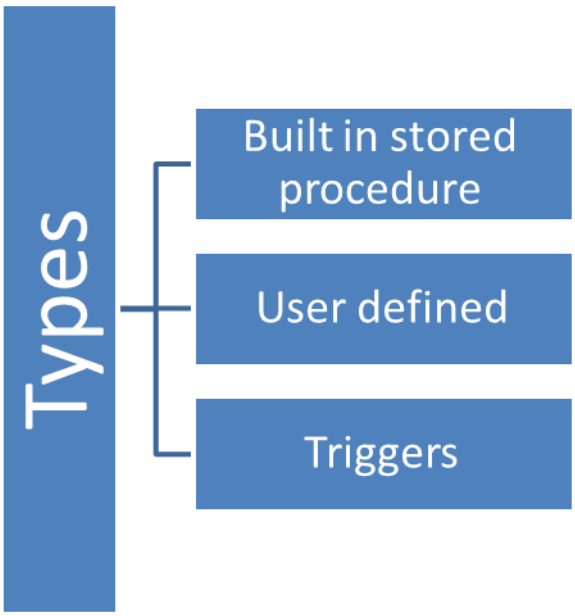
Stored procedures



Stored procedures VS query statement

- **Query statement =inline sql =Adhoc queries**

Steps to execute query

-parsing query

-optimization query

-query tree

-query plan = execution plan

Advantages of stored procedures

- **Execution plan retention and reusability.**
- **Reduces network traffic.**
- **Code re-usability and better maintainability.**
- **Better Security.**
- **Avoids SQL Injection attack**

Parameter mode

1- in parameter

2- out parameter

3- in-out parameter

Parameter passing

1- **Passing by name**

2- **Passing by position**

Examples of stored procedures

Create stored procedure to execute update statement

```
Create or REPLACE PROCEDURE update_employee_data (code in
EMPLOYEES.EMPLOYEE_ID%TYPE , commission  in number)

IS

BEGIN

update EMPLOYEES

set salary = salary*(1+ commission  /100)

where EMPLOYEE_ID = code ;

end update_employee_data ;

#calling stored procedure

execute update_employee_data (100,10) ;

#test the salary already change

select salary  from EMPLOYEES

where EMPLOYEE_ID=100  ;
```

---

stored procedure to select data

```
create or replace procedure retrieve_emp_data (id in EMPLOYEES.employee_id%type ,name
out EMPLOYEES.last_name%type,

salary out EMPLOYEES.salary%type )

Is

Begin

select last_name , salary into  name, salary  from employees

where employee_id=id;

end retrieve_emp_data;

#execute stored procedure retrieve_emp_data

SET SERVEROUTPUT ON
```

```
DECLARE
emp_name employees.last_name%type ; #variable
emp_salary employees.salary%type ; #variable
BEGIN
 retrieve_emp_data  (179,emp_name, emp_salary);  #calling the procedure
DBMS_OUTPUT.PUT_LINE(emp_name || emp_salary);
end ;
```

stored procedure insert data

```
create or replace procedure add_dept (id in departments.department_id%type,
name in DEPARTMENTS.DEPARTMENT_NAME%type ,loc in departments.location_id%type)
Is
Begin
Insert into departments (department_id,DEPARTMENT_NAME,location_id)
values (id,name,loc);
End add_dept;
#calling procedure
execute add_dept(   1005,'is', 1700)
#test the new record adding
select * from departments ;
```

---

stored procedures to delete record

```
create or replace procedure drop_dept(id in departments.department_id%type
)
Is
Begin
delete from departments
where department_id=id ;
```

end drop_dept;

#calling procedure

execute drop_dept (1005);

#test procedure delete record

SELECT * FROM departments

Trigger characteristics

- No parameters
- Calling automatically (after or instead of action or before)
- Created two tables (deleted and inserted)

Trigger timing

Trigger execute automatically in specified action

-after any sql statement

-before any sql statement

-instead of any sql statement

Trigger event

-sql statements (DML)

Trigger based on database objects

-Trigger apply only on tables and view

Write trigger to allow users to perform any SQL statements into employees table only in formal business hours

Hint: holiday (Saturday, Sunday), business hours from (8:00 am to 6 pm)

```
create or replace TRIGGER secure_emp
before insert or delete OR update  on employees
begin
if (TO_CHAR(SYSDATE ,'DY') in ('SAT','SUN')) or
(TO_CHAR(SYSDATE ,'HH24:MI')
not between '08:00' and '18:00'  ) then
RAISE_APPLICATION_ERROR(-20500, 'you may insert '||'insert employee table
during '||'business hours ');
end if ;
end ;
```

Write trigger to restrict salary for only employees who job_id is (ad_pres,ad_vp) and earn salary 15.000$

```
create or replace TRIGGER restricate_salary
before insert or update of salary on employees
for each row
begin
```

```
if not (:NEW.job_id in('ad_pres','ad_vp'))
and :new.salary >15000 then
RAISE_APPLICATION_ERROR(-20202, ' employee can not earn more than 150000 ');

end if ;
end ;
```