SQL Programming or PL/SQL

Overview

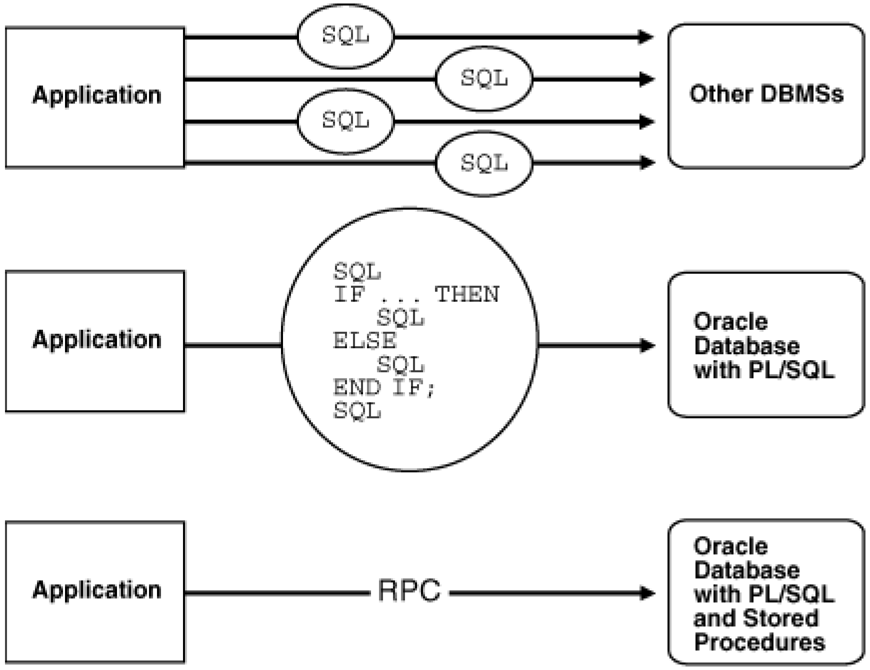
- Introduction – Programming SQL
- Cursors
- Procedures
- Triggers

Programming SQL

- PL/SQL is the procedural extension to SQL.
- PL/SQL is a block structured language.
- Data manipulation and query statements of SQL are included within procedural units of code.

Advantages

- Program with procedural language control structures.
- PL/SQL can handle errors
- High Performance
- High Productivity
- Full Portability
- Access to Predefined Packages
- Improved data security and integrity
- Support for Developing Web Applications and Server Pages



PL/SQL Block

- A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END.

```
  DECLARE                          -- Declarative part (optional)
  -- Declarations of local types, variables, & subprograms
BEGIN                              -- Executable part (required)
  -- Statements (which can use items declared in declarative part)
[EXCEPTION                         -- Exception-handling part (optional)
  -- Exception handlers for exceptions raised in executable part]
END;
```

PL/SQL Block

- A PL/SQL unit is any one of the following:
  - PL/SQL block
  - FUNCTION
  - PACKAGE
  - PROCEDURE
  - TRIGGER

PL/SQL Block

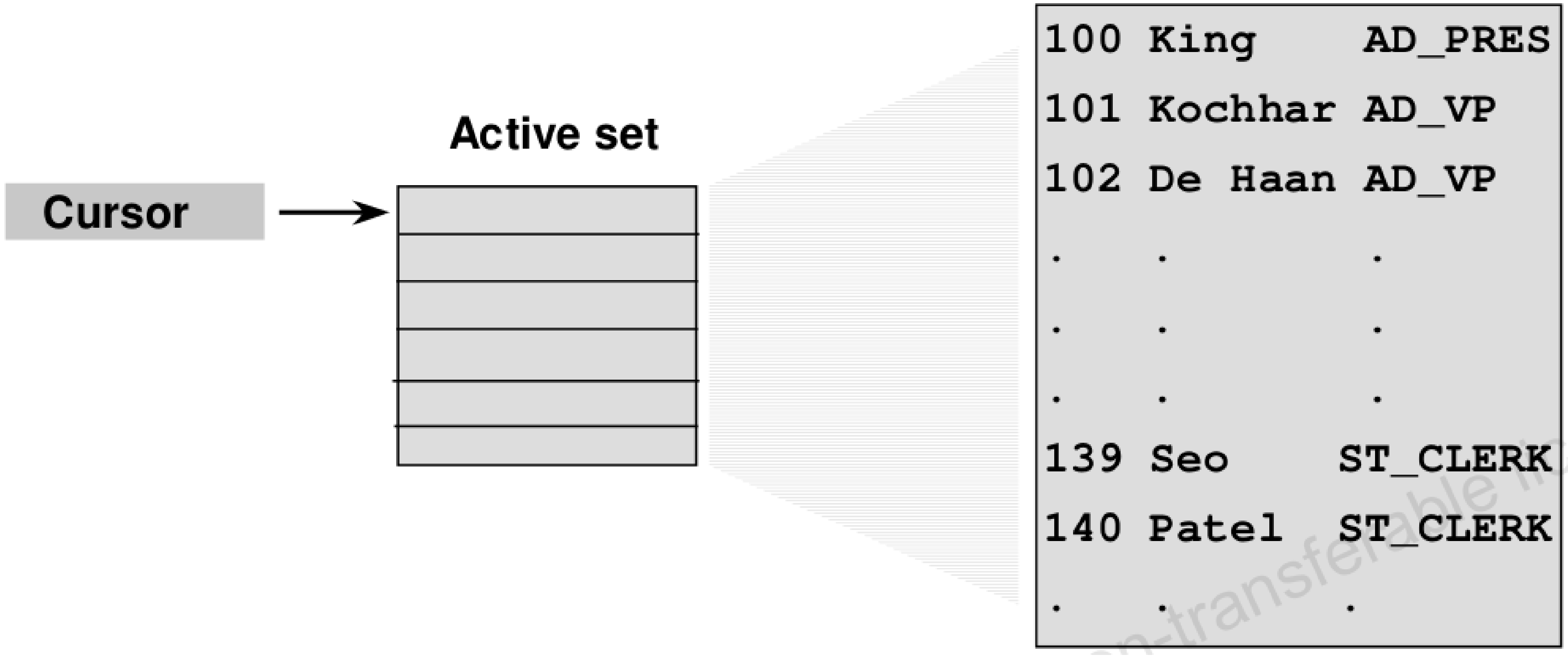
```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE('WELCOME TO SSN!');
  3  DBMS_OUTPUT.PUT_LINE('LEARNING PL/SQL IS FUN!');
  4  END;
  5  /
WELCOME TO SSN!
LEARNING PL/SQL IS FUN!

PL/SQL procedure successfully completed.
```

Cursors

- Every SQL statement executed by the Oracle Server has an individual cursor associated with it:
- Implicit cursors: Declared for all DML and PL/SQL SELECT statements
- Explicit cursors: Declared and named by the programmer

Cursors



```
100 King     AD_PRES
101 Kochhar  AD_VP
102 De Haan  AD_VP
.    .       .
.    .       .
.    .       .
139 Seo     ST_CLERK
140 Patel   ST_CLERK
.    .      .
```

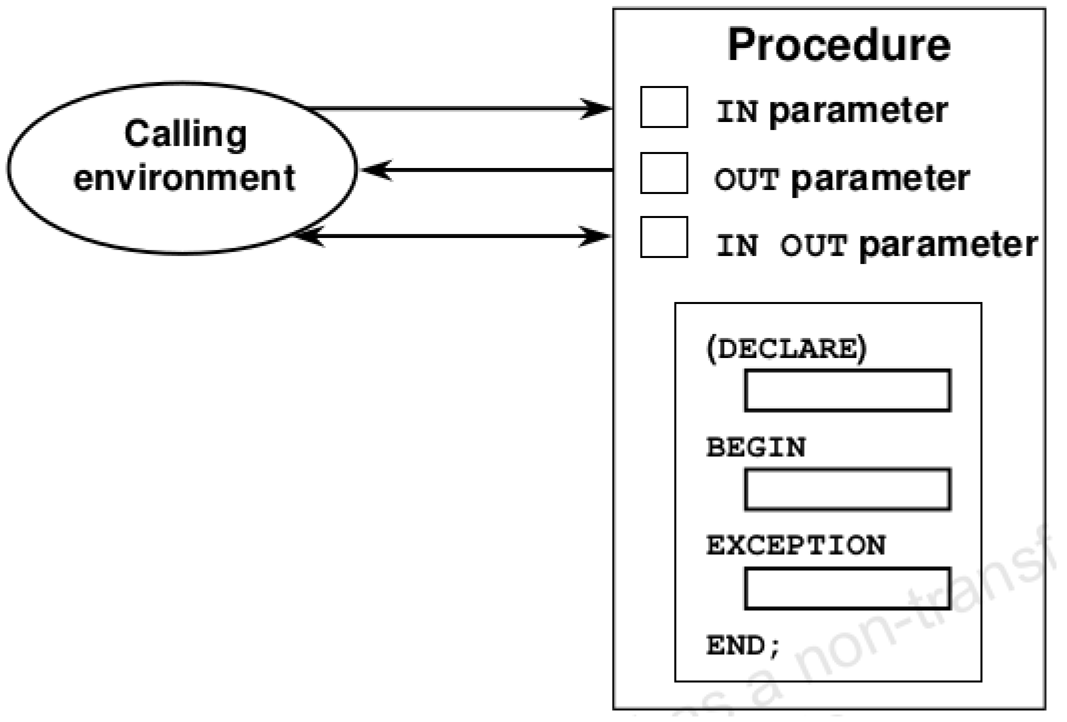
Cursors

```
Syntax:     CURSOR cursor_name IS
            select_statement;
Example:
DECLARE
    CURSOR emp_cursor IS
        SELECT employee_id, last_name
        FROM employees;
    CURSOR dept_cursor IS
        SELECT *
        FROM departments
        WHERE location_id = 170;
BEGIN
...
```

Anonymous PL/SQL using Cursors

```
DECLARE
    CURSOR emp_cursor IS
        SELECT last_name, department_id
        FROM employees;
BEGIN
    FOR emp_record IN emp_cursor LOOP
                        --implicit open and implicit fetch occur
        IF emp_record.department_id = 80 THEN
            DBMS_OUTPUT.PUT_LINE ('Employee' ||
             emp_record.last_name ||' works in the Sales');
        END IF;
    END LOOP; --implicit close and implicit loop exit
END ;
```

PL/SQL Procedure Parameters



Stored Procedure – Example 1

- Accept the parameter for emp_id and increment salary by 10%

```
CREATE OR REPLACE PROCEDURE raise_salary
       (p_id IN employees.employee_id%TYPE)
IS
BEGIN
   UPDATE employees
   SET salary = salary * 1.10
   WHERE employee_id = p_id;
END raise_salary;

To call subprogram:
EXECUTE raise_salary (176);
```

Stored Procedure – Example 1

- Stored procedures can be called from an anonymous PL/SQL blocks.

```
To call subprogram using anonymous PL/SQL block:

DECLARE
v_id NUMBER := 163;
BEGIN
    raise_salary(v_id);
END;
```

Stored Procedure – Example 2

- Invoking a stored procedure from another stored procedure that uses a cursor.

```
CREATE OR REPLACE PROCEDURE process_emps
IS
    CURSOR emp_cursor IS
        SELECT employee_id
        FROM employees;
BEGIN
    FOR emp_rec IN emp_cursor
    LOOP
        raise_salary(emp_rec.employee_id);
    END LOOP;
END process_emps;
```

Stored Procedure – Example 3

- Accept the parameter for emp_id and display emp details

```
CREATE OR REPLACE PROCEDURE query_emp
    (p_id IN employees.employee_id%TYPE,
     p_name OUT employees.last_name%TYPE,
     p_salary OUT employees.salary%TYPE,
     p_comm OUT employees.commission_pct%TYPE)
IS
BEGIN
    SELECT last_name, salary, commission_pct
    INTO p_name, p_salary, p_comm
    FROM employees
    WHERE employee_id = p_id;
END query_emp;
```

Stored Procedure – Example 3

- Accept the parameter for emp_id and display emp details

```
To call the procedure:


VARIABLE g_name VARCHAR2(25)

VARIABLE g_sal NUMBER

VARIABLE g_comm NUMBER


EXECUTE query_emp(171, :g_name, :g_sal, :g_comm)


PRINT g_name g_sal g_comm
```

Stored Procedure – Example 4

- Write a procedure to display the accused name for given chargesheet number.

```
SQL>
  1  create or replace procedure ch_accused(chno IN
                                      varchar2,acc OUT varchar2)
  2  is
  3  aname varchar2(20);
  4  begin
  5    select accusname into acc
       from chargesheet where chrgshno=chno;
  6* end;
SQL> /

Procedure created.
```

Stored Procedure – Example 4

- Use anonymous block to call the procedure.

```
  1   declare
  2     chno varchar2(20);
  3     aname varchar2(20);
  4   begin
  5     chno:='&chno';
  6     ch_accused(chno,aname);
  7     dbms_output.put_line(aname);
  8*  end;
SQL> /

Enter value for chno: CHR101

old   5: chno:='&chno';

new   5: chno:='CHR101';

Kutti Raja

PL/SQL procedure successfully completed.
```

Stored Procedure – Example 5

- Display the FIR details filed by the given police name.

```
SQL>
  1  create or replace procedure gen_report(pname IN
                                                  varchar2)
  2  is
  3  cursor fir_cur is
  4       select c.compno,cname,firno,distr
  5       from complaint c,fir f
  6       where c.compno=f.compno and policename=pname;
  7  begin
  8  dbms_output.put_line('FIR REPORT FOR POLICE: '||pname);
  9  for cur in fir_cur loop
 10     dbms_output.put_line(cur.compno||' '||cur.cname||
                              ' '|| cur.firno||' '||cur.distr);
 11  end loop;
 12* end;
SQL> /

Procedure created.
```

Stored Procedure – Example 5

- Display the FIR details filed by the given police name.

```
SQL>  declare
  2   pname varchar2(20);
  3   begin
  4   pname:='&police';
  5   gen_report(pname);
  6   end;
  7  /
Enter value for police: Vijayakanth
old   4:  pname:='&police';
new   4:  pname:='Vijayakanth';

FIR REPORT FOR POLICE: Vijayakanth
CO102 P.RANI FIR102 CHENNAI
CO103 C.RAMESH FIR103 CHENNAI

PL/SQL procedure successfully completed.
```

Stored Procedure – Exercise

- Generate the complaint report for the given month. The complaint report should print the complaint details and if FIR, chargesheet is filed, display its details

Triggers

- A trigger:
  - Is a PL/SQL block associated with a table, view, schema, or the database
  - Executes *implicitly* whenever a particular event takes place
- Can be either:
  - Application trigger: Fires whenever an event occurs with a particular application
  - Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or a database

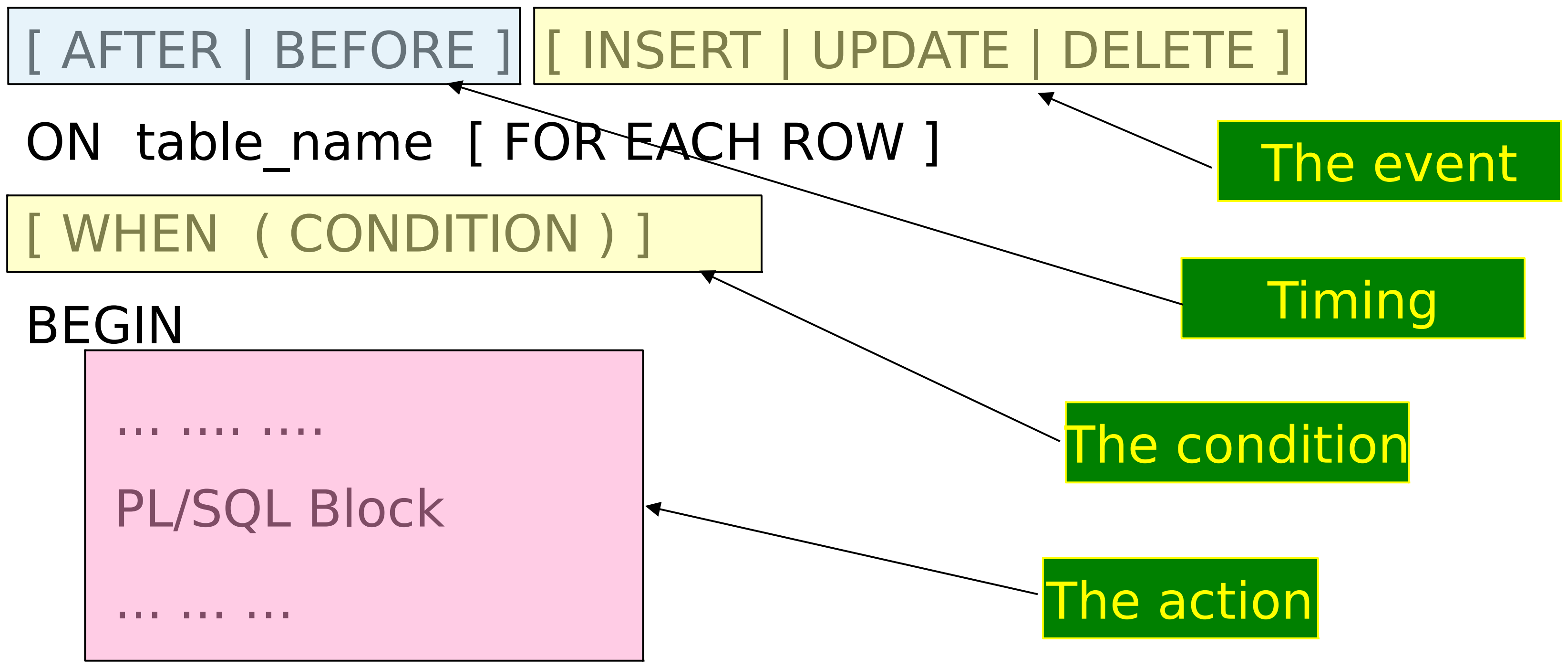
Triggers Applications

- Provide sophisticated auditing
- Prevent invalid transactions
- Enforce complex business rules
- Enforce complex security authorizations
- Provide transparent event logging
- Automatically generate derived column values
- Track system events

Triggers

- A triggering statement contains:
- Trigger timing
  - BEFORE, AFTER
- Triggering event: INSERT, UPDATE, or DELETE
- Table name: On table, view
- Trigger type: Row or statement
- WHEN clause: Restricting condition
- Trigger body: PL/SQL block

Triggers



- CREATE [OR REPLACE] TRIGGER trigg_name

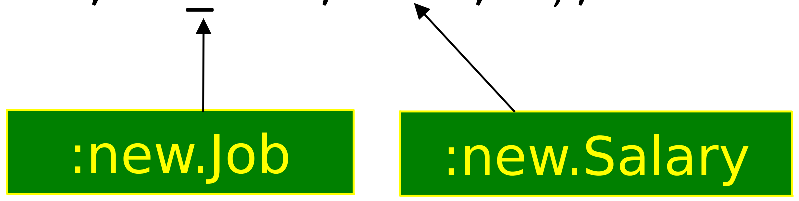
```
[ AFTER | BEFORE ] [ INSERT | UPDATE | DELETE ]
ON  table_name  [ FOR EACH ROW ]
[ WHEN  ( CONDITION ) ]
BEGIN
    … …. ….
    PL/SQL Block
    … … …
END;
```

Trigger – 1

```
CREATE OR REPLACE TRIGGER Salary_check
BEFORE INSERT OR UPDATE OF Sal, Job ON Employee
FOR EACH ROW
DECLARE
   Minsal NUMBER; Maxsal NUMBER;
BEGIN
    SELECT Minsal, Maxsal INTO Minsal, Maxsal
        FROM Salgrade
        WHERE Job_classification = :new.Job;
    IF (:new.Sal < Minsal OR :new.Sal > Maxsal) THEN
      Raise_application_error(-20322,'Salary out of range
                                        for '|| :new.Job);
    END IF;
END;
```

Triggers

```
INSERT INTO EMPLOYEE VALUES
(105,'RAMKUMAR','24-MAY-1998','SA_MGR',15000,10);
```



Write a trigger to check that the FIR DATE should always be greater (or more than) than Complaint date.

Trigger – 2

```
CREATE OR REPLACE TRIGGER CHK_DATE
BEFORE INSERT ON FIR
FOR EACH ROW
DECLARE
   CDATE DATE;
BEGIN
SELECT CDATE INTO CDATE FROM COMPLAINT WHERE
                                    COMPNO = :NEW. COMPNO;
   IF CDATE>:NEW.FIRDATE THEN
           RAISE_APPLICATION_ERROR(-20300,'FIR DATE SHOULD BE
                                        > THAN COMPL DATE!');
   END IF;
END;
SQL> /
Trigger created.
```

Trigger – 2

```
SQL> INSERT INTO FIR VALUES('FIR105','CO100','04-APR-
10',13.00,'DDD','CHENNAI','IPC340','CHENNAI','MAIN
ROAD',NULL,'SAMUEL');
INSERT INTO FIR VALUES('FIR105','CO100','04-APR-
10',13.00,'DDD','CHENNAI','IPC340','CHENNAI','MAIN
ROAD',NULL,'SAMUEL')
            *
ERROR at line 1:
ORA-20300: FIR DATE SHOULD BE > THAN COMPL DATE!
ORA-06512: at "SENTHIL.CHK_DATE", line 6
ORA-04088: error during execution of trigger
'SENTHIL.CHK_DATE'
```

Trigger – 3

Try yourself:

When a complaint is FIR filed, then make an entry in the complaint history to reflect the complaint status.

Summary

- PL/SQL Introduction
- Need for PL/SQL
- Various types of PL/SQL blocks:
  - Anonymous PL/SQL Blocks using Cursors
  - Stored Procedures
  - Triggers

Reference

https://docs.oracle.com/