



**L-Università ta' Malta**  
Faculty of Information &  
Communication Technology

Department of  
Computer Information  
Systems

# **Introduction to Databases and Information Management Coursework**

Matthias Bartolo\* (0436103L), Jerome Agius\* (0353803L), Isaac Muscat\* (0265203L)

2021/2022 SEMESTER 2

\*Bachelor of Science in Information Tech (Hons) Artificial Intelligence

---

Study-unit: **Introduction to Databases and Information Management**

Code: **CIS1043**

Lecturer: **Dr Michel Camilleri**

## Task 1 – Setup of PostgreSQL and PGAdmin and loading of Sample Database

Below, find a step-by-step guide on how to download PostgreSQL and PGAdmin and how to load the sample database 'scott'.

### Set-up PostgreSQL

1. Download PostgreSQL from this [link](#).
2. Run the application as administrator.
3. Follow the setup wizard. (Note: Do not install pgAdmin from this download)
4. Install PostgreSQL.
5. In Stack Builder, select 'PostgreSQL 12 (x64) on port 5432.
6. Unselect all Add-ons, tools and utilities and select all database drivers.
7. Follow the Stack Builder steps, leave everything as default.
8. Install the necessary dependencies.

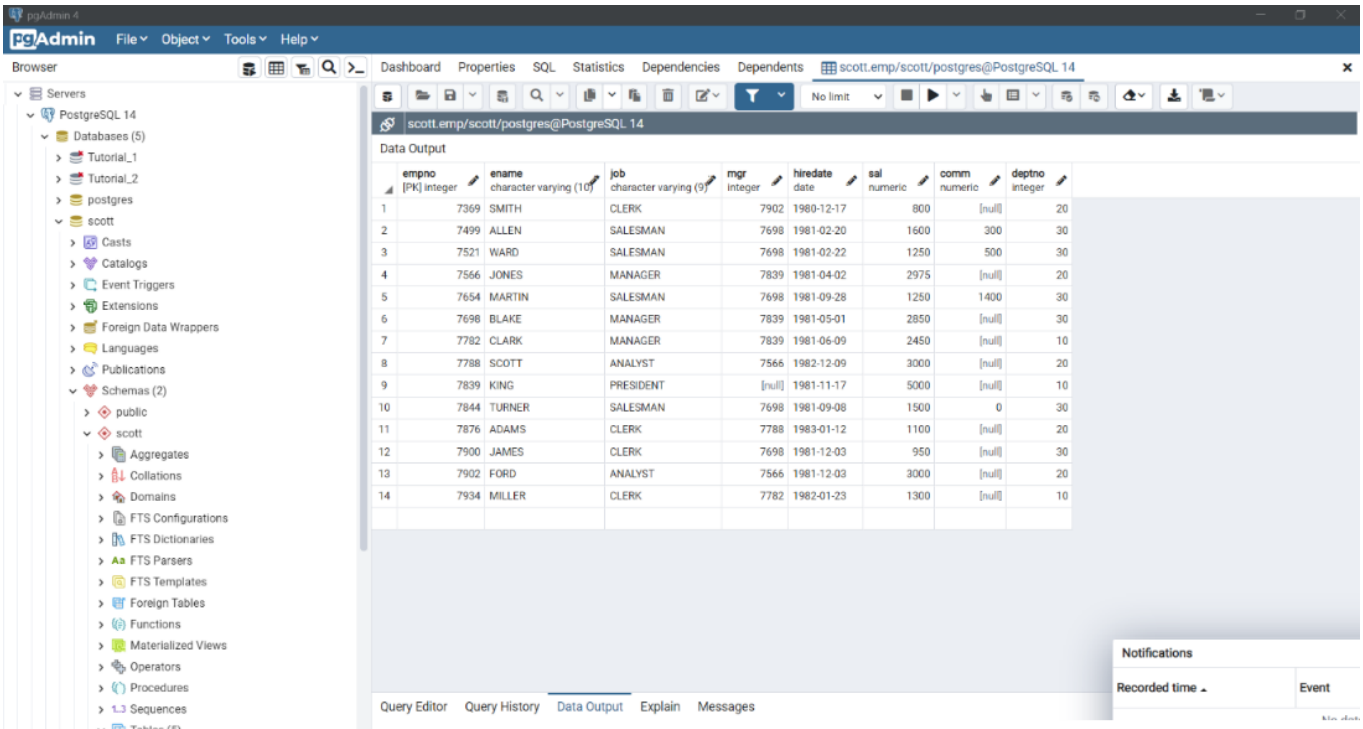
### Set-up pgAdmin

1. Download pgadmin4-4.16-x86.exe from this [link](#).
2. Run the application as administrator.
3. Save the pgAdmin password.
4. Open pgAdmin.
5. Type pgAdmin and exec in the Start/Search menu.
6. Choose the server and enter the password.
7. Execute an SQL command to check if set-up is successful.

## Load Scott Database

1. Install the Scott Database file from the University VLE.
2. Ensure the PostgreSQL server is started.
3. Use pgAdmin to connect with the server.
4. Create a new database.
5. Fill in the properties page.
6. Choose UTF8 in the definition page.
7. Click OK when done.
8. Right click on the database name and choose restore.
9. Select custom as a format.
10. Input the appropriate filename.
11. Select postgres as a rolename.
12. Tick the sections pre-data, data and post-data.
13. Click restore.
14. Refresh the database key.

The final product after completing all these steps should be as follows:



The screenshot shows the pgAdmin 4 interface. The left pane displays the database tree structure, with 'scott' selected under 'Databases (5)'. The right pane shows the 'Data Output' for the 'scott.emp' table, displaying 14 rows of employee data. The data is as follows:

empno	ename	job	mgr	hiredate	sal	comm	deptno
1	7369 SMITH	CLERK	7902	1980-12-17	800	[null]	20
2	7499 ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
3	7521 WARD	SALESMAN	7698	1981-02-22	1250	500	30
4	7566 JONES	MANAGER	7839	1981-04-02	2975	[null]	20
5	7654 MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
6	7698 BLAKE	MANAGER	7839	1981-05-01	2850	[null]	30
7	7782 CLARK	MANAGER	7839	1981-06-09	2450	[null]	10
8	7788 SCOTT	ANALYST	7566	1982-12-09	3000	[null]	20
9	7839 KING	PRESIDENT	[null]	1981-11-17	5000	[null]	10
10	7844 TURNER	SALESMAN	7698	1981-09-08	1500	0	30
11	7876 ADAMS	CLERK	7788	1983-01-12	1100	[null]	20
12	7900 JAMES	CLERK	7698	1981-12-03	950	[null]	30
13	7902 FORD	ANALYST	7566	1981-12-03	3000	[null]	20
14	7934 MILLER	CLERK	7782	1982-01-23	1300	[null]	10

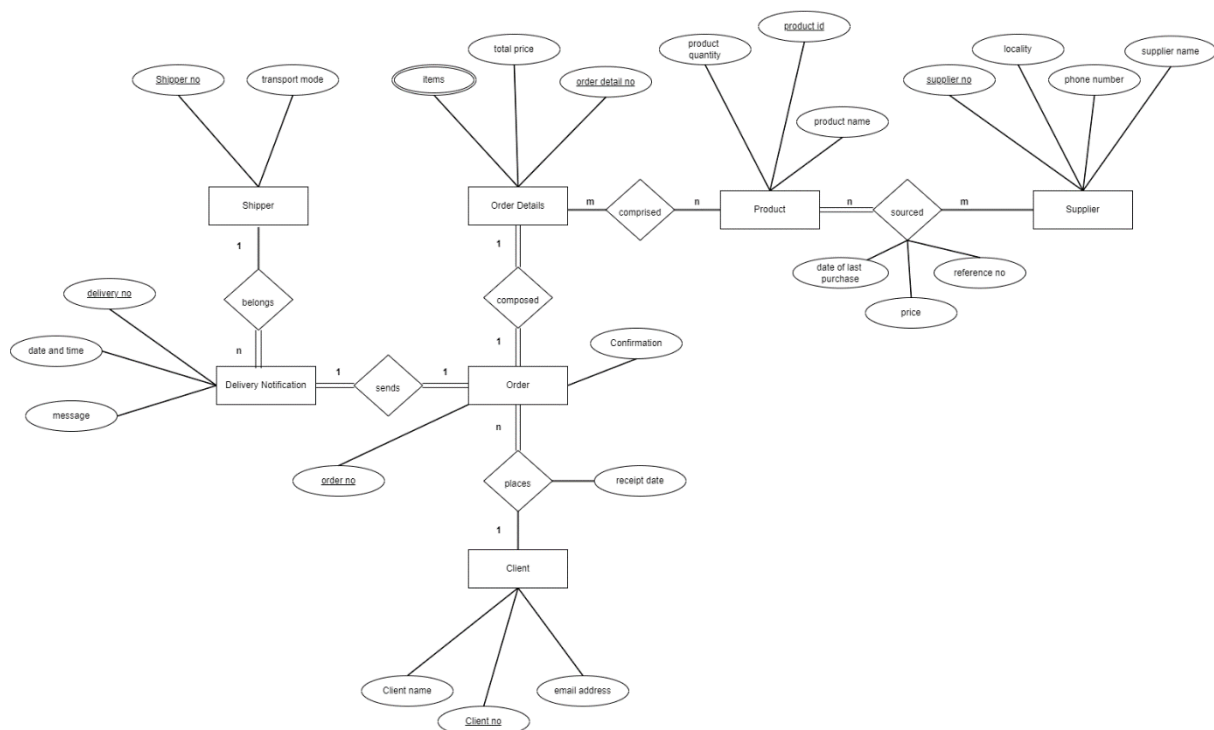
## Task 2 – Design of Entity Relationship models and DDL statements

### Question A

Produce an ERM diagram and submit a copy at the end of session. One can follow the steps:

1. Identify and draw the main look-up entities (e.g. client);
  - a. Add some sensible attributes that an order-processing system needs to function;
2. Build the relationship between entities product and suppliers;
3. Introduce the entities for order and order details and their relationship.
  - a. Relate order and order details entities with the other look-up entities (e.g. client and product).
4. Build a relationship between client and order.
5. Build a relationship between order and delivery note.
6. Build a relationship between delivery note and shipper.
7. Re-check design by considering the following:
  - a. Attributes in place and whether composite or multiple options considered?
  - b. Is there a primary key in each entity?
  - c. Are all relationships properly structured (e.g. between two entities)? Are cardinality and participation constraints in place?
  - d. Is there a weak entity (and therefore weak relationship)?

ER Diagram A)



## Question B

i) Create a schema in your PostgreSQL database to implement the following exercises.

*During the following work it's best to help yourself with test data. Remember integrity constraints, e.g. not null, will not get enabled (ie command will give errors) if the data present does not fit the constraint.*

First, we created the tables and then we created their relationships.

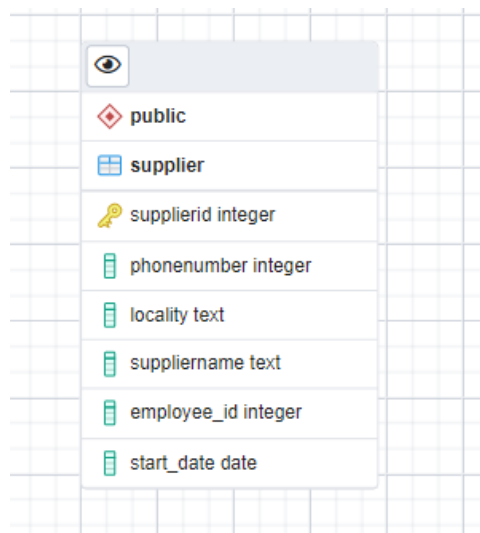
The following was used to create the supplier table:

```
CREATE TABLE IF NOT EXISTS public.supplier
(
    supplierid integer NOT NULL,
    phonenumber integer,
    locality text COLLATE pg_catalog."default",
    suppliername text COLLATE pg_catalog."default",
    employee_id integer,
    start_date date DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT supplier_pkey PRIMARY KEY
(supplierid)
);
```

The following table was created:

	supplierid [PK] integer	phonenumber integer	locality text	suppliername text	employee_id integer	start_date date

The following ER diagram was created:



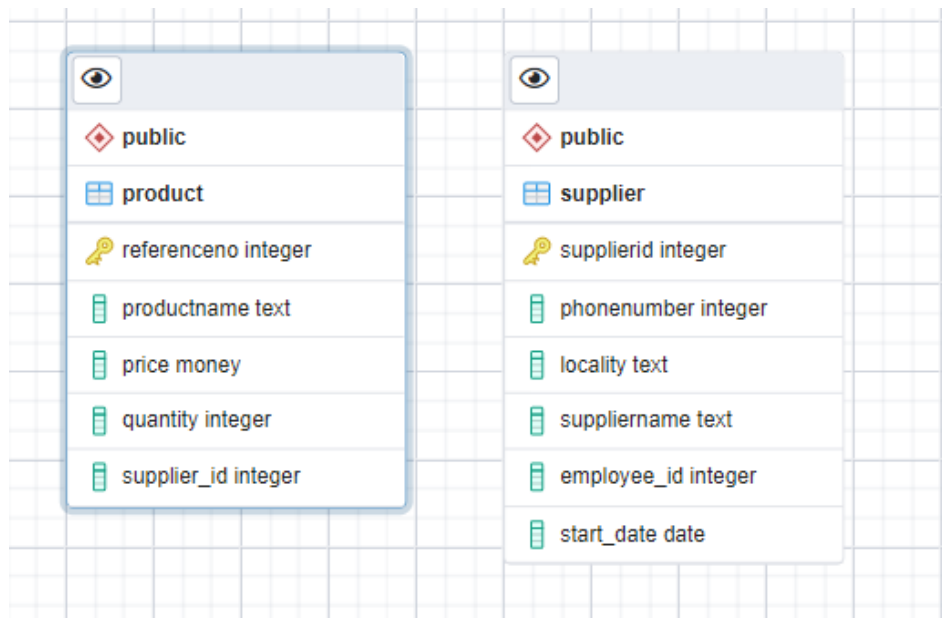
The following was used to create the product table:

```
CREATE TABLE IF NOT EXISTS public.product
(
    referenceno integer NOT NULL,
    productname text COLLATE pg_catalog."default",
    price money, quantity integer,
    supplier_id integer,
    CONSTRAINT product_pkey PRIMARY KEY
(referenceno)
);
```

The following table was created:

	referenceno [PK] integer	productname... text	price money	quantity integer	supplier_id integer

The following ER diagram was created:



The following code was used to create the employee table:

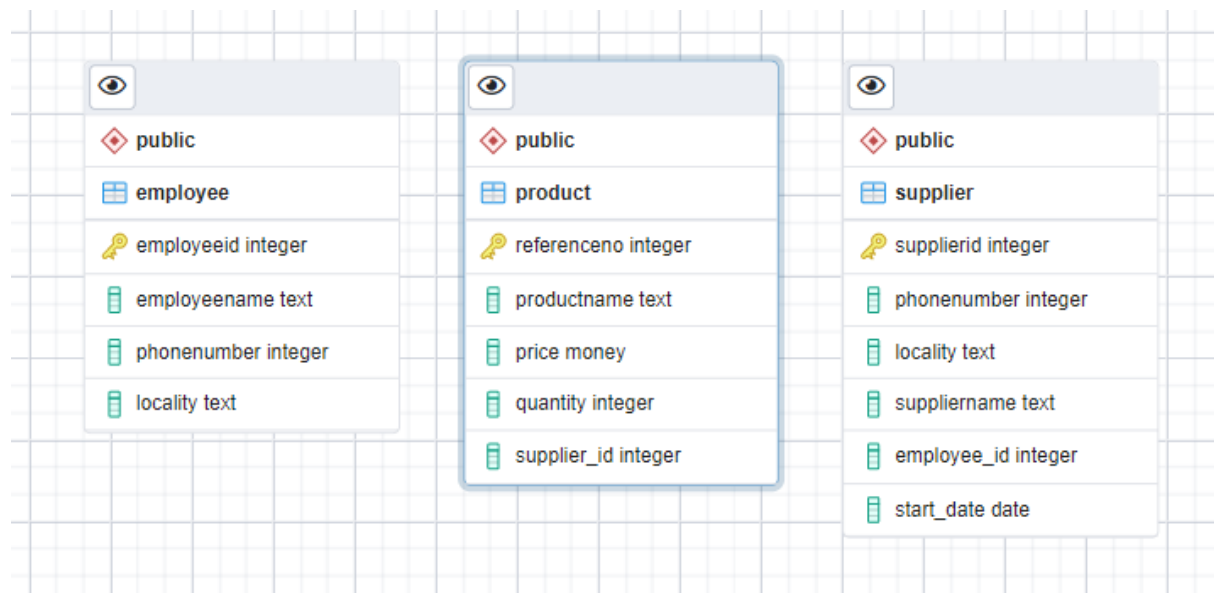
```

CREATE TABLE IF NOT EXISTS public.employee
(
    employeeid integer NOT NULL,
    employeename text COLLATE pg_catalog."default",
    phonenumber integer,
    locality text COLLATE pg_catalog."default",
    CONSTRAINT employee_pkey PRIMARY KEY
    (employeeid)
);
  
```

The following table was created:

	employeeid [PK] integer	employeename text	phonenumber integer	locality text

The following ER diagram was created:



The following was used to create the warehouse table:

```
CREATE TABLE IF NOT EXISTS public.warehouse
```

```
(
```

```
    warehouseno integer NOT NULL,
```

```
    locality text COLLATE pg_catalog."default",
```

```
    warehousesize double precision,
```

```
    product_no integer,
```

```
    CONSTRAINT warehouse_pkey PRIMARY KEY
```

```
    (warehouseno)
```

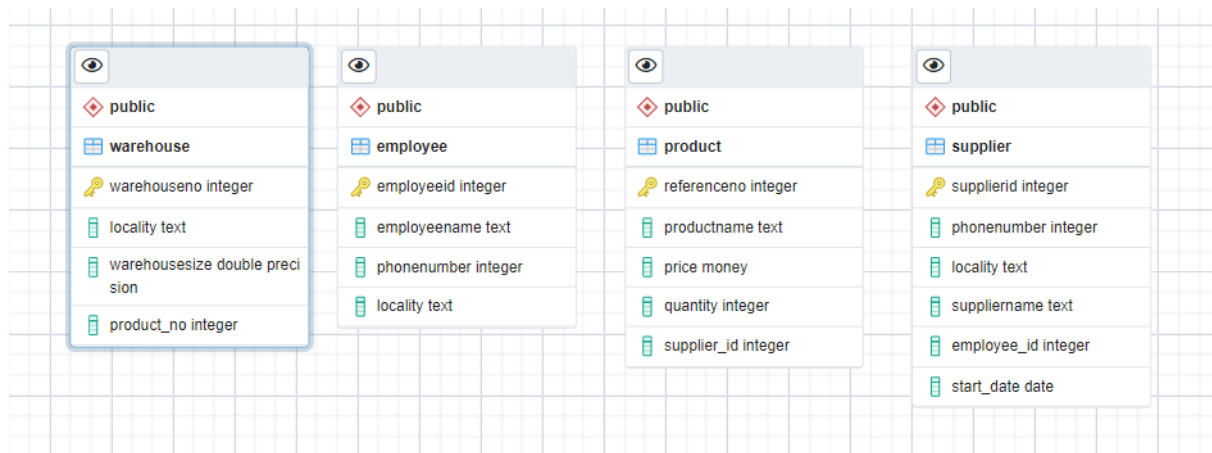
```
);
```

The following table was created:

	warehouseno [PK] integer	locality text	warehousesize double precision	product_no integer



The following ER diagram was created:



The following was used to create the relationship between Supplier and Product:

```
ALTER TABLE IF EXISTS public.product
```

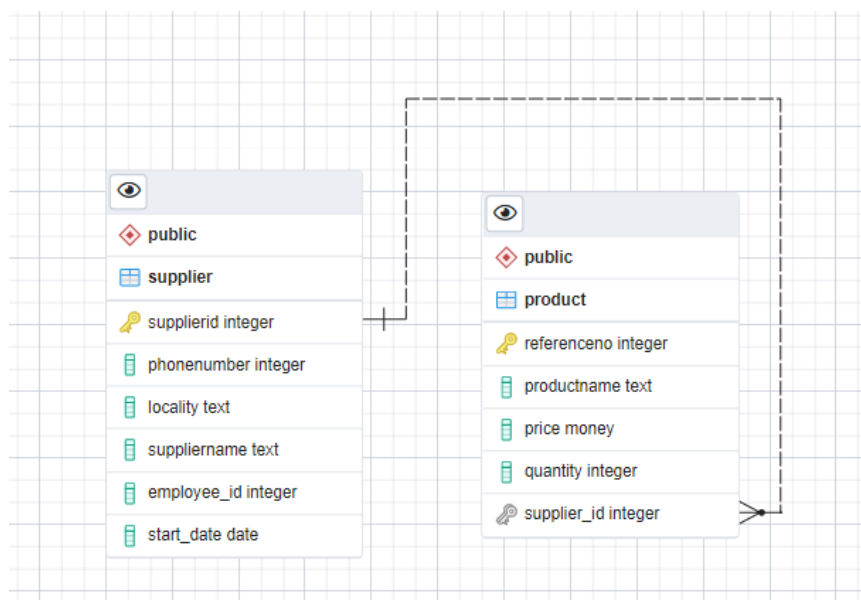
```
ADD CONSTRAINT supplier_fk FOREIGN KEY (supplier_id)
```

```
REFERENCES public.supplier (supplierid) MATCH SIMPLE
```

```
ON UPDATE NO ACTION
```

```
ON DELETE NO ACTION;
```

The following ER diagram was created:



The following was used to create the relationship between Employee and Supplier:

```
ALTER TABLE IF EXISTS public.supplier
```

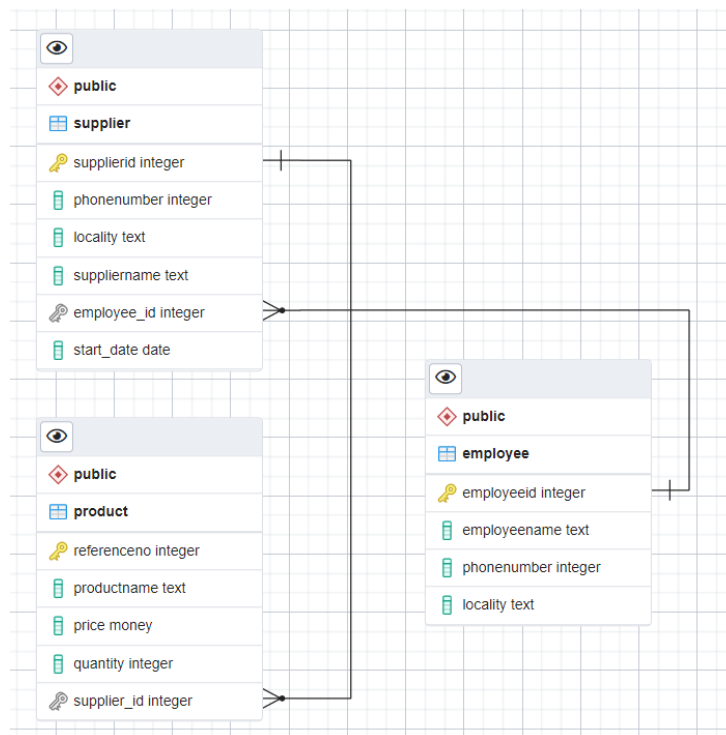
```
ADD CONSTRAINT employee_fk FOREIGN KEY (employee_id)
```

```
REFERENCES public.employee (employeeid) MATCH SIMPLE
```

```
ON UPDATE NO ACTION
```

```
ON DELETE NO ACTION;
```

The following ER diagram was created:



The following was used to create the product\_warehouse\_junction table:

```
CREATE TABLE IF NOT EXISTS
```

```
public.product_warehouse_junction
```

```
(
```

```
    warehouse_no integer NOT NULL,
```

```
    reference_no integer NOT NULL,
```

```
    items_on_stock integer,
```

```
    cost_price integer,
```

```
    CONSTRAINT product_warehouse_junction_pkey PRIMARY KEY (warehouse_no,  
reference_no)
```

```
);
```

The following table was created:

	<b>warehouse_no</b> [PK] integer	<b>reference_no</b> [PK] integer	<b>items_on_stock</b> integer	<b>cost_price</b> integer

The following was used to create the relationship between Product and product\_warehouse\_junction:

```
ALTER TABLE IF EXISTS
```

```
public.product_warehouse_junction
```

```
    ADD CONSTRAINT reference_fk FOREIGN KEY (reference_no)
```

```
    REFERENCES public.product (referenceno) MATCH SIMPLE
```

```
    ON UPDATE NO ACTION
```

```
    ON DELETE NO ACTION;
```

The following was used to create the relationship between warehouse and product\_warehouse\_junction:

```
ALTER TABLE IF EXISTS
```

```
public.product_warehouse_junction
```

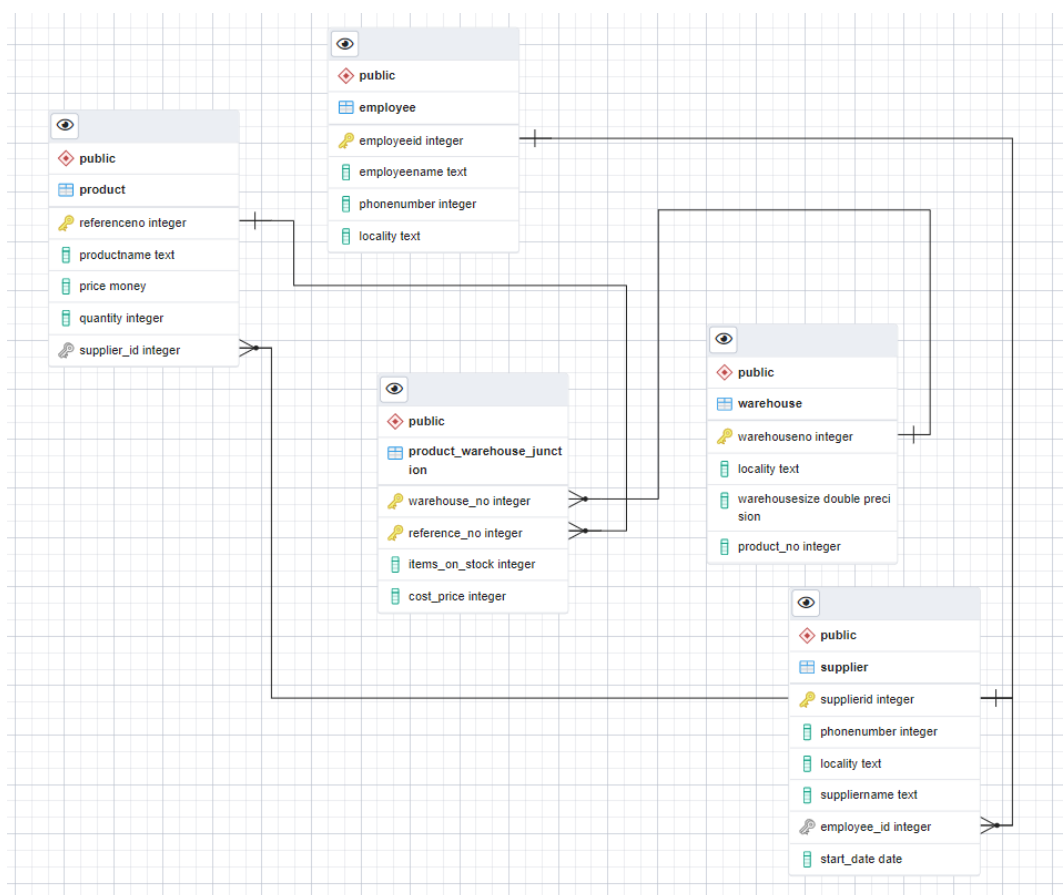
```
ADD CONSTRAINT warehouse_fk FOREIGN KEY (warehouse_no)
```

```
REFERENCES public.warehouse (warehouseno) MATCH SIMPLE
```

```
ON UPDATE NO ACTION
```

```
ON DELETE NO ACTION;
```




The following ER diagram was created:



The following was used to create the sub\_employee table:

```
CREATE TABLE IF NOT EXISTS public.sub_employee
(  
    employee_fk integer,  
    employee_id integer  
);
```

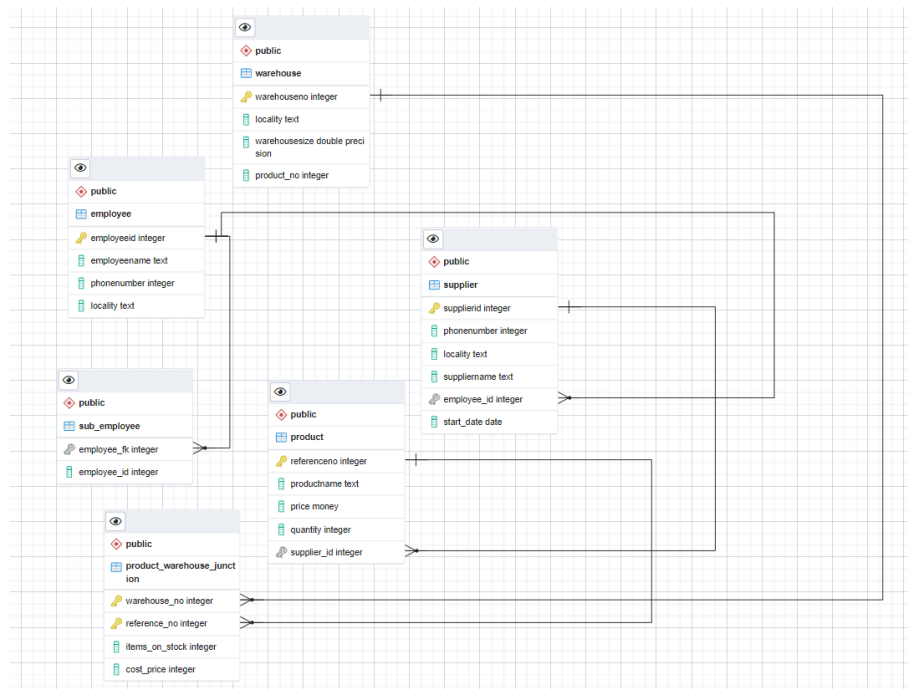
The following table was created:

	<b>employee_fk</b> integer 	<b>employee_id</b> integer 
---	---	---

The following was used to create the relationship between employee and sub\_employee:

```
ALTER TABLE IF EXISTS public.sub_employee  
    ADD CONSTRAINT employee_fk FOREIGN KEY (employee_fk)  
    REFERENCES public.employee (employeeid) MATCH SIMPLE  
    ON UPDATE NO ACTION  
    ON DELETE NO ACTION;
```

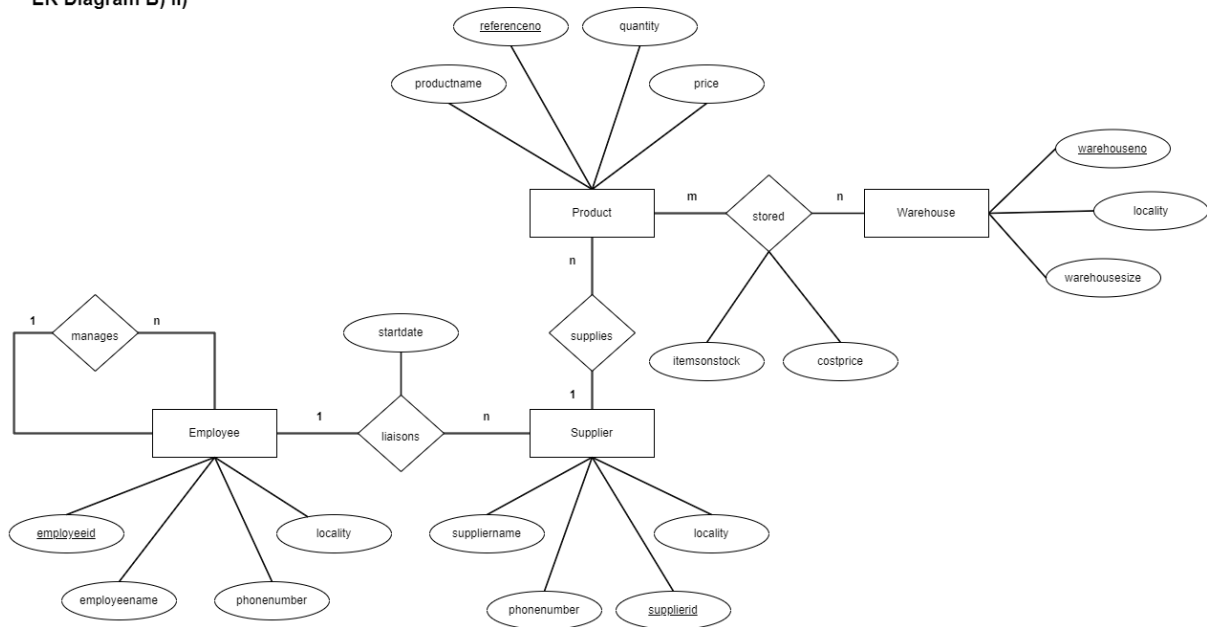
The following ER diagram was created:



ii) Draw the following ERMs and convert these to relational tables with SQL's Data Definition Language constructs.  
(Do not use triggers or cascading updates in this exercise).

1. Entities **Supplier** and **Product** are related with a 1(p)-N(t) relationship (Supplier supplies these products).
2. Entities **Employee** and **Supplier** are related with a 1(p)-N(t) relationship adorned with start-date attribute (Employee liaisons with many suppliers).
3. Entities **Product** and **Warehouse** are related with a M(p)-N(p) relationship adorned with items on stock and cost price attributes.
4. An employee can manage many other employees' instances.

ER Diagram B) ii)





### Task 3 – SQL practice

- 1 Get full details of all jobs (aka projects).
- 2 Get the names of all jobs in London.
- 3 Get a unique list of cities where jobs are being undertaken.
- 4 For all works with part 'P1' or 'P2', get the works details (e.g attributes s, p, j) and work out three quantities. The first is attribute qty less 20%, the second the actual quantity, and the third quantity plus 12%. Ensure that all numbers on output are of integer data type and that output headers have indicative names.
- 5 Print suppliers details that are located in 'PARIS' and their status is '10'.
- 6 Print suppliers details that are either located in 'LONDON' or their status is '10'.
- 7 Print works schedule with quantity greater than (or to equal) 400 and have supplier 'S5' and part 'P5'.
- 8 Print suppliers' details that are located in 'PARIS' and their status is not '10'.
- 9 Print all work schedule details that have part 'P2' and quantity involved is from 100 to 300.
- 10 Retrieve work schedules details where its quantity is either 100, 300, 500, 700 or 900.
- 11 Which work schedules involve suppliers with their respective status being a low '10'?
- 12 For each work schedule row give supplier name and product name and quantity involved. Sort the output by supplier and product names.
- 13 For each work schedule row give supplier name, product name, job name and quantity for suppliers' status '10', product colour 'RED' and city 'PARIS'.
- 14 For project 'J4' work out the bill of material (qty \* weight) of each part involved.
- 15 Which triplets of supplier, product and jobs are co-located (have the same city)?
- 16 For every job name match every possible product colour! (This is a PRODUCT join – check slides). Sort the output by job and colour.
- 17 Is there any job row without a value in the city attribute? Output the row with lacking value.
- 18 Is there any works row without a supplier, product or job?
- 19 Is there any product tuple that does not have their weight greater than (or equal) to 0?
- 20 Is there any product in spj (works) not present in the product table?

### Question 1

SELECT \* FROM date.j;

Data Returned:

	<b>j</b> [PK] character (20)	<b>jname</b> character (20)	<b>city</b> character (20)
1	J1	SORTER	PARIS
2	J2	DISPLAY	ROME
3	J3	OCR	ATHENS
4	J4	CONSOLE	ATHENS
5	J5	RAID	LONDON
6	J6	EDS	OSLO
7	J7	TAPE	LONDON

### Question 2

SELECT j.jname AS "Job\_Names" FROM date.j

WHERE j.city = 'LONDON';

Data Returned:

	<b>Job_Names</b> character (20)
1	RAID
2	TAPE

### Question 3

```
SELECT DISTINCT j.city AS "Cities" FROM date.j
```

```
WHERE j.jname IS NOT NULL;
```

Data Returned:

	Cities character (20) 🔒
1	ROME
2	OSLO
3	LONDON
4	PARIS
5	ATHENS

### Question 4

```
SELECT CAST(sum(spj.qty)*0.8 AS int) AS "-20%", sum(spj.qty) AS "Actual Quantity",  
CAST(sum(spj.qty)*1.12 AS int) AS "+12%"
```

```
FROM date.spj
```

```
WHERE spj.p = 'P1' OR spj.p = 'P2';
```

Data Returned:

	-20% integer 🔒	Actual Quantity bigint 🔒	+12% integer 🔒
1	1040	1300	1456

### Question 5

```
SELECT *
```

```
FROM date.s
```

```
WHERE s.status = 10 AND s.city = 'PARIS';
```

Data Returned:

	s [PK] character (20) ✎	sname character (20) ✎	status integer ✎	city character (20) ✎
1	S2	JONES	10	PARIS

### Question 6

SELECT \*

FROM date.s

WHERE s.status = 10 OR s.city = 'LONDON';

Data Returned:

	<b>s</b> [PK] character (20)	<b>sname</b> character (20)	<b>status</b> integer	<b>city</b> character (20)
1	S1	SMITH	20	LONDON
2	S2	JONES	10	PARIS
3	S4	CLARK	20	LONDON

### Question 7

SELECT \*

FROM date.spj

WHERE spj.qty >= 400 AND spj.s = 'S5' AND spj.p = 'P5';

Data Returned:

	<b>s</b> [PK] character (20)	<b>p</b> [PK] character (20)	<b>j</b> [PK] character (20)	<b>qty</b> integer
1	S5	P5	J5	500
2	S5	P5	J4	400





### Question 8

SELECT \*

FROM date.s

WHERE NOT(s.status = 10) AND s.city = 'PARIS';

Data Returned:

	<b>s</b> [PK] character (20) 	<b>sname</b> character (20) 	<b>status</b> integer 	<b>city</b> character (20) 
1	S3	BLAKE	30	PARIS





### Question 9

SELECT \*

FROM date.spj

WHERE spj.qty BETWEEN 100 AND 300 AND spj.p = 'P2';

Data Returned:

	<b>s</b> [PK] character (20) 	<b>p</b> [PK] character (20) 	<b>j</b> [PK] character (20) 	<b>qty</b> integer 
1	S5	P2	J2	200
2	S5	P2	J4	100





### Question 10

SELECT \*

FROM date.spj

WHERE spj.qty IN (100,300,500,700,900);

Data Returned:

	<b>s</b> [PK] character (20) 	<b>p</b> [PK] character (20) 	<b>j</b> [PK] character (20) 	<b>qty</b> integer 
1	S2	P3	J4	500
2	S2	P5	J2	100
3	S3	P4	J2	500
4	S4	P6	J3	300
5	S4	P6	J7	300
6	S5	P2	J4	100
7	S5	P5	J5	500
8	S5	P5	J7	100
9	S5	P1	J4	100
10	S5	P6	J4	500
11	S1	P1	J4	700

### Question 11

SELECT \*





FROM date.spj

WHERE spj.s IN (SELECT s.s

FROM date.s

WHERE s.status=10);

Data Returned:

	<b>s</b> [PK] character (20) 	<b>p</b> [PK] character (20) 	<b>j</b> [PK] character (20) 	<b>qty</b> integer 
1	S2	P3	J2	200
2	S2	P3	J3	200
3	S2	P3	J4	500
4	S2	P3	J5	600
5	S2	P3	J6	400
6	S2	P3	J7	800
7	S2	P5	J2	100
8	S2	P3	J1	400

### Question 12

SELECT s.sname, p.pname, spj.qty

FROM date.s, date.p, date.spj

WHERE spj.s = s.s AND spj.p = p.p

ORDER BY s.sname ASC, p.pname DESC;

Data Returned:

	sname character (20) 🔒	pname character (20) 🔒	qty integer 🔒
1	ADAMS	SCREW	200
2	ADAMS	SCREW	800
3	ADAMS	NUT	100
4	ADAMS	COG	200
5	ADAMS	COG	500
6	ADAMS	CAM	100
7	ADAMS	CAM	400
8	ADAMS	CAM	500
9	ADAMS	BOLT	200
10	ADAMS	BOLT	100
11	BLAKE	SCREW	500
12	BLAKE	SCREW	200
13	CLARK	COG	300
14	CLARK	COG	300
15	JONES	SCREW	400
16	JONES	SCREW	200
17	JONES	SCREW	500
18	JONES	SCREW	600
19	JONES	SCREW	400
20	JONES	SCREW	800
21	JONES	SCREW	200
22	JONES	CAM	100
23	SMITH	NUT	200



### Question 13

SELECT s.sname, p.pname, j.jname

FROM date.s, date.p, date.j

WHERE s.status = 10 AND p.colour = 'RED' AND j.city = 'PARIS';

Data Returned:

	sname character (20)	pname character (20)	jname character (20)
1	JONES	NUT	SORTER
2	JONES	SCREW	SORTER
3	JONES	COG	SORTER

### Question 14

SELECT spj.j,p.p, spj.qty\*p.weight AS "BILL"

FROM date.p, date.spj

WHERE spj.j = 'J4' AND spj.p = p.p

Data Returned:

	j character (20)	p character (20)	BILL integer
1	J4	P1	8400
2	J4	P1	1200
3	J4	P2	1700
4	J4	P3	3400
5	J4	P3	8500
6	J4	P4	11200
7	J4	P5	4800
8	J4	P6	9500

**Question 15 (Assuming the triplet appears as a group in the SPJ table)**

```
SELECT spj.j,spj.p,spj.s
```

```
FROM date.p, date.j, date.s, date.spj
```

```
WHERE p.p = spj.p AND j.j = spj.j AND s.s = spj.s AND j.city = p.city AND p.city = s.city
```

Data Returned:

	<b>j</b> [PK] character (20)	<b>p</b> [PK] character (20)	<b>s</b> [PK] character (20)
1	J7	P6	S4

**Question 16**

```
SELECT DISTINCT j.jname,p.colour
```

```
FROM date.p, date.j
```

```
CROSS JOIN date.spj
```

```
WHERE p.p = spj.p AND j.j = spj.j
```

```
ORDER BY j.jname ASC, p.colour ASC
```

Data Returned:

	<b>jname</b> character (20)	<b>colour</b> character (20)
1	CONSOLE	BLUE
2	CONSOLE	GREEN
3	CONSOLE	RED
4	DISPLAY	BLUE
5	DISPLAY	GREEN
6	DISPLAY	RED
7	EDS	BLUE
8	OCR	BLUE
9	OCR	RED
10	RAID	BLUE
11	SORTER	BLUE
12	SORTER	RED
13	TAPE	BLUE
14	TAPE	RED

### Question 17

```
SELECT j.j, j.jname, j.city
```

```
FROM date.j
```

```
WHERE j.city IS NULL;
```

Data Returned:

	j [PK] character (20)	jname character (20)	city character (20)

### Question 18

```
SELECT spj.j, spj.p, spj.s
```

```
FROM date.spj
```

```
WHERE spj.j IS NULL OR spj.s IS NULL OR spj.p IS NULL;
```

Data Returned:

	j [PK] character (20)	p [PK] character (20)	s [PK] character (20)

### Question 19

```
SELECT p.p, p.pname, p.colour, p.weight, p.city
```

```
FROM date.p
```

```
WHERE p.weight < 0;
```

Data Returned:

	p [PK] character (20)	pname character (20)	colour character (20)	weight integer	city character (20)

### Question 20

```
SELECT spj.p AS "Missing Product"
```

```
FROM date.spj
```

```
WHERE spj.p NOT IN (SELECT DISTINCT p.p FROM date.p);
```

Data Returned:

Missing Product character (20)

#### Task 4: Normalisation Exercises

1. Reduce the following data structure into a relation in **1NF**. (Note braces (i.e. { and }) enclose a set of values from a domain). Also create a **primary key set** for the resultant relation. The first row contains attribute names; you are free to change as required.

StudNum	StudDept	StudA_levels	StudI_Levels
1315	CIS	{Music, Computing}	{Maths, Philosophy}
1344	MATHS	{Maths, Physics}	{Computing}
1399	STATS	{ }	{ }

#### Answer

StudNum	StudDept	StudA_Levels	StudI_Levels
1315	CIS	Computing	Maths
1315	CIS	Computing	Philosophy
1315	CIS	Music	Maths
1315	CIS	Music	Philosophy
1344	MATHS	Maths	Computing
1344	MATHS	Physics	Computing
1399	STATS	{ }	{ }

2. Reduce the following 1NF relation into a 2NF compliant relation. (If need be do introduce a primary key set). Clearly state and confirm any FDs and how the decomposition progressed.

StudNum	StudName	UnitNum	UnitName	TransGrade
1315	Vella, James	CIS1041	Intro to DB	C+
1315	Vella, James	CIS1099	Prog in C	B+
1344	Desira, Peter	CIS1011	Soft Eng	A
1344	Desira, Peter	CIS1041	Intro to DB	B+
1344	Desira, Peter	CIS1099	Prog in C	A+
1399	Stivala, Belle	CIS1099	Prog in C	A+

### Answer

<u>StudNum</u>	StudName
1315	Vella, James
1344	Desira, Peter
1399	Stivala, Belle

<u>UnitNum</u>	UnitName
CIS1041	Intro to DB
CIS1099	Prog in C
CIS1011	Soft Eng

<u>StudNum</u>	<u>UnitNum</u>	TransGrade
1315	CIS1041	C+
1315	CIS1099	B+
1344	CIS1011	A
1344	CIS1041	B+
1344	CIS1099	A+
1399	CIS1099	A+

### Functional dependencies

StudNum->StudName (StudName is functionally dependent on StudNum)

StudNum,UnitNum->TransGrade (TransGrade is functionally dependent on StudNum & UnitNum)

UnitNum->UnitName (UnitName is functionally dependent on UnitNum)

3. Reduce the following 1NF relation into 3NF compliant relation. (If need be do introduce a primary key set). Clearly state and confirm any FDs and how the decomposition progressed.

StudNum	StudName	UnitNum	UnitName	TransGrade	GradeDesc
1315	Vella, James	CIS1041	Intro to DB	C+	Average (well almost)
1315	Vella, James	CIS1099	Prog in C	B+	Very Good (really)
1344	Desira, Peter	CIS1011	Soft Eng	A	Excellent (well almost)
1344	Desira, Peter	CIS1041	Intro to DB	B+	Very Good (really)
1344	Desira, Peter	CIS1099	Prog in C	A+	Excellent
1399	Stivala, Belle	CIS1099	Prog in C	A+	Excellent

### Answer

<u>StudNum</u>	StudName
1315	Vella, James
1344	Desira, Peter
1399	Stivala, Belle

<u>UnitNum</u>	UnitName
CIS1041	Intro to DB
CIS1099	Prog in C
CIS1011	Soft Eng

<u>StudNum</u>	<u>UnitNum</u>	TransGrade
1315	CIS1041	C+
1315	CIS1099	B+
1344	CIS1011	A
1344	CIS1041	B+
1344	CIS1099	A+
1399	CIS1099	A+

<u>TransGrade</u>	GradeDesc
A+	Excellent
A	Excellent (well almost)
B+	Very Good (really)
C+	Average (well almost)

### Functional dependencies

StudNum->StudName (StudName is functionally dependent on StudNum)

StudNum,UnitNum->TransGrade (TransGrade is functionally dependent on StudNum& UnitNum)

UnitNum->UnitName (UnitName is functionally dependent on UnitNum)

TransGrade->GradeDesc (GradeDesc is functionally dependent on TransGrade)

## Task 5: Procedural Language Tasks

- 1 Write stored functions that have the following structure (PLPGSQL slide 12 & 13):  
i) & ii) Takes no arguments and returns a single integer and a set of records;  
iii) & iv) Takes an in argument and returns a single integer and a set of records;

i.

```
CREATE OR REPLACE FUNCTION getRecordNo() RETURNS int AS $$
```

```
DECLARE
```

```
    CNT int;
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO CNT
```

```
    FROM scott.emp;
```

```
    RETURN CNT;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

Input: `SELECT getRecordNo();`

Output:

getrecordno integer	
1	14



ii.

```
CREATE OR REPLACE FUNCTION ShowEmployeeRecords()
```

```
RETURNS setof scott.emp
```

```
AS 'select * from scott.emp order by empno asc;'
```

```
LANGUAGE 'sql';
```

Input: SELECT ShowEmployeeRecords();

Output:

	<b>showemployeerecords</b> scott.emp
1	(7369,SMITH,CLERK,7902,1980-12-17,800,,20)
2	(7499,ALLEN,SALESMAN,7698,1981-02-20,1600,300,30)
3	(7521,WARD,SALESMAN,7698,1981-02-22,1250,500,30)
4	(7566,JONES,MANAGER,7839,1981-04-02,2975,,20)
5	(7654,MARTIN,SALESMAN,7698,1981-09-28,1250,1400,30)
6	(7698,BLAKE,MANAGER,7839,1981-05-01,2850,,30)
7	(7782,CLARK,MANAGER,7839,1981-06-09,2450,,10)
8	(7788,SCOTT,ANALYST,7566,1982-12-09,3000,,20)
9	(7839,KING,PRESIDENT,,1981-11-17,5000,,10)
10	(7844,TURNER,SALESMAN,7698,1981-09-08,1500,0,30)
11	(7876,ADAMS,CLERK,7788,1983-01-12,1100,,20)
12	(7900,JAMES,CLERK,7698,1981-12-03,950,,30)
13	(7902,FORD,ANALYST,7566,1981-12-03,3000,,20)
14	(7934,MILLER,CLERK,7782,1982-01-23,1300,,10)

iii.

```
CREATE OR REPLACE FUNCTION NoOfPplSameJob(in varchar(255), out ans int)
AS $$
SELECT COUNT(*) FROM scott.emp AS emp WHERE emp.job = $1;
$$ LANGUAGE SQL;
```

Input: SELECT NoOfPplSameJob('CLERK');

Output:

	noofpplsamejob integer	
1		4

iv.

```
CREATE OR REPLACE FUNCTION InfoOfPplSameJob(in varchar(255))
```


```
RETURNS setof scott.emp
```

```
AS 'select * from scott.emp as emp where emp.job = $1 order by emp.empno asc;'
```

```
LANGUAGE 'sql';
```

Input: SELECT InfoOfPplSameJob('CLERK');

Output:

	<b>infoofpplsamejob</b> scott.emp 
1	(7369,SMITH,CLERK,7902,1980-12-17,800,,20)
2	(7876,ADAMS,CLERK,7788,1983-01-12,1100,,20)
3	(7900,JAMES,CLERK,7698,1981-12-03,950,,30)
4	(7934,MILLER,CLERK,7782,1982-01-23,1300,,10)

- 2 Write a function that takes an argument and returns a value. Same function is then used in a where clause of a select statement (as in PLPGSQL slide 15).

```
CREATE OR REPLACE FUNCTION EmployeeDept(empId int) RETURNS int AS $$
```

```
DECLARE
```

```
    deptNo int;
```

```
BEGIN
```

```
    SELECT emp.deptno INTO deptNo
```

```
    FROM scott.emp AS emp
```

```
    WHERE emp.empno = empId;
```

```
    RETURN deptNo;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```


```
CREATE OR REPLACE FUNCTION EmpSameDept(empId int) RETURNS setof scott.emp AS $$
```

```
    SELECT * FROM scott.emp AS emp WHERE emp.deptno = EmployeeDept(empId);
```

```
$$ LANGUAGE SQL;
```

Input: `SELECT EmpSameDept(7369);`

Output:

	<b>empsamedept</b> scott.emp 
1	(7369,SMITH,CLERK,7902,1980-12-17,800,,20)
2	(7566,JONES,MANAGER,7839,1981-04-02,2975,,20)
3	(7788,SCOTT,ANALYST,7566,1982-12-09,3000,,20)
4	(7876,ADAMS,CLERK,7788,1983-01-12,1100,,20)
5	(7902,FORD,ANALYST,7566,1981-12-03,3000,,20)

**3 Write a function that avoids using an out argument (PLPGSQL slide 22).**

```
CREATE OR REPLACE FUNCTION InfoOfPplSameSal(in int)
RETURNS setof scott.emp
AS 'select * from scott.emp as emp where emp.sal = $1 order by emp.empno asc;'
LANGUAGE 'sql';
```

Input: SELECT InfoOfPplSameSal(1250);

Output:

	infofpplsamesal scott.emp
1	(7521,WARD,SALESMAN,7698,1981-02-22,1250,500,30)
2	(7654,MARTIN,SALESMAN,7698,1981-09-28,1250,1400,30)

4 Write a server function that prints 'hello world' on the client (e.g. pgAdmin/SQL/Message window – PLPGSQL slide 34).

```
CREATE OR REPLACE FUNCTION sayHello() RETURNS int AS $$  
BEGIN  
    RAISE NOTICE 'hello world';  
    RETURN 0;  
END;  
$$ LANGUAGE plpgsql;
```

Input: SELECT sayHello();

Output:

```
NOTICE:  hello world
```

```
Successfully run. Total query runtime: 101 msec.  
1 rows affected.
```

5 Write a function that takes a job type and prints, on the client, the number of employees' having that job. Ensure adequate error control is coded.

```
CREATE OR REPLACE FUNCTION NOoOfEmployeeJobType(checkJob varchar(255)) RETURNS  
int AS $$
```

```
DECLARE
```

```
    CNT int;
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO CNT
```

```
    FROM scott.emp AS emp
```

```
    WHERE emp.job = checkJob;
```

```
    RAISE NOTICE 'There are % employees with the same job', CNT;
```

```
    RETURN CNT;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

Input: `SELECT NOoOfEmployeeJobType('CLERK');`

Output:

nooofemployeejobtype integer	
1	4

```
NOTICE:  There are 4 employees with the same job
```

```
Successfully run. Total query runtime: 107 msec.
```

```
1 rows affected.
```

6 Reproduce the getFactorial function discussed in class (PLPGSQL slide 38).

```
CREATE OR REPLACE FUNCTION getFactorial( fn int) RETURNS int as $$  
DECLARE  
    product int;  
    minus1 int;  
BEGIN  
    if (fn > 12) then  
        RAISE EXCEPTION 'Error: getFactorial - argument too large %!',fn;  
    end if;  
    minus1 := fn - 1;  
    if (minus1>0)  
    then  
        product := fn * getFactorial(minus1);  
        return product;  
    end if;  
    return fn;  
END;  
$$ language plpgsql;
```

Input: SELECT getFactorial(7);

Output:

getfactorial	
integer	
1	5040



```
CREATE OR REPLACE FUNCTION EMPJOBS_DEPT(DNUMBER INTEGER)
RETURNS TABLE (JOBDESCRIPTION TEXT) AS
$$ SELECT DISTINCT emp.job FROM scott.emp AS emp WHERE emp.deptno = $1;
$$ LANGUAGE SQL;
```

Input: SELECT EMPJOBS\_DEPT(20);

Output:

	empjobs_dept text
1	ANALYST
2	CLERK
3	MANAGER

## 8      Reproduce the parameterised views (relative to postgresql) (SQL slide 103).

```
CREATE OR REPLACE FUNCTION CreateView(dptNo int) RETURNS void AS
$BODY$
BEGIN
    EXECUTE 'CREATE OR REPLACE VIEW scott.newView AS SELECT * FROM scott.emp AS emp
WHERE emp.deptno =' || $1;

    RETURN;
END;
$BODY$ LANGUAGE plpgsql STRICT;
```

Input: SELECT CreateView(20);

Output:

	empno integer	ename character varying (10)	job character varying (9)	mgr integer	hiredate date	sal numeric	comm numeric	deptno integer
1	7369	SMITH	CLERK	7902	1980-12-17	800	[null]	20
2	7566	JONES	MANAGER	7839	1981-04-02	2975	[null]	20
3	7788	SCOTT	ANALYST	7566	1982-12-09	3000	[null]	20
4	7876	ADAMS	CLERK	7788	1983-01-12	1100	[null]	20
5	7902	FORD	ANALYST	7566	1981-12-03	3000	[null]	20