990-07-20', 'yyyy-mm-dd'), 1500, sysdate);

insert into accounts values (1, 1, 'Savings', 1000, sysdate);

insert into accounts values (2, 2, 'Checking', 1500, sysdate);

insert into employees values (1, 'Alice Johnson', 'Manager', 70000, 'HR', to\_date('2015-06-15', 'yyyy-mm-dd'));

insert into employees values (2, 'Bob Brown', 'Developer', 60000, 'IT', to\_date('2017-03-20', 'yyyy-mm-dd'));

Commit;

create or replace procedure safetransferfunds(from\_acc in number, to\_acc in number, amount in number) is from\_balance number;

begin select balance into from\_balance from accounts where accountid = from\_acc; if from\_balance < amount then dbms\_output.put\_line('error: insufficient funds in source account.');

Rollback;

else update accounts set balance = balance - amount, lastmodified = sysdate where accountid = from\_acc; update accounts set balance = balance + amount, lastmodified = sysdate where accountid = to\_acc; commit; dbms\_output.put\_line('transfer completed successfully.'); end if; exception when no\_data\_found then dbms\_output.put\_line('error: one of the account ids is invalid.'); rollback; when others then dbms\_output.put\_line('transfer failed: ' || sqlerrm); rollback; end;

/

create or replace procedure updatesalary(emp\_id in number, percent in number) is begin update employees set salary = salary + (salary \* percent / 100) where employeeid = emp\_id; if sql%rowcount = 0 then dbms\_output.put\_line('error: employee not found.'); rollback; else commit; dbms\_output.put\_line('salary updated successfully.'); end if; exception when others then dbms\_output.put\_line('update failed: ' || sqlerrm); rollback; end;

/

create or replace procedure addnewcustomer(cust\_id in number, name in varchar2, dob in date, balance in number) is begin insert into customers (customerid, name, dob, balance, lastmodified) values (cust\_id, name, dob, balance, sysdate); commit; dbms\_output.put\_line('new customer added successfully.'); exception when dup\_val\_on\_index then dbms\_output.put\_line('error: customer id already exists.'); rollback; when others then dbms\_output.put\_line('insert failed: ' || sqlerrm); rollback; end;

/

begin safetransferfunds(1, 2, 300); safetransferfunds(1, 2, 5000); end;

/

begin updatesalary(1, 10); updatesalary(999, 10); end;

/

begin addnewcustomer(3, 'Meera Rao', to\_date('1995-03-10', 'yyyy-mm-dd'), 2000); addnewcustomer(1, 'Duplicate John', to\_date('1990-01-01', 'yyyy-mm-dd'), 500); end;

/

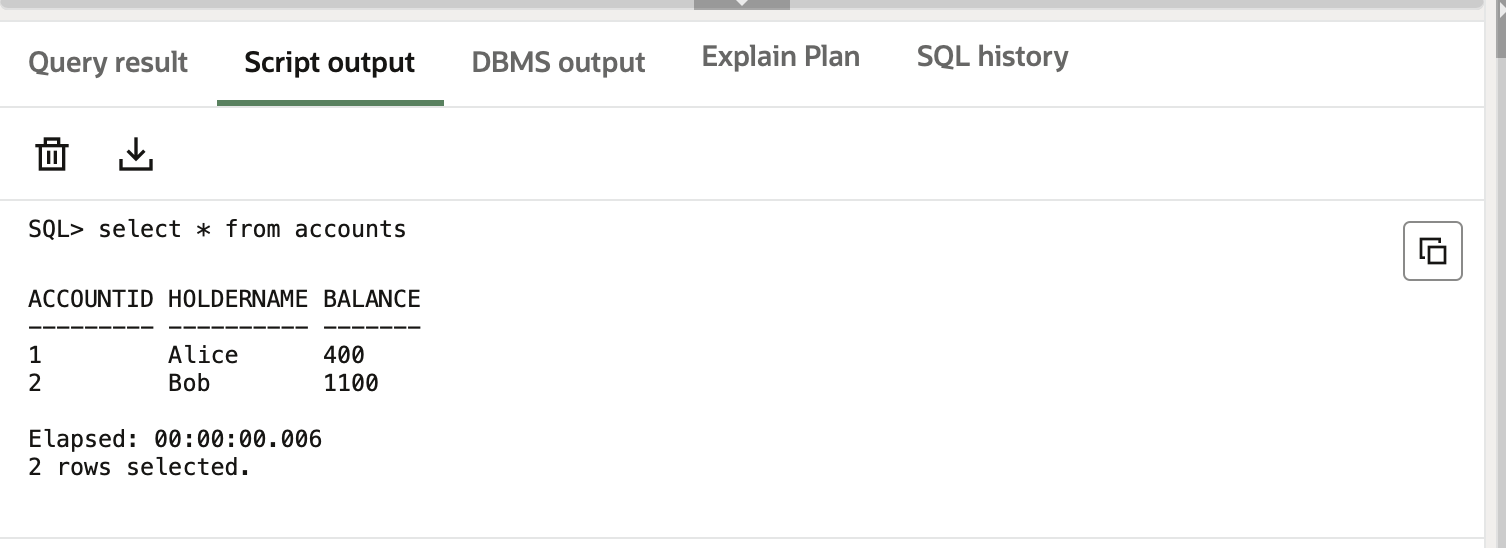
select \* from accounts;

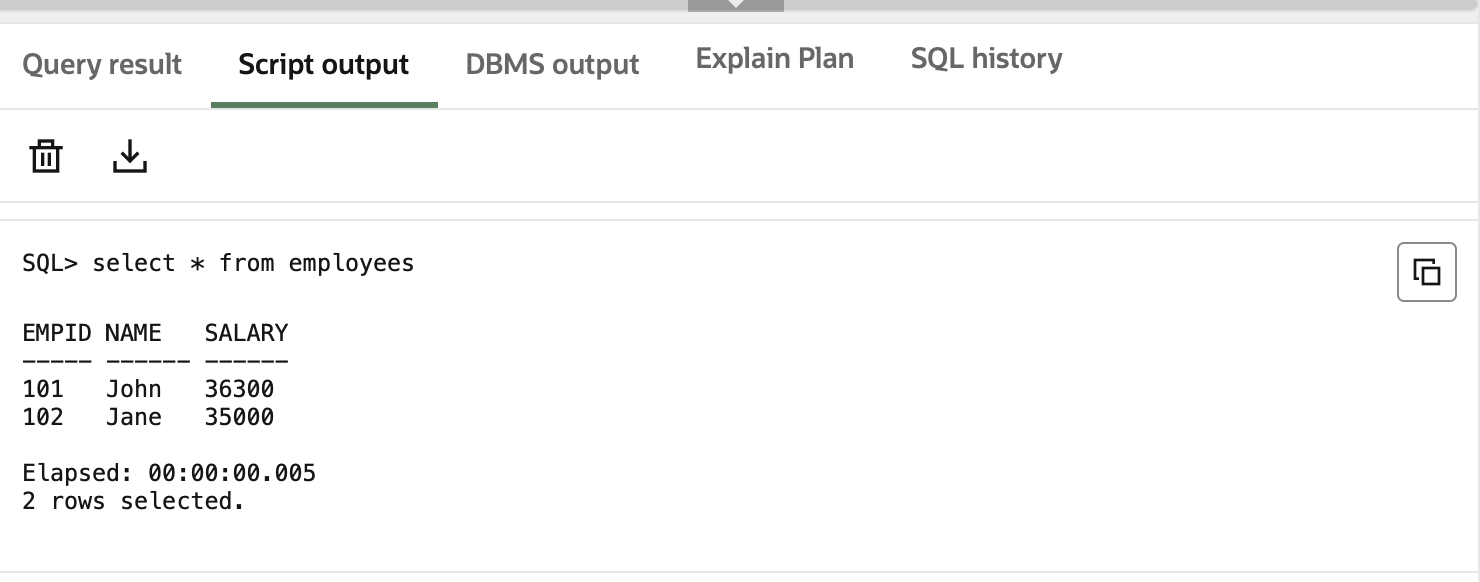
select \* from employees;

select \* from customers;

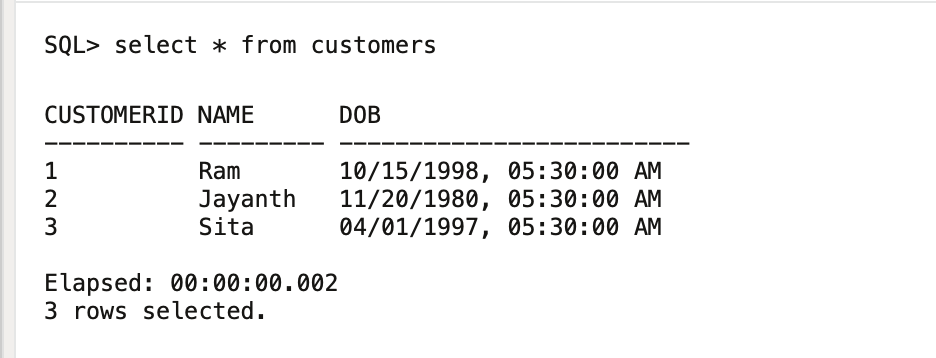
**Output:**

**Accounts table**

****

**Employees table:  
**

**Customers table:**

****

**Exercise 3: Stored Procedures (MANDATORY)**

create table customers (customerid number primary key, name varchar2(100), dob date, balance number, lastmodified date);

create table accounts (accountid number primary key, customerid number, accounttype varchar2(20), balance number, lastmodified date, foreign key (customerid) references customers(customerid));

create table employees (employeeid number primary key, name varchar2(100), position varchar2(50), salary number, department varchar2(50), hiredate date);

insert into customers values (1, 'John Doe', to\_date('1985-05-15', 'yyyy-mm-dd'), 1000, sysdate);

insert into customers values (2, 'Jane Smith', to\_date('1990-07-20', 'yyyy-mm-dd'), 1500, sysdate);

insert into accounts values (1, 1, 'Savings', 1000, sysdate);

insert into accounts values (2, 2, 'Savings', 1500, sysdate);

insert into employees values (1, 'Alice Johnson', 'Manager', 70000, 'HR', to\_date('2015-06-15', 'yyyy-mm-dd'));

insert into employees values (2, 'Bob Brown', 'Developer', 60000, 'IT', to\_date('2017-03-20', 'yyyy-mm-dd'));

commit;

create or replace procedure processmonthlyinterest is

begin

update accounts set balance = balance + (balance \* 0.01), lastmodified = sysdate where accounttype = 'Savings';

commit;

dbms\_output.put\_line('Monthly interest processed for all savings accounts.');

exception when others then dbms\_output.put\_line('Error: ' || sqlerrm); rollback;

end;

/

create or replace procedure updateemployeebonus(dept in varchar2, bonus\_percent in number) is

begin

update employees set salary = salary + (salary \* bonus\_percent / 100) where department = dept;

if sql%rowcount = 0 then dbms\_output.put\_line('No employees found in department ' || dept); rollback;

else commit; dbms\_output.put\_line('Bonus updated for department ' || dept); end if;

exception when others then dbms\_output.put\_line('Error: ' || sqlerrm); rollback;

end;

/

create or replace procedure transferfunds(from\_acc in number, to\_acc in number, amt in number) is from\_balance number;

begin

select balance into from\_balance from accounts where accountid = from\_acc;

if from\_balance < amt then dbms\_output.put\_line('Error: Insufficient balance.'); rollback;

else update accounts set balance = balance - amt, lastmodified = sysdate where accountid = from\_acc;

update accounts set balance = balance + amt, lastmodified = sysdate where accountid = to\_acc;

commit; dbms\_output.put\_line('Transfer successful.'); end if;

exception when no\_data\_found then dbms\_output.put\_line('Error: Invalid account ID.'); rollback;

when others then dbms\_output.put\_line('Error: ' || sqlerrm); rollback;

end;

/

begin processmonthlyinterest; end;

/

begin updateemployeebonus('IT', 5); updateemployeebonus('SALES', 10); end;

/

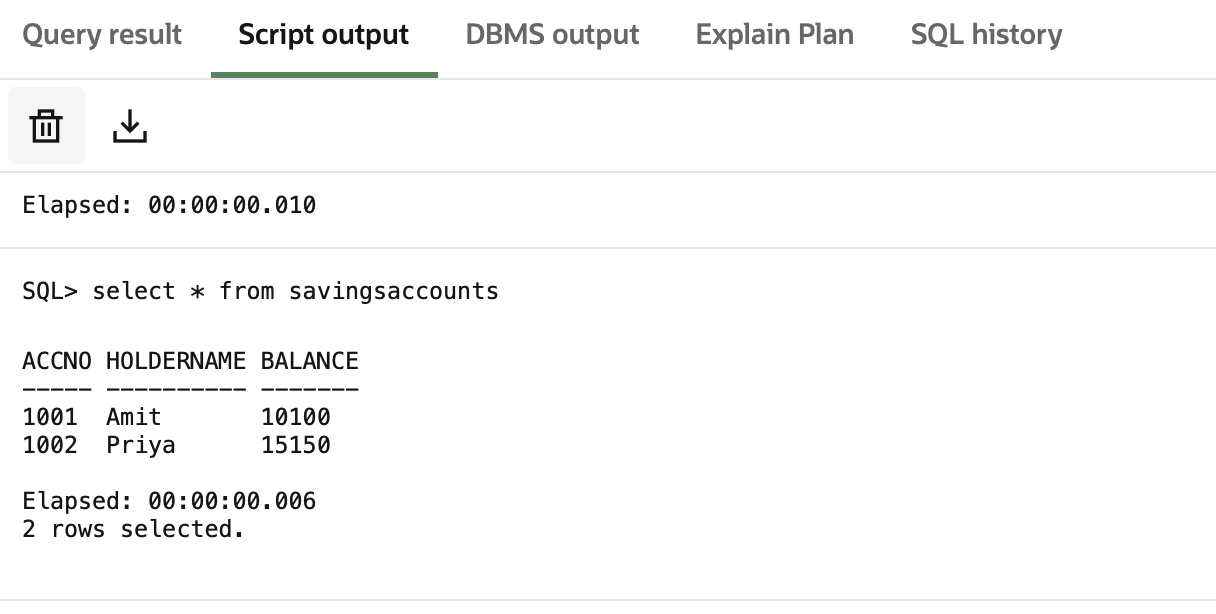
begin transferfunds(1, 2, 300); transferfunds(1, 2, 10000); end;

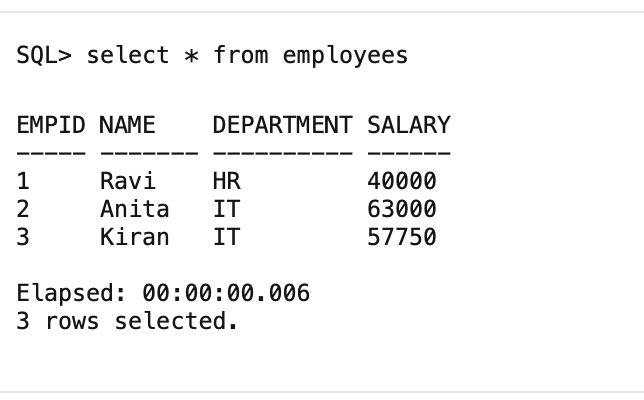
/

select \* from accounts;

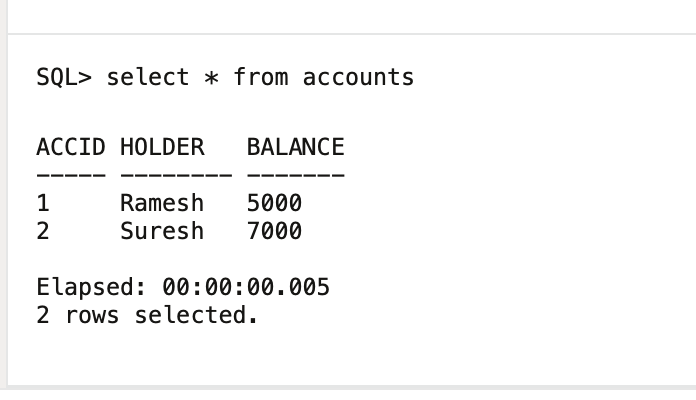
select \* from employees;

**Output:**

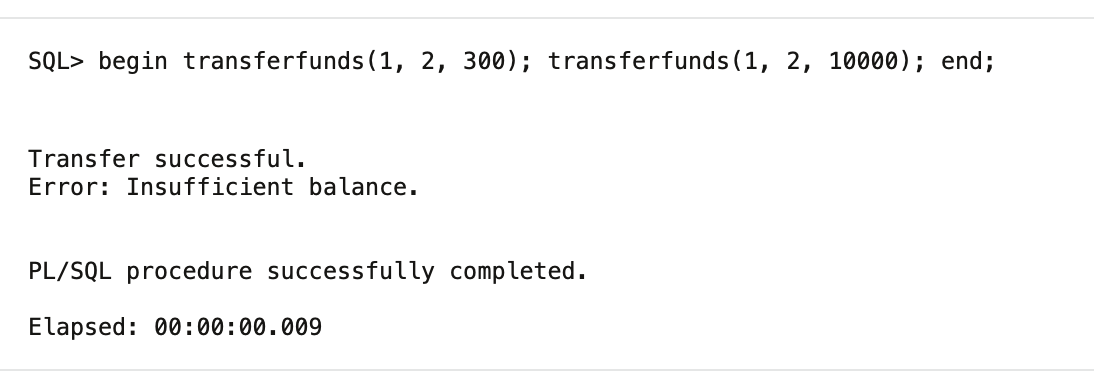
****

****

**Final output:**

****

**Implemented procedure**

****

**Exercise 4: Functions**

create or replace function calculateage(dob in date) return number is

age number;

begin

age := floor(months\_between(sysdate, dob) / 12);

return age;

end;

/

create or replace function calculatemonthlyinstallment(

loanamount in number,

interestrate in number,

duration\_years in number

) return number is

monthly\_rate number;

total\_months number;

installment number;

begin

monthly\_rate := interestrate / 1200;

total\_months := duration\_years \* 12;

installment := loanamount \* monthly\_rate / (1 - power(1 + monthly\_rate, -total\_months));

return round(installment, 2);

end;

/

create or replace function hassufficientbalance(accid in number, amt in number) return boolean is

acc\_balance number;

begin

select balance into acc\_balance from accounts where accountid = accid;

return acc\_balance >= amt;

exception

when no\_data\_found then

return false;

when others then

return false;

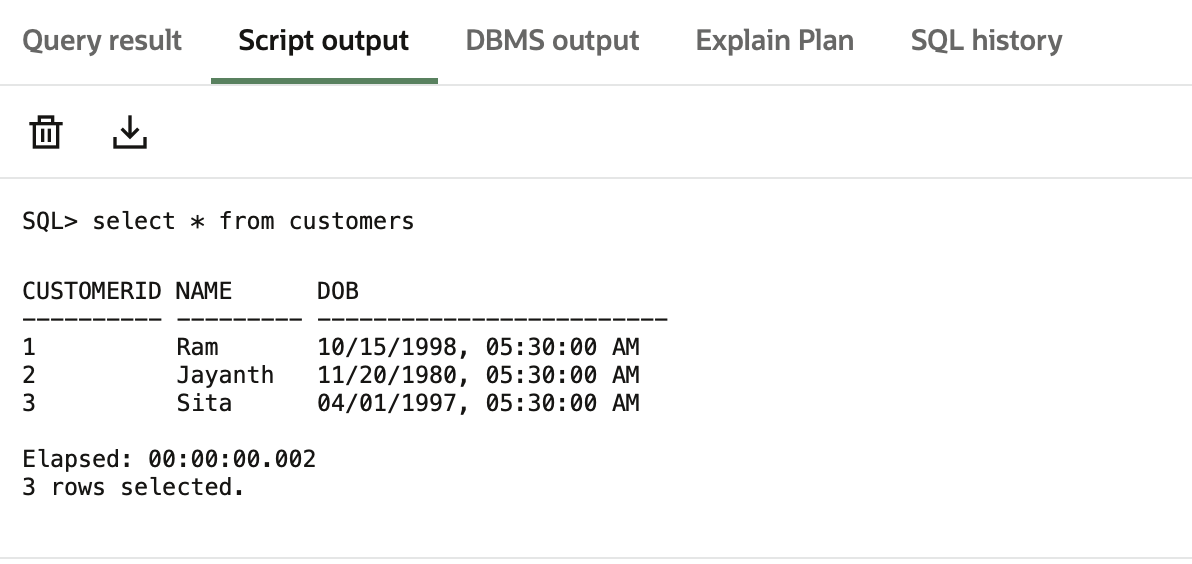
end;

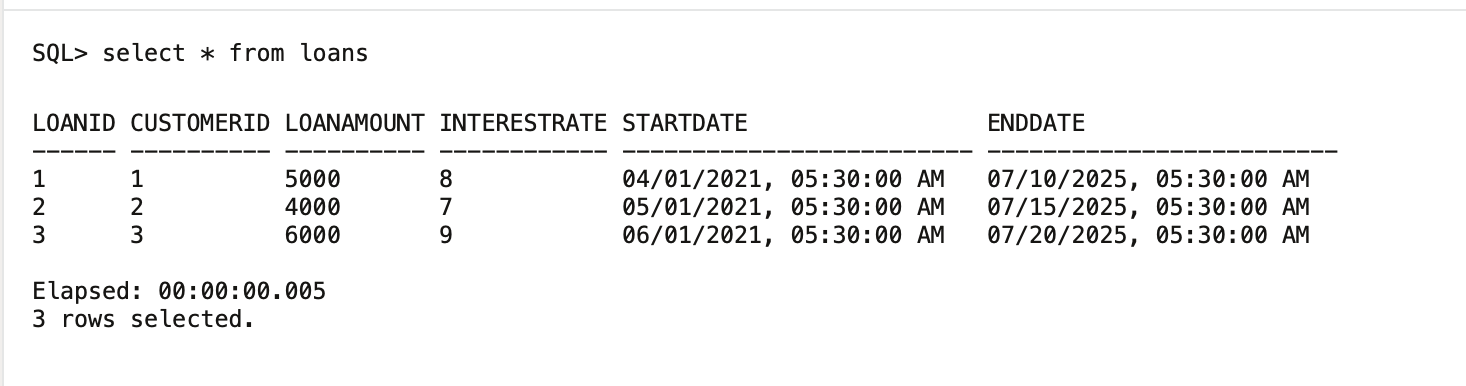
/

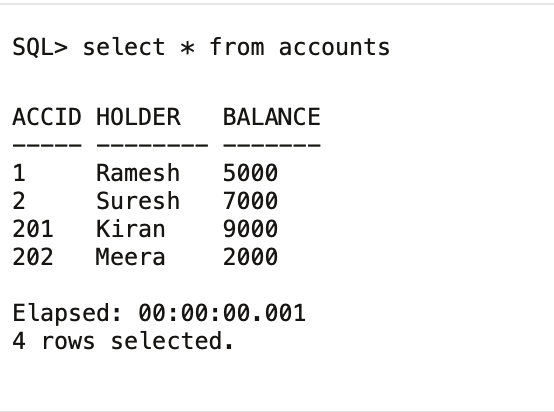
select calculateage(to\_date('1990-07-20', 'yyyy-mm-dd')) as age from dual;

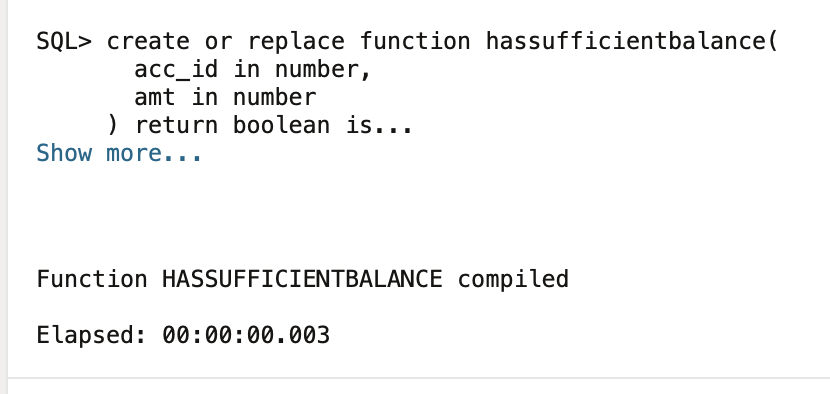
select calculatemonthlyinstallment(5000, 5, 5) as monthly\_installment from dual;

select hassufficientbalance(1, 500) as has\_funds from dual;

**Output:   
**

****

****

**Function execution:   
**

**Exercise 5: Triggers**

create table auditlog (logid number generated always as identity primary key, transactionid number, message varchar2(100), logdate date);

create or replace trigger updatecustomerlastmodified

before update on customers

for each row

begin

:new.lastmodified := sysdate;

end;

/

create or replace trigger logtransaction

after insert on transactions

for each row

begin

insert into auditlog (transactionid, message, logdate)

values (:new.transactionid, 'Transaction recorded for account ' || :new.accountid, sysdate);

end;

/

create or replace trigger checktransactionrules

before insert on transactions

for each row

declare

acc\_bal number;

begin

select balance into acc\_bal from accounts where accountid = :new.accountid;

if :new.transactiontype = 'Withdrawal' and :new.amount > acc\_bal then

raise\_application\_error(-20001, 'Withdrawal exceeds available balance.');

end if;

if :new.transactiontype = 'Deposit' and :new.amount <= 0 then

raise\_application\_error(-20002, 'Deposit amount must be positive.');

end if;

exception

when no\_data\_found then

raise\_application\_error(-20003, 'Account does not exist.');

end;

/

update customers set balance = balance + 100 where customerid = 1;

insert into transactions (transactionid, accountid, transactiondate, amount, transactiontype)

values (3, 1, sysdate, 500, 'Deposit');

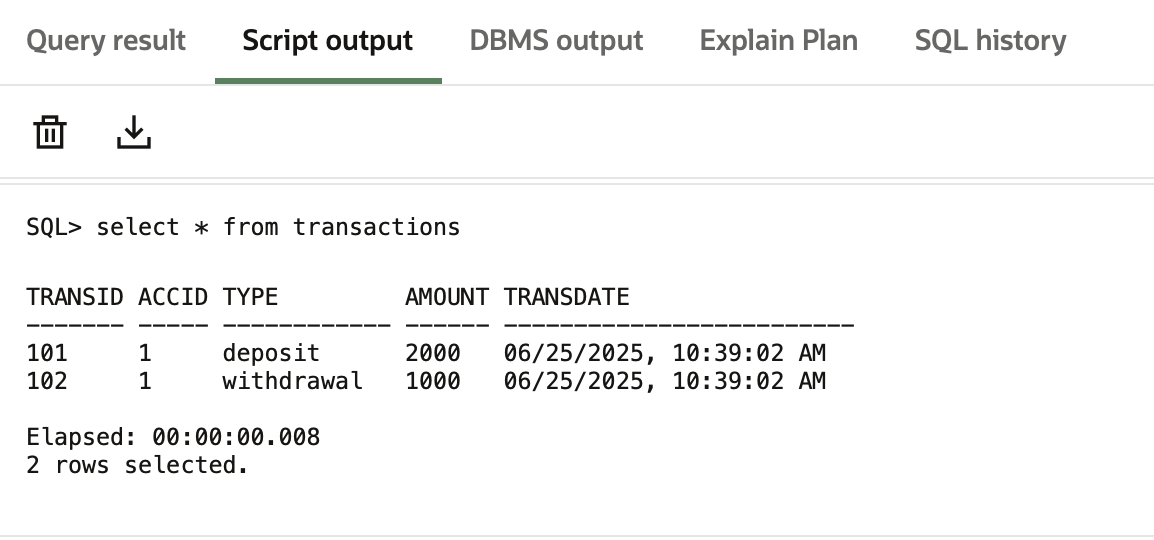
insert into transactions (transactionid, accountid, transactiondate, amount, transactiontype)

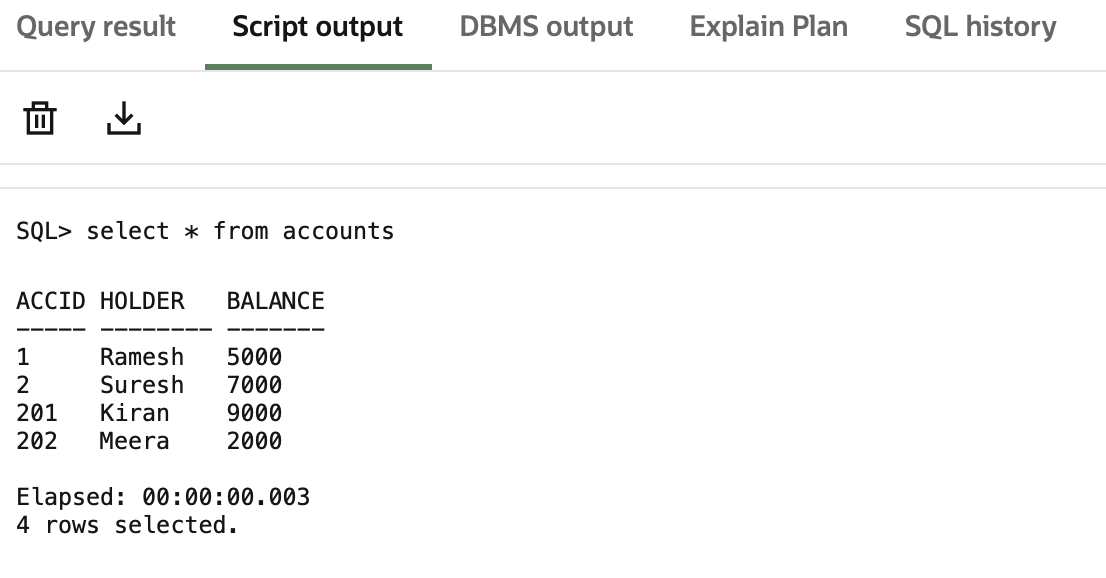
values (4, 2, sysdate, 2000, 'Withdrawal');

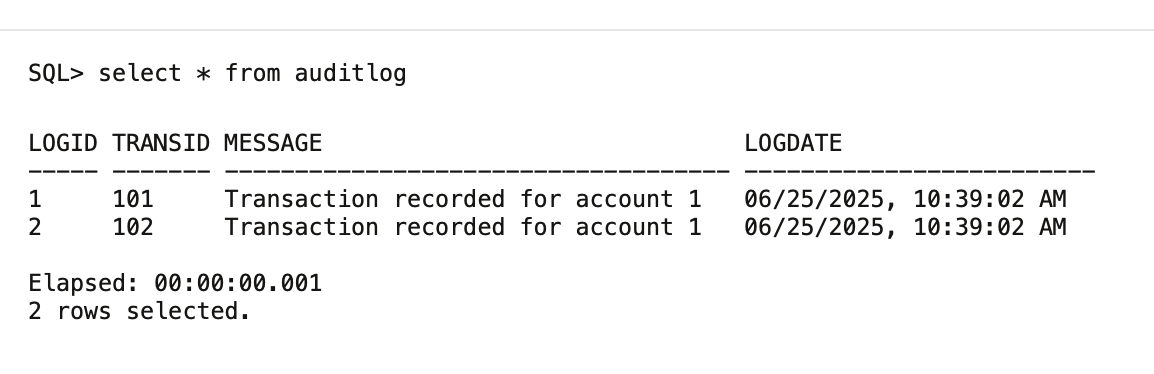
select \* from customers;

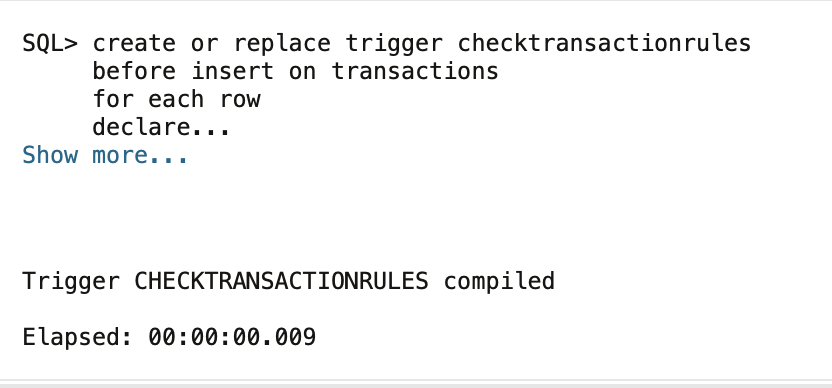
select \* from transactions;

select \* from auditlog;

**Output:   
**

****

****

**Trigger Execution:   
**

**Exercise 6: Cursors**

begin

for r in ( select c.name, t.transactiondate, t.amount, t.transactiontype

from customers c join accounts a on c.customerid = a.customerid

join transactions t on a.accountid = t.accountid where extract(month from t.transactiondate) = extract(month from sysdate)

and extract(year from t.transactiondate) = extract(year from sysdate)

) loop

dbms\_output.put\_line('Customer: ' || r.name || ', Date: ' || r.transactiondate || ', Amount: ' || r.amount || ', Type: ' || r.transactiontype);

end loop;

end;

/

declare

cursor fee\_cursor is select accountid, balance from accounts;

fee number := 100;

begin

for rec in fee\_cursor loop

update accounts set balance = balance - fee, lastmodified = sysdate where accountid = rec.accountid;

end loop;

commit;

dbms\_output.put\_line('Annual fee applied to all accounts.');

end;

/

declare

cursor loan\_cursor is select loanid, loanamount, interestrate from loans;

begin

for l in loan\_cursor loop

if l.loanamount > 6000 then

update loans set interestrate = interestrate + 0.5 where loanid = l.loanid;

end if;

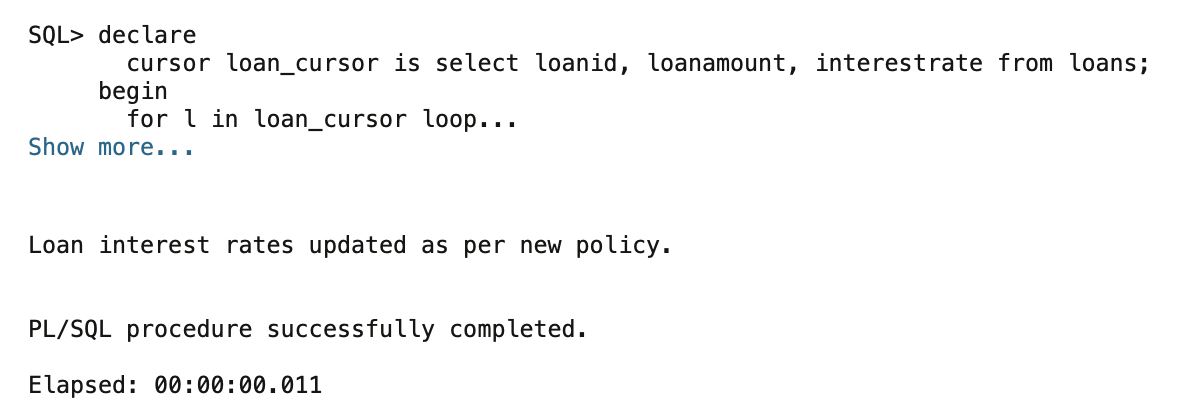
end loop;

commit;

dbms\_output.put\_line('Loan interest rates updated as per new policy.');

end;

/

**Output: Cursor implemented   
**

**Exercise 7: Packages**

create or replace package body customermanagement as

procedure addcustomer(cid number, name varchar2, dob date, balance number) is

begin

insert into customers values (cid, name, dob, balance, sysdate);

end;

procedure updatecustomer(cid number, name varchar2) is

begin

update customers set name = name, lastmodified = sysdate where customerid = cid;

end;

function getbalance(cid number) return number is

bal number;

begin

select balance into bal from customers where customerid = cid;

return bal;

end;

end;

/

create or replace package employeemanagement as

procedure hire(empid number, name varchar2, pos varchar2, sal number, dept varchar2, hdate date);

procedure updateemp(empid number, sal number);

function annualsalary(empid number) return number;

end;

/

create or replace package body employeemanagement as

procedure hire(empid number, name varchar2, pos varchar2, sal number, dept varchar2, hdate date) is

begin

insert into employees values (empid, name, pos, sal, dept, hdate);

end;

procedure updateemp(empid number, sal number) is

begin

update employees set salary = sal where employeeid = empid;

end;

function annualsalary(empid number) return number is

total number;

begin

select salary \* 12 into total from employees where employeeid = empid;

return total;

end;

end;

/

create or replace package accountoperations as

procedure openacc(aid number, cid number, atype varchar2, bal number);

procedure closeacc(aid number);

function totalbalance(cid number) return number;

end;

/

create or replace package body accountoperations as

procedure openacc(aid number, cid number, atype varchar2, bal number) is

begin

insert into accounts values (aid, cid, atype, bal, sysdate);

end;

procedure closeacc(aid number) is

begin

delete from accounts where accountid = aid;

end;

function totalbalance(cid number) return number is

total number;

begin

select sum(balance) into total from accounts where customerid = cid;

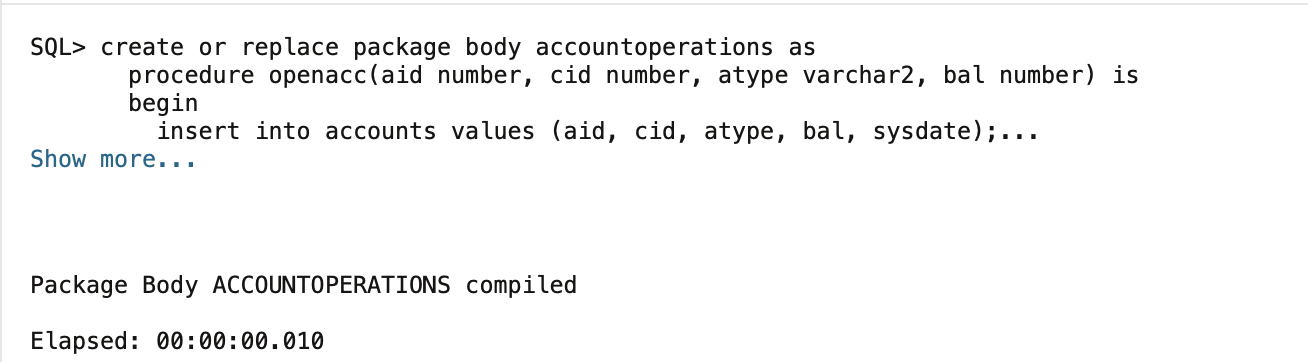
return total;

end;

end;

/

**Output: Package Implemented**

****