School of Information Technology
Department of Computer Science

COS326 Database Systems:
Practical 4 2025

Release Date: 19 August 2025

Submission Date: 04 September 2025 @ 23:59 Hrs

Lecturer: Mr S.M Makura

Total: 50 Marks

A. Objectives

1. Practice the use of advanced features of PostgreSQL ORDBMS.
2. Learn how to implement PL/pgSQL functions, triggers, and database constraints using triggers.

B. Submission Procedure:

You should have PostgreSQL installed on your computer in order to complete this practical. When you are done:

1. You must submit files, named:
   - **a.** **EERD.pdf** which contains the database design as an EER diagram. Your name and student number must appear in this document.
   - **b.** **CreateStatements.sql** which contains all statements necessary to create the database 'objects' i.e. user defined types, sequences, tables, functions and triggers.
   - **c.** **InsertQueries.sql** which contains all statements that add to the content of the database (INSERT statements).

d. **SelectQueries.sql** which contains all statements that provide reports from the database (SELECT statements).

e. **TestTriggerQueries.sql** which contains all statements for testing the triggers (INSERT, UPDATE, and DELETE statements).

f. Compress the above documents into an archive (zip file) and upload it to ClickUP using the Practical 4 submission link **before** the due date and time. The file name for the archive must have your student number as part of the file name, e.g. ***uXXXXXXXX__Surname__Initials***(*XXXXXXXX is your student number) e.g u12345678_Smith_JN*

2. The practical will be marked through a live demo on Discord.

NO LATE submissions will be accepted after the submission date and time has lapsed. Do not wait till the last minute to submit and start giving excuses that you faced technical challenges when you tried to submit.

Question 1: PostgresSQL ORDMS

Scenario continuation from Practical 3

Codealot Pvt Ltd were very impressed by your task you did in practical 3. They have decided to test you further by extending the database design you created in practical 3. This time around, they would like you to create functions, triggers and trigger functions.

Source: Makura S.M (2025)

Improvements to the practical 3 database design

A simple database design was used in practical 3. In this practical exercise, the database design specification is extended as follows:

A. It is necessary to ensure that all data that is entered in the database is valid. While the DBMS can check values against built-in and user-defined data types, there are some aspects of data values the DBMS does not have the ability to check, unless it is provided with triggers and special (trigger) functions for this purpose.

It is necessary to ensure that:

(1) each *provincialCode* in a Full-Time employee's *provincialRegistration* array is a valid code which already exists in the Province table (referential integrity).

(2) the *provincialCode* in a Full-Time employee's *provincialRegistration* array cannot be duplicated.

(3) each *employeeNumber* for an E m p l o y e e ' s record is a valid code which already exists in the Employee table (referential integrity).

Since *provincialRegistration* is an array, referential integrity cannot be implemented using *foreign keys*. A straightforward object-relational database design for the *employeeCode* referential integrity constraint would be to implement the **Province** table with a *foreign key* that references a row in the **Employee** table (by specifying REFERENCES **Employee**). Unfortunately, in PostgreSQL 17.5 inheritance and foreign keys are not compatible. (refer to chapter 5 section 5.11 of the PostgreSQL 17.5 documentation for details). So, for the database implementation in this practical exercise we will use triggers to enforce the referential integrity between the tables **Employee, Contract, Full-Time, Part-Time** and **Province.**

B. It is also necessary to record the details of all the database users who perform DELETE operations on the **Full-Time** and **Part-Time** tables as well as the data they have deleted.

The following table summarises the above aspects of the required database design for this practical exercise. Some of these aspects are similar to practical 3.

Database 'object' type	**Database 'objects'**	**Description**
SEQUENCE (same as Prac3)	`empSeq`	generates surrogate (primary) keys for Employee table
	`provSeq`	generates surrogate (primary) keys for the Province table
	`contractSeq`	generates surrogate (primary) keys for the Contract, Full-Time and Part-Time tables
TYPE (same as Prac3)	`Title`	description of titles with values: Ms, Mev, Miss, Mrs, Mr, Mnr, Dr, Prof
	`Full_Name`	ROW type holding (title, firstName, surname)
TABLE (same as Prac3)	`Employee`	with attributes: employeeKey, employeeNumber, employName, employeeYearHired, employeeDOB
	`Province`	with attributes: provincialKey, provincialCode, provinceName, department
	`Contract`	with attributes: contractKey, contract_id, contractType, contractNumberOfYears
	`Full-Time`	inherits Employee and additionally has attribute: provincialRegistration
	`Part-Time`	inherits Employee and additionally has attributes: mentor
These are new tables	`DeletedFullTime`	When an Fulltime employee record is deleted, the deleted record is copied to this table. Attributes are: (all the attributes of **Fulltime** , plus date-and-time of deletion, userid of the user who deleted the record)
	`DeletedPartTime`	When a Parttime student record is deleted, the deleted record is copied to this table. Attributes are: (all the attributes of **Partime** , plus date-and-time of deletion, userid of the user who deleted the record)

FUNCTION (you must create PL/pgSQL functions)	`personFullNames (NameType)`	returns a text string with the title, first name and surname
	`ageInYears (DATE)`	returns a student's age in years
	`isLocatedAt (text, text[ ])`	returns a Boolean value to indicate if the full time employee is working for the at the province with the provincial code passed in the first parameter.
	`isValidProvincialCode`	returns true if the given provincialCode value is one of the p r o v i n c i a l codes in the Province table.
	`hasValidProvincialCodes`	returns true if the new record to be inserted into the FullTime table has a valid provincial code which exist in the
	`hasDuplicateProvincialCodes`	returns true if the new record to be inserted into the Fulltime table has duplicate provincial codes. **HINT: use a FOREACH loop within a FOREACH loop.**
	`isValidEmployeeNumber`	returns true if the new record to be inserted into the Fulltime table has a valid employee number which exists in the Employee table.
TRIGGER PROCE - DURE (you must create PL/pgSQL functions)	`check_valid_contract_code`	Trigger procedure called by *check_valid_contract*. Return type is TRIGGER.
	`check_valid_provincial_codes`	Trigger procedure called by *check_valid_provincial_registration.* Return type is TRIGGER. Checks that all provincial codes are valid and there are no duplicates.
	`record_delete_fulltime, record_delete_parttime,`	Trigger procedures called by *audit_delete_fulltime, audit_delete_parttime* Return type is TRIGGER. *record_delete_fulltime* writes data for the deleted fulltime employee in the **DeletedFullTime** table and *record_delete_parttime* writes data for the deleted Partime employee in the **DeletedPartTime** table.

TRIGGERS	`check_valid_contract`	Triggers on the Contract, Fulltime and Parttime tables for INSERT & UPDATE operations. Checks that the employeeContractCode is valid. Uses the *check_valid_contract_code* function.
	`check_valid_provincial_registration`	Trigger on the Fulltime table for INSERT & UPDATE operations. Checks the provincial codes in the provincialRegistration array for a Full time employee. Uses the *check_valid_provincial_codes* function.
	`audit_delete_fulltime` `audit_delete_parttime`	Triggers on the Fulltime and Parttime tables for DELETE operations. Uses functions *record_delete_fulltime* and *record_delete_parttime*

Task 1: Creation of the database 'objects' [30 marks]

1. Using PostgreSQL, write SQL statements and PL/pgSQL code to create all the database 'objects' in the above table and any other 'objects' you consider to be necessary. **Note:** you should re-use the CREATE statements for practical 3 for creating types, sequences and tables.

2. **Create a database in PostgreSQL called *employeesDBprac4* and run all the SQL statements in (1) above to create the database 'objects'. Note:** marks for part (2) will only be awarded if the database 'objects' actually get created. Marks will be awarded as follows:

a.	Sequences and types:	no marks
b.	Tables from practical 3:	no marks
c.	New tables	2 marks
d.	Re-implementation of Practical 3 functions using PL/pgSQL:	4 marks
e.	Implementation of the new functions	10 marks
f.	Functions for triggers:	8 marks

g.	Triggers:	6 marks

Task 2: Inserting data into the Database tables [no marks]

Re-use the *INSERT INTO* SQL statements of practical 3 to add the following data (same as practical 3) into the database. Execute some SELECT statements to confirm that you entered the data correctly.

	Attribute values: note that the values of attributes **... key** are generated by the SEQUENCEs that you created							
Contract	**Contract key**	**Contract Code**	**Contract Type**	**Number of years**				
		CAL 113	Full Time	5				
		CAL 114	Part Time	2				
Province	**Province key**	**Provincial code**	**Province name**	**Department**				
		GP	Gauteng	Software Development				
		WC	Western Cape	Auditing				
		FS	Free State	Finance				
		NW	North West	Human Resources				
Full Time	**Employee key**	**Employee number**	**Employee name (title, fname, surname)**	**Date of birth (dd-mm- yyyy)**	**Contract code**	**Year Hired**	**provincialRegistration**	
		140010	choose title & names	10-01-1999	CAL 113	2010	NW	
		140015	choose title & names	25-05-1997	CAL 113	2017	GP, NW	
		131120	choose title & names	30-01-1997	CAL 113	2020	FS	
		131140	choose title & names	20-02-1998	CAL 113	2023	WC	
Part Time	**Employee key**	**Employee number**	**Employee name (title, fname,**	**Date of birth (dd-mm- yyyy)**	**Contract code**	**Year Hired**	**Mentor (title,fname, sname)**	

		101122	choose title & names	15-06-2009	CAL 114	2022	choose title & names
		121101	choose title & names	27-04-2007	CAL 114	2021	choose title & names

Task 3: SELECT statements to test the functions [10 marks]

1. Write a SELECT statement that will list the following details of all Employees working at the North West province. The query must use the functions: *personFullNames, ageinYears, isLocatedAt.*
(2 marks)

2. Write two SELECT statements to demonstrate that the *hasValidProvinceCodes* function provides the correct result.
(4 marks)

3. Write two SELECT statements to demonstrate that the *hasDuplicateProvinceCodes* function provides the correct result.
(4 marks)

Task 4: Insert, update and delete operations to test the triggers [10 marks]

Write and test SQL statements to do the operations listed in the table below. Marks will only be awarded if the SQL statement produces the correct output.

Operation	**Table**	**values / actions**	**Testing trigger (for)**	**Marks**
INSERT	Fulltime	A new record with an invalid contract code	*check_valid_contract* (referential integrity)	1
INSERT	Fulltime	A new record with a provincialRegistration array which has an invalid provincial code	*check_valid_provincial_registration* (referential integrity)	1
INSERT	Parttime	A new record with an invalid contract code	*check_valid_contract* (referential integrity)	1

UPDATE	Fulltime	to change the contract code of an existing record to an invalid contract code	*check_valid_contract* (referential integrity)	1
UPDATE	Fulltime	to change the provincialRegistration array of an existing record to another array which has an invalid provincial code	*check_valid_provincial_codes* (referential integrity)	1
UPDATE	Parttime	to change the contract code of an existing record to an invalid contract code	*check_valid_contract* (referential integrity)	1
DELETE	Fulltime	delete one existing record	*audit_delete_fulltime* (auditing)	2
DELETE	Fulltime	delete one existing record	*audit_delete_parttime* (auditing)	2