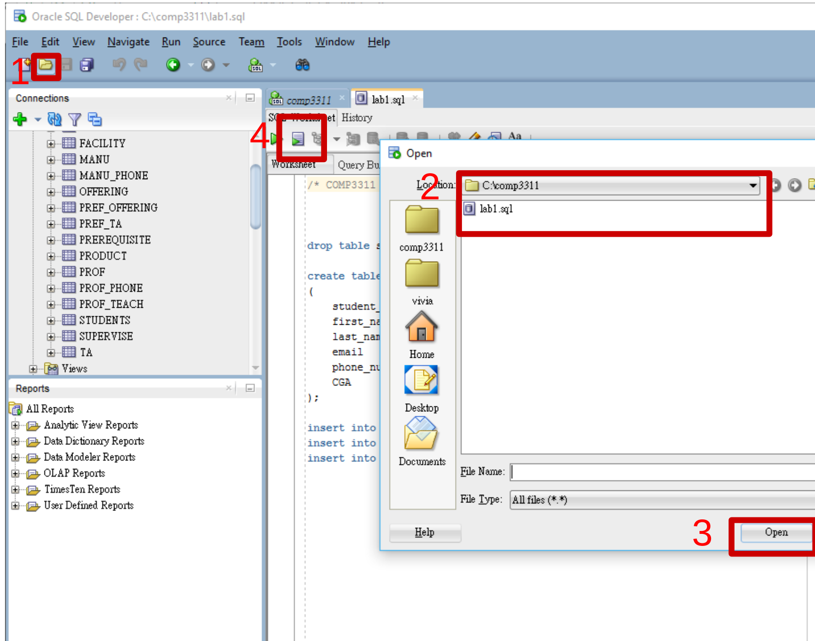
COMP 3311 Database Management Systems

Lab 5. PL/SQL, cursors and triggers

Downloading and running the lab SQL script file

- login Oracle database server using SQL Developer with your Oracle account
- Download (save) the lab5.sql file to local file system
  - http://course.cs.ust.hk/comp3311/labs/lab5.sql
- Open file
- Run script
- The tables created last time were dropped.
- Some new tables are created.



Download PL/SQL files

- Download PL/SQL files to local file system
  - lab5_plsql1.sql
  - lab5_plsql2.sql
  - lab5_plsql3.sql
  - lab5_plsql4.sql
  - lab5_cursor1.sql
  - lab5_cursor2.sql
  - lab5_trigger1.sql
  - lab5_trigger2.sql

Introduction to PL/SQL

- PL/SQL stands for Procedural Language/SQL.
- Basic unit in PL/SQL is called a block.
- PL/SQL extends the capabilities of SQL by adding to it the functionalities that are supported by procedural languages.

Basic Structure of PL/SQL

```
DECLARE
/* Declarative section: variables, types, and local subprograms. */

BEGIN
/* Executable section: procedural and SQL statements go here. */
/* This is the only section of the block that is required. */

EXCEPTION
/* Exception handling section: error handling statements go here. */

END;
```

More about PL/SQL

- PL/SQL is case insensitive
- C style comments /*…*/
- The SQL statements allowed in a PL/SQL program are SELECT, INSERT, UPDATE and DELETE.
- Data definition language like CREATE, DROP, ALTER are not allowed.
- In PL/SQL we used the ":=" operator to assign values to a variable.
- The "=" operator is for comparison.

Data type supported in PL/SQL

- One of the types supported by SQL for defining the columns (i.e. NUMBER, INTEGER, CHAR, VARCHAR2, DATE, TIMESTAMP, etc).
- Types declared to be of the same types as some database columns.
- Some generic types.

Declaring Variables

- Declares a variable of the type number
    - DECLARE
      count NUMBER;
- Declares a variable with the same type as the no_of_projectors column in the facility table.
    - DECLARE
      projectors facility.no_of_projectors%TYPE;
- Declares a variable which is the same type as a row (record).
    - DECLARE
      facility_record facility%ROWTYPE;

PL/SQL Example 1

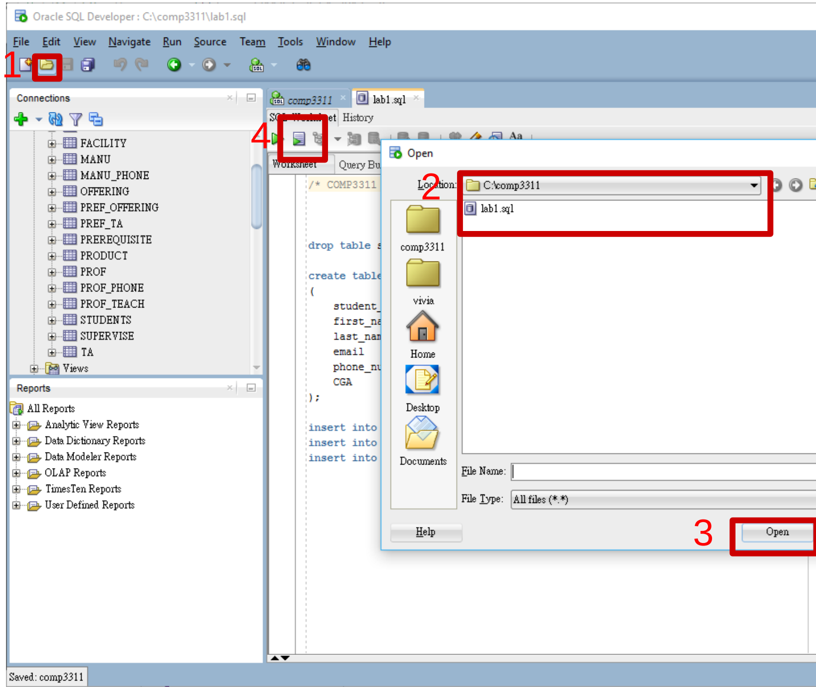
- A simple PL/SQL that extract information from the departments table to a table called math_dept (lab5_plsql1.sql):

```
DECLARE
dept_name departments.name%TYPE;
dept_room departments.room_number%TYPE;
   BEGIN
SELECT name, room_number INTO dept_name,dept_room FROM departments
WHERE department_id='MATH';
     INSERT INTO math_dept values (dept_name,dept_room);
   END;
.
run;
```

- The dot "." indicates the end of the PL/SQL code.
- The statement "run;" tells the database engine to execute the PL/SQL codes defined, we can use "/" to replace "." +"run;"

Running the PL/SQL statements

- Open file from local file system (e.g.lab5_plsql1.sql)
- Run script just like sql statement
- Below statement should be written to Script Output
  - PL/SQL procedure successfully completed.
- For qlsql1, check data of table MATH_DEPT after running PL/SQL



```
/* COMP3311

drop table s

create table
(
    student_
    first_na
    last_nam
    email
    phone_nu
    CGA
);

insert into
insert into
insert into
```

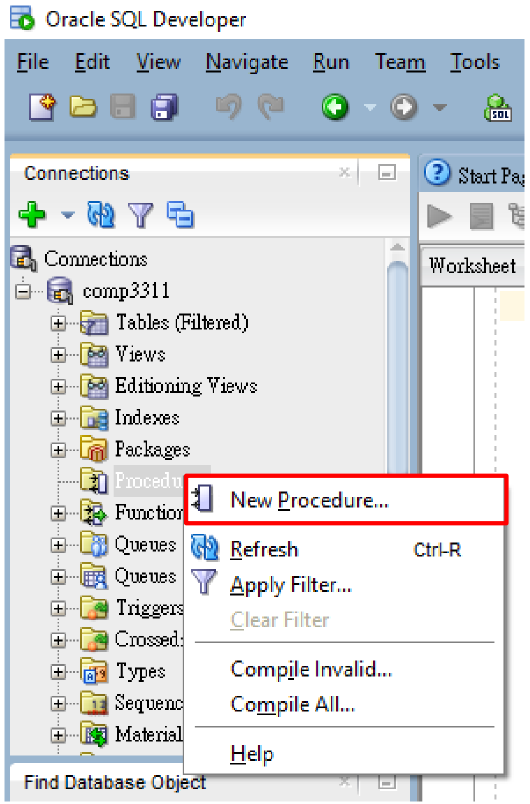
PL/SQL Example 2

Flow Control: IF-THEN-ELSE-END IF (lab5_plsql2.sql)

```
DECLARE
        room departments.room_number%TYPE;
  BEGIN
        SELECT room_number INTO room FROM departments
        WHERE department_id='COMP';
        IF (room>3000 and room<4000) THEN
                UPDATE departments SET room_number=room+2000
                WHERE department_id='COMP';
        ELSE
                UPDATE departments SET room_number=5528
                WHERE department_id='COMP';
        END IF;
  END;
  /
```

Appendix: Create a PL/SQL Procedure

- Connect to the database
- Right-click the Procedures node in the schema hierarchy on the left side
- Select "New Procedure..."
- Enter procedure name and parameters (if necessary) using the Create PL/SQL Procedure dialog box



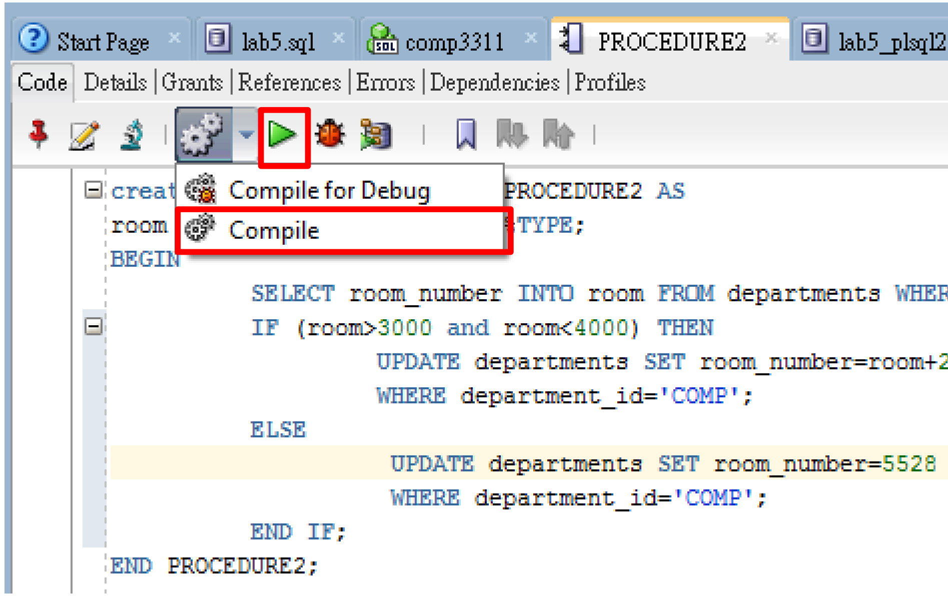
Appendix: Procedure

Turn example 2(lab5_plsql2.sql) into procedure

```
create or replace PROCEDURE PROCEDURE2 AS
room departments.room_number%TYPE;
BEGIN
    SELECT room_number INTO room FROM departments WHERE department_id='COMP';
    IF (room>3000 and room<4000) THEN
            UPDATE departments SET room_number=room+2000
            WHERE department_id='COMP';
    ELSE
            UPDATE departments SET room_number=5528
            WHERE department_id='COMP';
    END IF;
END PROCEDURE2;
```

Appendix: Compilation

- Commit changes before run PL/SQL procedure
  - open SQL worksheet
  - enter “commit;”
  - run statement
- Click Compile
- Click Run
  - Connecting to the database comp3311.
  - Process exited.
  - Disconnecting from the database comp3311.



```
creat                          PROCEDURE2 AS
room                           TYPE;
BEGIN
        SELECT room_number INTO room FROM departments WHER
        IF (room>3000 and room<4000) THEN
                UPDATE departments SET room_number=room+2
                WHERE department_id='COMP';
        ELSE
                UPDATE departments SET room_number=5528
                WHERE department_id='COMP';
        END IF;
END PROCEDURE2;
```

PL/SQL Example 3

Flow control: LOOP (lab5_plsql3.sql)

```
DECLARE
                i testloop.i%TYPE :=1;
    BEGIN
    LOOP
                INSERT INTO testloop VALUES (i);
                i:=i+1;
                EXIT WHEN i>10;
    END LOOP;
    END;
    /
```

A LOOP can be terminated by the EXIT WHEN keyword

PL/SQL Example 4

Flow control: FOR LOOP (lab5_plsql4.sql)

```
DECLARE
        i NUMBER(2):=1;
    BEGIN
        FOR R IN (
                SELECT * FROM facility
        )LOOP
        UPDATE facility SET no_of_computers= no_of_computers+i WHERE
        department_id=R.department_id;
        i:=i+1;
    END LOOP;
    END;
    /
```

VAR is a variable local to the for-loop and need not be declared. R is the VAR in the above example

Appendix: PL/SQL Debugging

- SQL developer allows:
    - Setting and/or removal of breakpoint
    - Monitoring and manipulation of variables
- “Complie for Debug” must be executed to make an object available for debugging
- Once complied for Debugging, whenever executed in “Debug” mode the code will stop where directed

Appendix: Prerequisites of debugging

- Unlock the user. Login to SQL Developer as the SYS user and execute the following commands:
  - alter user <username> identified by <password> account unlock;
  - grant debug connect session to <username>;
  - grant debug any procedure to <username>;
- We **don’t** have enough privilege for lab, but you can try it with your own database server

Introduction to Cursor 1

- The SELECT statement in PL/SQL can only fetch a single record.
- If the query returns more than one records, a cursor is needed.
- A cursor is like a pointer that points to a single record each time.
- Using the cursor, the records can be fetched in a one-by-one manner.

Introduction to Cursor 2

- A cursor should be defined at the DECLARE section of the PL/SQL codes.
- It needs to be activated by the OPEN command.
- Then we can FETCH the records in a oneby-one manner.
- When all the records are fetched, “%NOTFOUND” will return a true (will see its details 2 slides later) .
- We need to CLOSE the cursor after using it, so as to free up the resources.

Syntax of a Cursor

```
DECLARE
        CURSOR cursor_name
        IS select_statement;
```

An example:

```
  DECLARE
        CURSOR facility_cursor
        IS SELECT      department_id, name, no_of_projectors,
                              no_of_computers
        FROM facility;
```

The above cursor fetches all the records from the facility table.

Status of a Cursor

- Getting the status of a cursor:
  - cursor_name%NOTFOUND
    Whether the previous fetch has failed.
  - cursor_name%FOUND
    Whether the previous fetch is successful.
  - cursor_name%ROWCOUNT
    Number of records fetched so far.
  - cursor_name%ISOPEN
    Is the cursor still open.

Example of Cursor 1

Incorporating the Cursor to PL/SQL codes (lab5_cursor1.sql) :

```
DECLARE
   var_deptid facility.department_id%TYPE;
   var_name  facility.name%TYPE;
   CURSOR facility_cursor
   IS SELECT department_id, name FROM facility;
BEGIN
   OPEN facility_cursor;
   LOOP
      FETCH facility_cursor INTO var_deptid,var_name;
      EXIT WHEN facility_cursor%NOTFOUND;
      INSERT INTO test VALUES (var_deptid,var_name);
   END LOOP;
   CLOSE facility_cursor;
END;
/
```

The above cursor fetches records from the facility table, and insert the values one by one into another table called test.

Example of Cursor 2

Cursor loop (lab5_cursor2.sql) :

```
DECLARE
    var_deptid facility.department_id%TYPE;
   var_name  facility.name%TYPE;
   CURSOR facility_cursor
   IS SELECT department_id, name FROM facility;
BEGIN
            FOR rec in facility_cursor
            LOOP
                        var_deptid:=rec.department_id;
                        var_name:=rec.name;
                        INSERT INTO test VALUES (var_deptid,var_name);
            END LOOP;
END;
/
```

Example of Cursor 2

- The facility_cursor on the previous slide is automatically opened by the FOR LOOP.
- The rec is a rowtype data, but there is no need for us to declare it.
- Codes inside the LOOP are execute once for each row of the cursor, and each time the two columns department_id, and name are copied into rec.
- We can access the data in rec directly (as shown in the codes).
- LOOP terminates automatically once all the records in the cursor are fetched.
- The cursor is then closed automatically.

Triggers

- Triggers are stored PL/SQL blocks associated with a table, a schema, or the database, or anonymous PL/SQL blocks or calls to a procedure implemented in PL/SQL or Java.
- The trigger is automatically executed when the specified conditions occur.

Syntax for Creating a Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name

[BEFORE | AFTER | INSTEAD OF] database_event

[REFERENCING [OLD AS old_name] [NEW AS new_name]]

trigger Level

[WHEN criteria]

BEGIN

 trigger body [PL/SQL blocks]

END;
```

Semantics of Trigger

- BEFORE: if this keyword presents, the trigger will be started before each of the affected rows has been changed.
- AFTER: if this keyword presents, the trigger will be started after each of the affected rows has been changed.
- INSTEAD OF: if this keyword presents, the trigger will be started instead of performing the DML on the view
- Database_event:indicates the specific database events that will start the trigger
- FOR EACH ROW: the trigger will be started once for each row (record).
- WHEN: specifies the trigger condition.
- NEW: this keyword refers to a new record retrieved
- OLD: this keyword refers to an exisiting record.

Example of Trigger 1

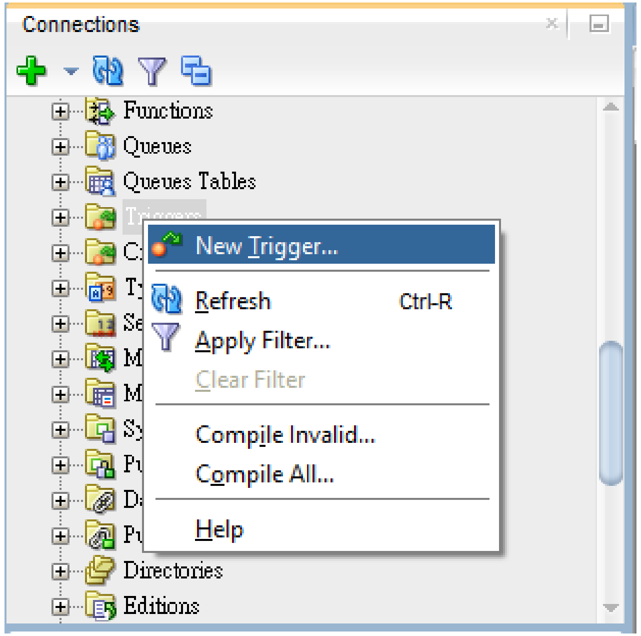
- The following trigger adds a prefix to the email address for the CS students when a new CS student record is being inserted. (lab5_trigger1.sql)

```
CREATE OR REPLACE TRIGGER chk_email
      BEFORE INSERT ON students
FOR EACH ROW
      WHEN (NEW.department_id = 'COMP')
DECLARE
     prefix CHAR(3) := 'cs_';
BEGIN
     :NEW.email := prefix || :NEW.email;
END;
/
```

- Note that the red part of the codes is a PL/SQL block.

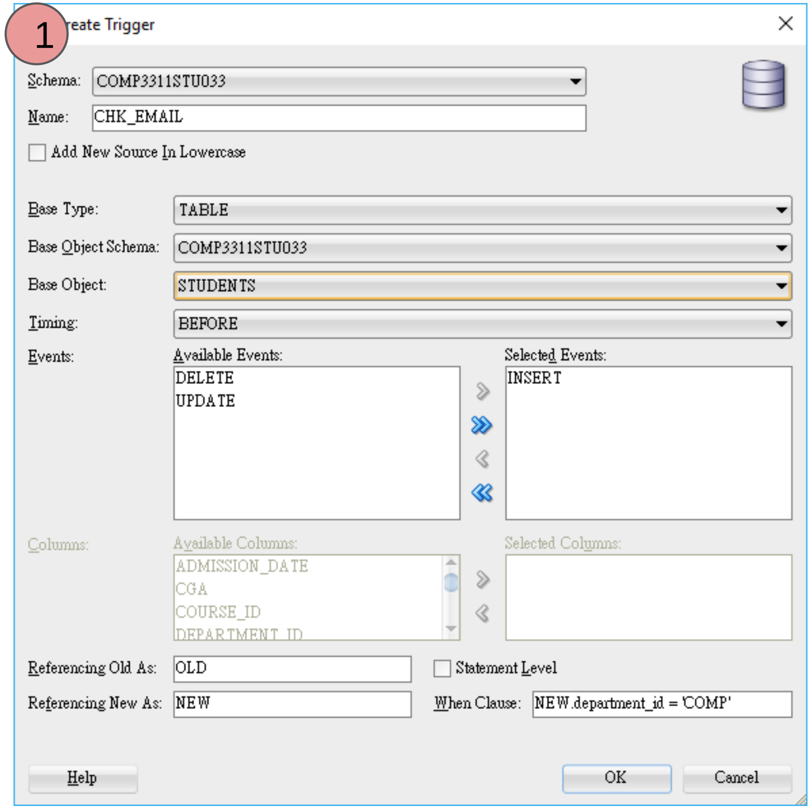
Appendix: Create Trigger with SQL Developer

- Right click on Trigger
- Select “New Trigger...”
- Try to enter basic information for trigger chk_email (in previous slide)
- Hint:
  Statement Level or Row Level: For a trigger on a table, Statement Level fires the trigger once before or after the triggering statement that meets the optional trigger constraint defined in the WHEN condition; Row Level fires the trigger once for each row that is affected by the triggering statement and that meets the optional trigger constraint defined in the WHEN condition.



Appendix: Create Trigger with SQL Developer

- Name: CHK_EMAIL
- Base Type: TABLE
- Base Object: STUDENTS
- Timing: BEFORE
- Events: INSERT
- Statement Level: false
- When Clause:
  NEW.department_id = 'COMP'
- Click "OK"



Schema:	COMP3311STU033
Name:	CHK_EMAIL
Base Type:	TABLE
Base Object Schema:	COMP3311STU033
Base Object:	STUDENTS
Timing:	BEFORE

Events:

Available Events:	Selected Events:
DELETE	INSERT
UPDATE	

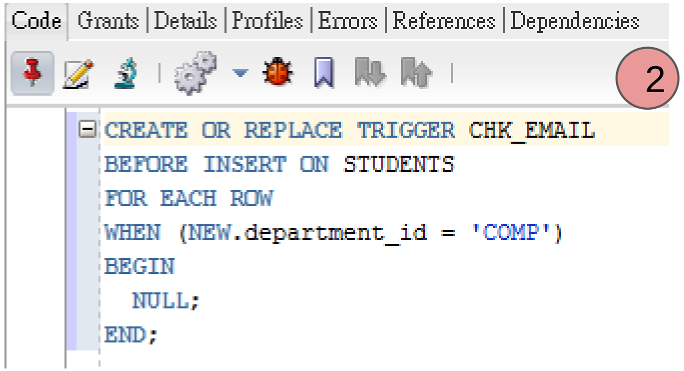
Columns:

Available Columns:	Selected Columns:
ADMISSION_DATE	
CGA	
COURSE_ID	
DEPARTMENT_ID	

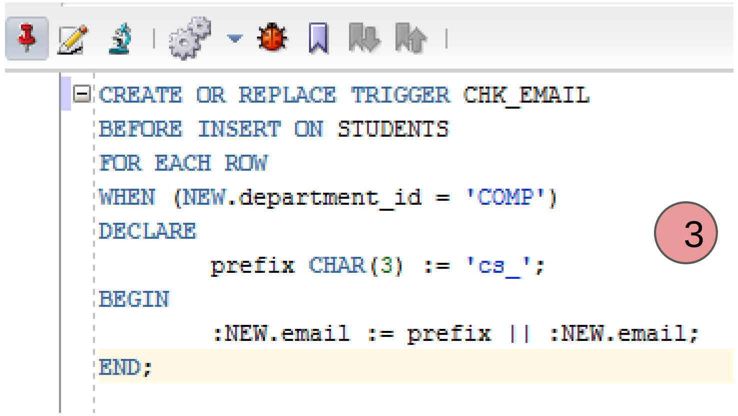
Referencing Old As:	OLD	Statement Level	☐
Referencing New As:	NEW	When Clause:	NEW.department_id = 'COMP'

Appendix: Create Trigger with SQL Developer 2

Then you will get

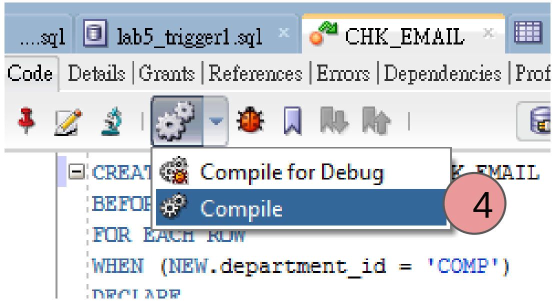

Code | Grants | Details | Profiles | Errors | References | Dependencies
2

```
CREATE OR REPLACE TRIGGER CHK_EMAIL
BEFORE INSERT ON STUDENTS
FOR EACH ROW
WHEN (NEW.department_id = 'COMP')
BEGIN
  NULL;
END;
```



```
CREATE OR REPLACE TRIGGER CHK_EMAIL
BEFORE INSERT ON STUDENTS
FOR EACH ROW
WHEN (NEW.department_id = 'COMP')
DECLARE
        prefix CHAR(3) := 'cs_';
BEGIN
        :NEW.email := prefix || :NEW.email;
END;
```

- Type the trigger body [PL/SQL blocks]
- Compile the trigger


....sql
lab5_trigger1.sql
CHK_EMAIL
Code | Details | Grants | References | Errors | Dependencies | Prof
Compile for Debug
Compile
4

```
CREAT          K_EMAIL
BEFOR
FOR EACH ROW
WHEN (NEW.department_id = 'COMP')
DECLARE
```

Appendix: Test Trigger

- Insert new record department_id=”COMP” that into table students
    - open SQL worksheet
    - enter insert statement, for example:
        - insert into students values (03456789, 'Rowling', 'Joanne', 'JK', 23781234, 11.50, 'COMP','03-SEP-82', 'COMP197');
    - run statement
    - open data of table students or select * from students
- The trigger should adds a prefix ”cs_” to the email address to COMP students
- You can also try to insert record that department are not COMP

Example of Trigger 2

- The following example backs up the record for the facility table in the old_facility table, if the record is to be removed from the facility table. (lab5_trigger2.sql)

```
CREATE OR REPLACE TRIGGER backup_facility
        BEFORE DELETE ON facility
FOR EACH ROW
DECLARE
        id_null EXCEPTION;
BEGIN
        INSERT INTO old_facility
        VALUES (:old.department_id, :old.name, :old.no_of_projectors,
        :old.no_of_computers);
EXCEPTION
        WHEN id_null THEN
        DBMS_OUTPUT.PUT_LINE('Department ID missing');
END;
/
```

- Note that the red part is also a PL/SQL block.

Test Trigger 2

- Open lab5_trigger2.sql from local file system
- Run script
  - Script Output: Trigger BACKUP_FACILITY compiled
- Delete record from table facility
  - open SQL worksheet
  - enter delete statement, for example:
    - delete from facility where department_id = 'MATH';
  - run statement
  - open data of table facility and old_facility
- The facility delete should be saved to table old_facility

Conclusion

- We covered the following topics in this lab:
    - Simple PL/SQL syntax.
    - PL/SQL procedure
    - Building Cursors with PL/SQL.
    - Building triggers with PL/SQL.