Overview of Function

Function is a multi-tenant, fully managed, on-demand, highly scalable and service platform.

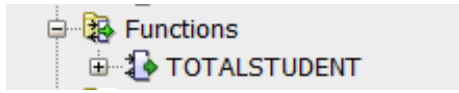
It is built on enterprise grade Cloud Infrastructure and powered by open source engine.

Using Functions to focus on writing code to meet business needs.

Example

```
CREATE OR REPLACE FUNCTION TotalStudent
RETURN number IS
total number(2) := 0;
BEGIN
SELECT count(*) into total
FROM student;
RETURN total;
END;
```

```
Function TOTALSTUDENT compiled
```



Example

```
DECLARE
c number(2);
BEGIN
c := TotalStudent();
dbms_output.put_line('Total no. of student: ' || c);
END;
```

```
Total no. of student: 6

PL/SQL procedure successfully completed.
```

Example

```
DECLARE
a number;
b number;
c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
z number;
BEGIN
```

Example

```
IF x > y THEN
z:= x;
RETURN z;
else
z:=y;
return z;
end if;RETURN z;
END;
BEGIN
a:= 23;
b:= 45;
c := findMax(a, b);
dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
```

Overview of Packages

Packages are objects that groups logically related variables, types, and subprograms.

A package will have two parts:

- ✓ Package specification.
- ✓ Package body or definition.

Overview of Packages

1. Public objects: All objects placed in the specification.

2. private object : Any subprogram not in the package specification.

Example

```
CREATE PACKAGE std_mark AS
PROCEDURE find_mark(s_id student.id%type);
END std_mark;
```

```
Package STD_MARK compiled
```

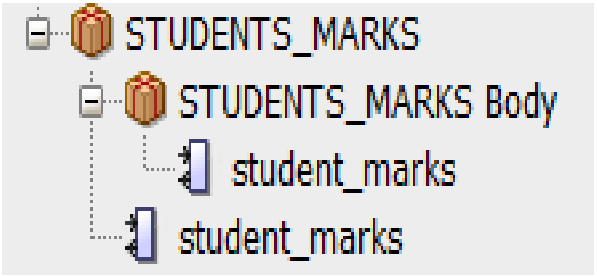
The package body has the codes for different ways declared in the package specification.

Private declarations are hidden from the code outside the package.

Package Body Example

```
CREATE PACKAGE students_marks AS
  PROCEDURE student_marks(std_id student.id%TYPE);
  END students_marks ;

CREATE OR REPLACE PACKAGE BODY students_marks AS
  PROCEDURE student_marks
    (std_id student.id%TYPE) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE
      ('Marks ' || std_id);
  END;
END students_marks;
```



Example

```
CREATE OR REPLACE PACKAGE students_package AS
PROCEDURE addstudent(std_id student.id%type,
std_name student.name%type,
std_mark student.mark%type);
PROCEDURE delstudent(std_id student.id%TYPE);
END students_package;
```

Example

```
CREATE OR REPLACE PACKAGE BODY Package_student AS
PROCEDURE addstudent(std_id student.id%type,
std_name student.name%type,
std_mark student.mark%type);
IS
BEGIN
INSERT INTO student (id,name,mark)
VALUES(std_id,std_name,std_mark);
END addstudent;
PROCEDURE delstudent(std_id student.id%TYPE)IS
BEGIN
DELETE FROM student
WHERE id = std_id;
END delstudent;
```

Example

```
DECLARE
code student.id%type:= 1;
BEGIN
Package_student.addstudent(20, 'R', 93);
Package_student.addstudent(50, 'M', 93);
Package_student.delstudent(code);
END;
```

Overview of Triggers

Triggers are written to be executed in response when any of the following events occurs:

- ✓ A database manipulation (DML) commands (DELETE, INSERT, or UPDATE).
- ✓ A database definition (DDL) commands (CREATE, ALTER, or DROP).
- ✓ A database operation (LOGOFF, SERVERERROR, STARTUP, or SHUTDOWN).

Syntax

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
```

Example

```
CREATE TABLE audits (
      audit_id          NUMBER GENERATED BY DEFAULT
      AS IDENTITY PRIMARY KEY,
      table_name        VARCHAR2(255),
      transaction_name VARCHAR2(10),
      by_user           VARCHAR2(30),
      transaction_date DATE
);
```

Example

```
CREATE OR REPLACE TRIGGER student_audit
    AFTER
    UPDATE OR DELETE
    ON student
    FOR EACH ROW
    DECLARE
    l_transaction VARCHAR2(10);
    BEGIN
         l_transaction := CASE
         WHEN UPDATING THEN 'UPDATE'
         WHEN DELETING THEN 'DELETE'
   END;
```

Example

```
INSERT INTO audits (table_name, transaction_name,
by_user, transaction_date)
VALUES('CUSTOMERS', l_transaction, USER, SYSDATE);
END;
```

Example

```
AFTER UPDATE OR DELETE ON student

UPDATE
  student
SET
   name= 'Mohammed'
WHERE
   id =1;
```

	NAME	MARK	ID
1	Mohammed	90	1
2	ali	80	2
3	sami	88	3
4	fade	85	4
5	Mohammed	50	1
6	Mohammed	90.5	1

Example

```
SELECT * FROM audits;
```

	AUDIT_ID	TABLE_NAME	TRANSACTION_NAME	BY_USER	TRANSACTION_DATE
1	1	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
2	2	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
3	3	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
4	4	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
5	5	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
6	6	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21

Example

```
DELETE FROM student
WHERE id = 2;
```

	NAME	MARK	ID
1	Mohammed	90	1
2	sami	88	3
3	fade	85	4
4	Mohammed	50	1
5	Mohammed	90.5	1

Example

```
SELECT * FROM audits;
```

	AUDIT_ID	TABLE_NAME	TRANSACTION_NAME	BY_USER	TRANSACTION_DATE
1	1	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
2	2	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
3	3	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
4	4	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
5	5	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
6	6	CUSTOMERS	UPDATE	TRAIN_USER01	09-SEP-21
7	7	CUSTOMERS	DELETE	TRAIN_USER01	09-SEP-21

Introduction to Statement-level triggers

A statement-level trigger is fired when a trigger occurs on a table regardless of number of rows are affected.

A statement-level trigger executes once for each transaction.

Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
    {BEFORE | AFTER } triggering_event
    ON table_name
    [FOLLOWS | PRECEDES another_trigger]
    [ENABLE / DISABLE ]
    [WHEN condition]
DECLARE
    declaration statements
BEGIN
    executable statements
EXCEPTION
    exception_handling statements
END;
```

Example

```
CREATE OR REPLACE TRIGGER customers_credit_trg
    BEFORE UPDATE OF credit_limit
    ON customers
DECLARE
    l_day_of_month NUMBER;
BEGIN
    -- determine the transaction type
    l_day_of_month := EXTRACT(DAY FROM sysdate);

    IF l_day_of_month BETWEEN 28 AND 31 THEN
        raise_application_error(-20100,'Cannot
update customer credit from 28th to 31st');
    END IF;
END;
```

Example

```
UPDATE
    customers
SET
    credit_limit = credit_limit * 110;
```

INSTEAD OF Triggers

An INSTEAD OF trigger allows to update data in the tables via their view which cannot be modified directly through DML statements.

Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
INSTEAD OF {INSERT | UPDATE | DELETE}
ON view_name
FOR EACH ROW
BEGIN EXCEPTION ...
END;
```

Example

```
CREATE VIEW vw_customers AS
    SELECT
        name,
        address,
        website,
        credit_limit,
        first_name,
        last_name,
        email,
        phone
    FROM
        customers
    INNER JOIN contacts USING (customer_id);
```

Example

```
INSERT INTO
    vw_customers(
        name,
        address,
        website,
        credit_limit,
        first_name,
        last_name,
        email,
        phone
    )
```

Example

```
VALUES(
    'Ahmed',
    'Irbid',
    'Ahmed.com',
    2000,
    'Ahmed',
    'mohammed',
    'ahmedmohammed@gmail.com',
    '0782944885'
);
```

```
Error at Command Line : 471 Column : 9
Error report -
SQL Error: ORA-01779: cannot modify a column which maps to a non key-preserved table
01779. 00000 -  "cannot modify a column which maps to a non key-preserved table"
*Cause:    An attempt was made to insert or update columns of a join view which
           map to a non-key-preserved table.
*Action:   Modify the underlying base tables directly.
```

Example

```
CREATE OR REPLACE TRIGGER new_customer_trg
    INSTEAD OF INSERT ON vw_customers
    FOR EACH ROW
DECLARE
    l_customer_id NUMBER;
BEGIN
    -- insert a new customer first
    INSERT   INTO   customers(name,   address,   website,
credit_limit)
    VALUES(:NEW.NAME,     :NEW.address,     :NEW.website,
:NEW.credit_limit)
    RETURNING customer_id INTO l_customer_id;
```

Example

```
 -- insert the contact
    INSERT  INTO  contacts(first_name,  last_name,  email,
phone, customer_id)
    VALUES(:NEW.first_name,  :NEW.last_name,  :NEW.email,
:NEW.phone, l_customer_id);
END;
```

Example

```
INSERT INTO
    vw_customers(
        name,
        address,
        website,
        credit_limit,
        first_name,
        last_name,
        email,
        phone
    )
```

Example

```
VALUES(
    'Ahmed',
    'Irbid',
    'Ahmed.com',
    2000,
    'Ahmed',
    'mohammed',
    'ahmedmohammed@gmail.com',
    '0782944885'
);
```

```
1 row inserted.
```

Example

```
SELECT * FROM customers;
```

	CUSTOMER_ID	NAME	ADDRESS	WEBSITE	CREDIT_LIMIT
1	320	Ahmed	Irbid	Ahmed.com	2000

```
SELECT * FROM contacts;
```

	CONTACT_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE	CUSTOMER_ID
1	320	Ahmed	mohammed	ahmedmohammed@gmail.com	0782944885	320