Modularity in PL/SQL

Functions

This presentation…

- Functions
- Raising exceptions

Functions in SQL

- We have used functions in SQL previously. e.g.

  Select title from book where lower(title) like ‘harry%’;

  Select round(months_between(sysdate, order_date)/12,1) from corder;

- We can write our own and store it in our schema.

PL/SQL Functions

- Up to now, our PL/SQL programs:
  - Took in data using &
  - Returned results using DBMS_OUTPUT.PUT_LINE
- To make them into functions:
  - Replace DECLARE with the function header
  - Replace the substitution variables with parameters.
  - Replace the printed information with returned information.
  - In the exceptions section use RAISE.

Example: Add a student

- We used tables:
  - P_stage (prog_code*, stage_code, mentor, capacity)
  - P_student (studentno, (prog_code, stage_code)*, studentname, studentaddress)
- Logic
  - Get the programme code, stage, student name and address from the user.
  - Select the capacity in the programme stage
  - Count the number of students already enrolled
  - If there is room,
    - Generate a new student number
    - Add the student and make it permanent
  - Report back the (student number and) status.

Function: Add a student

- Logic
  - **Pass the programme code, stage, student name and address into the function as parameters.**
  - Select the capacity in the programme stage
  - Count the number of students already enrolled
  - If there is room,
    - Generate a new student number
    - Add the student and make it permanent
  - **Report back**
    - **the student number**
    - **Status**

Our original program

```
SET serveroutput ON
DECLARE
  v_prog p_student.prog_code%TYPE     := '&Enter_Programme_Code';
  v_stg p_student.stage_code%TYPE     :=  &Enter_Stage_Code;
  v_name p_student.studentname%TYPE   := '&Enter_Student_Name';
  v_addr p_student.studentaddress%TYPE:= '&EnterStudent_Address';
  v_in_stage INTEGER                  := 0; -- Number currently in stage
  v_capacity INTEGER                  := 0; -- Capacity of stage
  v_sno p_student.studentno%type;

BEGIN
  SELECT capacity INTO v_capacity FROM p_stage
  WHERE (v_prog = prog_code AND v_stg = stage_code);
  SELECT COUNT(*) INTO v_in_stage FROM p_student
  WHERE (v_prog = prog_code AND v_stg = stage_code);
  IF v_in_stage < v_capacity THEN
    v_sno      := 'C17'||studseq.nextval;
    INSERT INTO p_student VALUES
      (v_sno,v_prog,v_stg,v_name,v_addr);
    COMMIT;
    dbms_output.put_line(v_name||' is added, with student number '||v_sno);
  ELSE
    dbms_output.put_line('This stage is full.  The student is not added.');
  END IF;

EXCEPTION
WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error number '||SQLCODE||
      ' meaning '||SQLERRM||'. Rolling back...');
  ROLLBACK;
END;
```

- Let's split it into 3 sections
  - Declaration
  - Executable
  - Exception

Declaration -> Function

```
DECLARE
  v_prog p_student.prog_code%TYPE     := '&Enter_Programme_Code';
  v_stg p_student.stage_code%TYPE := &Enter_Stage_Code;
  v_name p_student.studentname%TYPE   := '&Enter_Student_Name';
  v_addr p_student.studentaddress%TYPE:= '&EnterStudent_Address';
  v_in_stage INTEGER ;
  v_capacity INTEGER ;
 v_sno p_student.studentno%type;
```

```
CREATE OR REPLACE FUNCTION
ADDSTUDENT(
 P_prog p_student.prog_code%TYPE,
 P_stg p_student.stage_code%TYPE,
 P_name p_student.studentname%TYPE,
 P_addr
p_student.studentaddress%TYPE)
RETURN VARCHAR2 AS
 v_in_stage INTEGER ;
 v_capacity INTEGER ;
 v_sno p_student.studentno%type;
```

Executable sections

```
BEGIN
  SELECT capacity INTO v_capacity FROM
 p_stage
WHERE (v_prog = prog_code AND
              v_stg = stage_code);
  SELECT COUNT(*) INTO v_in_stage
  FROM p_student
  WHERE (v_prog = prog_code AND
   v_stg = stage_code);
  IF v_in_stage < v_capacity THEN
    v_sno      := 'C17'||studseq.nextval;
    INSERT INTO p_student VALUES
      (v_sno,v_prog,v_stg,v_name,v_addr);
    COMMIT;
    dbms_output.put_line(v_name||' is added,
with student number '||v_sno);
  ELSE
    dbms_output.put_line('This stage is full.
The student is not added.');
  END IF;
---exception section goes here
END;
```

```
BEGIN
  SELECT capacity INTO v_capacity
  FROM p_stage
  WHERE (p_prog = prog_code AND
              p_stg = stage_code);
  SELECT COUNT(*) INTO v_in_stage
  FROM p_student
  WHERE (p_prog = prog_code AND
             p_stg = stage_code);
  IF v_in_stage < v_capacity THEN
    v_sno      := 'C17'||studseq.nextval;
    INSERT INTO p_student VALUES
      (v_sno,p_prog,p_stg,p_name,p_addr);
    COMMIT;
    RETURN  v_sno;
  ELSE
    RETURN NULL;
END IF;
---exception section goes here
END ADDSTUDENT;
```

Exceptions

```
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(
'Error number '||SQLCODE||
      ' meaning '||SQLERRM||'.
Rolling back...');
  ROLLBACK;
```

```
EXCEPTION
WHEN OTHERS THEN
  ROLLBACK;
  RAISE;
```

/*RAISE propagates the error back to the calling environment where it can be handled by e.g. SQLException in Java*/

Function declaration

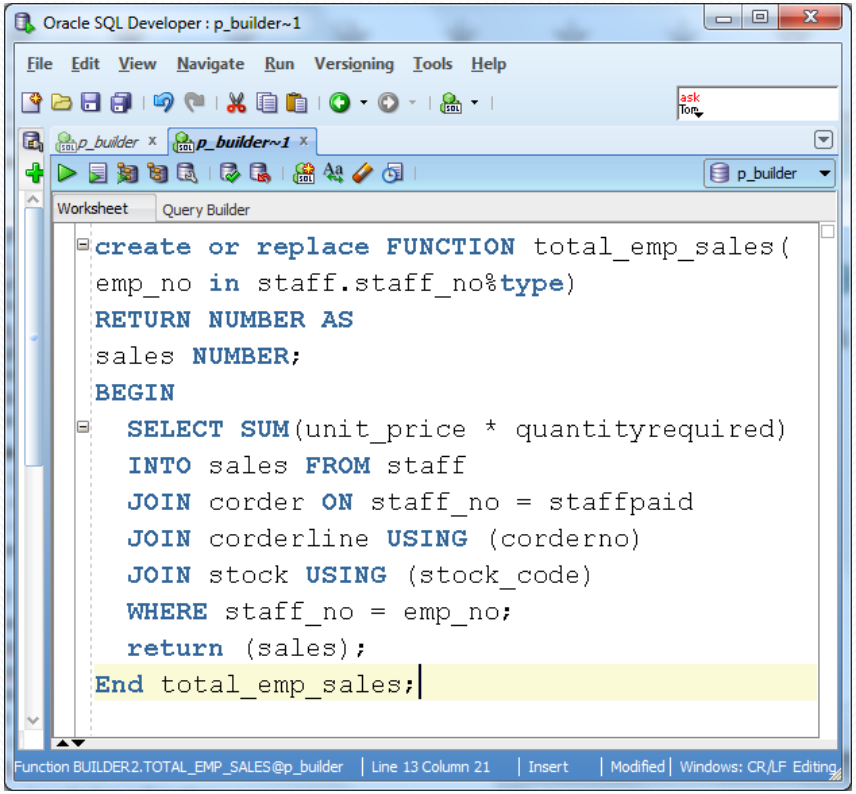
```
CREATE OR REPLACE FUNCTION
<function_name> ( <parameter_list> )
RETURN <return_datatype> AS
<declarations>
BEGIN
<pl/sql code, including
      RETURN <return_value>>;
END <function_name>;
```

return_value must have the datatype listed in <return_datatype>

Specifying a function

```
CREATE OR REPLACE FUNCTION total_emp_sales(
emp_no in staff.staff_no%type)
RETURN NUMBER AS
-- Calculate an employee's total sales
   sales NUMBER;
BEGIN
  SELECT SUM(unit_price * quantityrequired)
  INTO sales FROM staff
  JOIN corder ON staff_no = staffpaid
  JOIN corderline USING (corderno)
  JOIN stock USING (stock_code)
  WHERE staff_no = emp_no;
  return (sales);
End total_emp_sales;
```

Creating / replacing a function

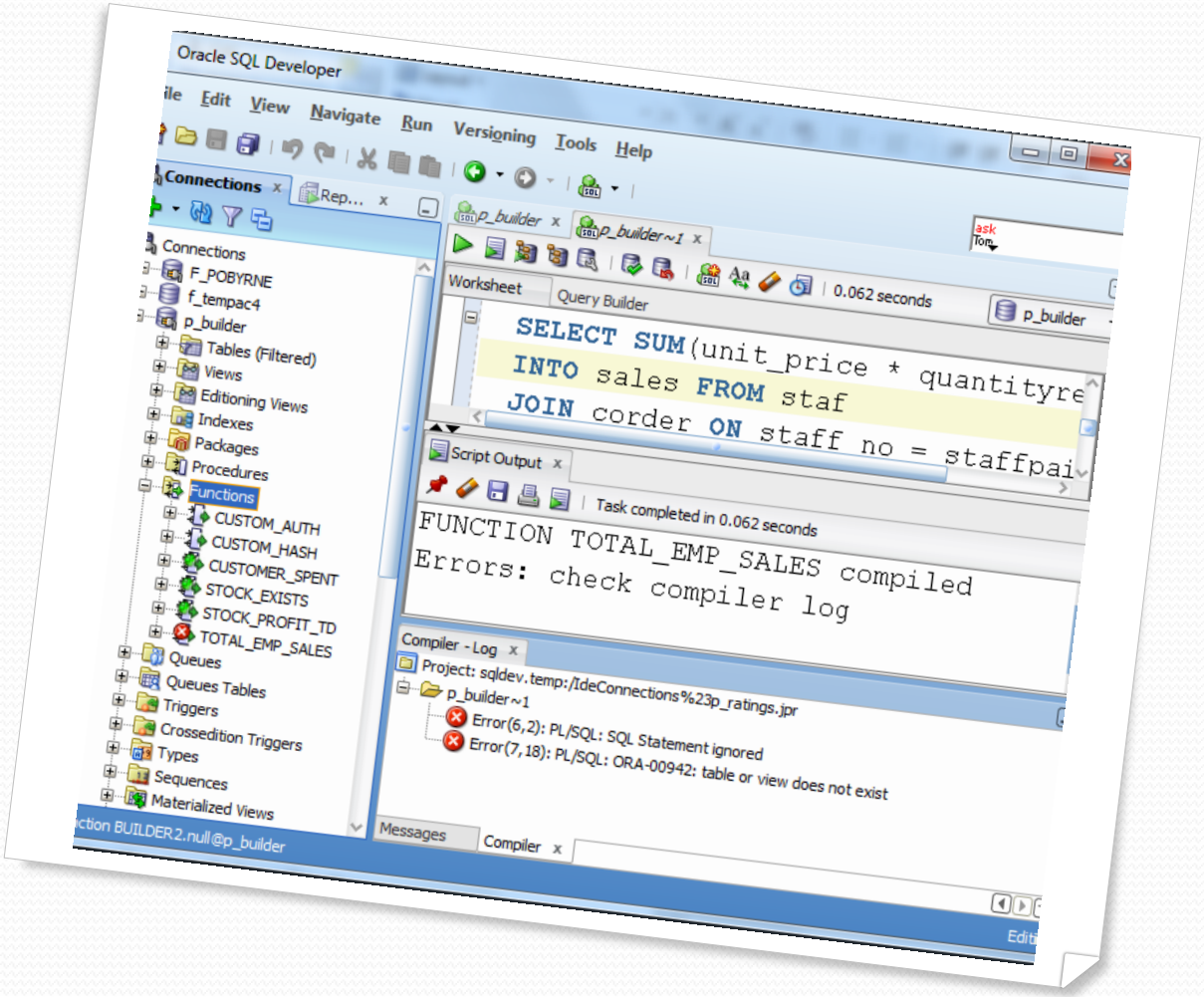


- Run the script and it will compile the function.
- If there are errors, you can debug it.

Compiled function, with error

You may need to refresh the function list to see your function.

When all errors are cleared, the function shows up green.



Calling the function

- From my own schema:
  - SELECT stock_code, stock_profit_TD (stock_code) FROM stock;
  - SELECT customer_id, customer_name, to_char (customer_spent(customer_id), 'U999,999.99') FROM customer;
  - SELECT staff_no, staff_name, total_emp_sales(staff_no) FROM staff;

To call it from another schema:

Oracle SQL Developer : p_hr

File Edit View Navigate Run Versioning Tools Help

Connections Reports

Connections
- F_POBYRNE
- f_tempac4
- p_builder
  - Tables (Filtered)
  - Views
  - Editioning Views
  - Indexes
  - Packages
  - Procedures
  - Functions
    - CUSTOM_AUTH
    - CUSTOM_HASH
    - CUSTOMER_SPENT
    - STOCK_EXISTS
    - STOCK_PROFIT_TD
    - TOTAL_EMP_SALES
  - Queues
  - Queues Tables
  - Triggers
  - Crossedition Triggers
  - Types
  - Sequences
  - Materialized Views

p_builder

Worksheet | Query Builder

```
SELECT stock_code, stock_profit_TD (st
SELECT customer_id, customer_name,
to_char(customer_spent(customer_id),'U
FROM customer;
SELECT staff_no, staff_name, total_emp
grant select on customer to hr;
```

Script Output | Query Result

SQL | All Rows Fetched: 7 in 0 seconds

	CUSTOMER_ID	CUSTOMER_NAME	TO_CHAR(CUSTOMER_SPENT(CUS
1	1	John Flaherty	€2,715
2	2	Gerard Temple	€1,000
3	3	John Browne	(null)
4	4	Aidan Bourke	€8
5	5	Mary Martin	€1,802

p_hr

Worksheet | Query Builder

```
SELECT customer_id, customer_name,
to_char(customer_spent(customer_id),'U999,999
FROM builder2.customer;
```

Query Result

SQL | Executing:SELECT customer_id, customer_name, to_char(customer_spent(customer_id),'U9...

ORA-00904: : invalid identifier
00904. 00000 - "%s: invalid identifier"
*Cause:
*Action:
Error at Line: 2 Column: 9

Function BUILDER2.null@p_builder | Line 3 Column 11 | Insert | Modified | Windows: CR/LF Editing

Granting access

- A function can only be run by a user if the user has EXECUTE privileges on it.
- From the builder2 schema:
  - GRANT EXECUTE ON <function name> TO <user / role>

In Webcourses…

- Function code ADDSTUDENT.SQL
- Java source, batch command for compile and .jar

Running Java

- Find javac.exe on your computer
- Set your path environment variable to include the folder that javac.exe is in.