You are required to design a database for a book publishing company. The database needs to store a table of authors and books. An author has many books.

Table definitions are as follows:

|  |  |
| --- | --- |
| authors | id SERIAL, name varchar(50) |
| books | Id SERIAL, title varchar(100), author\_id |
| Reviews | Id SERIAL, rating INTEGER, reviewer\_id INTEGER, book\_is  Note: reviewer\_id is nothing but authors id |

Write a query that will join together these two tables.

1. For each book, print the title of the book and name of the author.

SELECT title, name

FROM books

JOIN authors on books.author\_id=authors.id;

1. Return title of each book, along with the name of the author. All authors should be included, even if they don’t have a book associated with them.

**(You need to provide two possible solutions using Joins)**

SELECT title, name

FROM books

RIGHT JOIN authors on books.author\_id=authors.id;

SELECT title, name

FROM authors

LEFT JOIN books on authors.id=books.author\_id;

1. Return title of each book, along with the name of the author, and the rating of a review. Only show rows where the author of the book is also the author of the review.

SELECT title, name, rating

FROM reviews

JOIN books ON reviews.book\_id=books.id

JOIN authors ON books.author\_id=authors.id

WHERE author\_id=reviewer\_id;

1. Prints an author’s id and the number of books they have authored.

**(Check do you need two tables for this query).**

SELECT author\_id, Count(\*) AS numberOfBooks

FROM books

GROUP BY author\_id;

1. Print an author’s name and the number of books they have authored.

SELECT authors.id, count(\*)

FROM authors

JOIN books on authors.id=books.author\_id

GROUP BY authors.id;

Table details are as follows:

|  |  |
| --- | --- |
| Phones | name Varchar(50), manufacturer varchar(100), price INTEGER, units\_sold INTEGER |

1. Write a query that prints the name of manufacturers and total revenue (price \* units\_sold) for all phones. Only print the manufacturers who have revenue greater than 2,000,00 for all phones they sold.

SELECT manufacturer, total\_revenue

FROM (SELECT SUM(price\*units\_sold) AS total\_revenue, manufacturer

FROM phones

GROUP BY manufacturer

HAVING SUM(price\*units\_sold) > 100000

) AS FIND\_REV

1. Write a query that shows the names of only the second and third most expensive phones.

SELECT name FROM phones

ORDER BY price DESC

LIMIT 2

1. Write a query that will print the manufacturer of phones where the phone’s price is less than 170. Also print all manufacturer that have created more than two phones.

SELECT manufacturer, numPhonesLessThan170

FROM (SELECT manufacturer, count(\*) as numPhonesLessThan170

FROM (SELECT \* FROM phones

WHERE price<170) AS price\_find

GROUP BY manufacturer) AS count\_find

1. Write a query that prints the name and price for each phone. In addition, print out the ratio of the phones price against max of all prices. Rename this third column to price\_ratio.

SELECT name, price,

(CAST(price AS double precision) / (SELECT max(price) FROM phones)) AS price\_ratio

FROM phones

1. Write a query that will print the name and price of all phones that sold greater than 5000 units.

SELECT name, price

FROM phones

WHERE units\_sold>5000

1. Write a query that will select the name and manufacturer for all phones created by Apple or Samsung.

SELECT name, price, manufacturer

FROM phones

WHERE manufacturer = 'Apple' OR manufacturer = 'Samsung'

manufacturer

1. Write a query that will print the name and total\_revenue of all phones with a total\_revenue greater than 1,000,00.

SELECT name, total\_revenue

FROM (SELECT SUM(price\*units\_sold) AS total\_revenue, name

FROM phones

GROUP BY name

HAVING SUM(price\*units\_sold)>100000) AS find\_revenue

Create a new dataset from the shared product-orders dataset sheet.

Write the following queries:

1. Print the number of paid and unpaid orders.

SELECT paid, count(\*) FROM orders

GROUP BY paid;

1. Print the first\_name and last\_name of each user along with the whether they have paid for their order.

SELECT first\_name, last\_name, paid FROM users

JOIN orders on users.id = orders.user\_id;

Udemy Questions

126

CREATE OR REPLACE FUNCTION set\_employee\_defualt\_photo() RETURNS void AS $$

UPDATE employees

SET photopath='default.bmp'

WHERE photopath IS NULL;

$$ LANGUAGE SQL;

127

CREATE OR REPLACE FUNCTION biggest\_order() RETURNS double precision AS $$

SELECT max(largest\_order) FROM

(

SELECT SUM(unitprice\*quantity) as largest\_order, orderid

FROM order\_details

GROUP BY orderid

)as order\_price

$$ LANGUAGE SQL

CREATE OR REPLACE FUNCTION most\_ordered\_products(cust\_id bpchar) RETURNS varchar(40) AS $$

SELECT productname FROM products

WHERE productid IN

( SELECT productid

FROM order\_details

JOIN orders USING (orderid)

WHERE customerid=cust\_id

GROUP BY productid

ORDER BY SUM(quantity) DESC

LIMIT 1)

$$ LANGUAGE SQL

128

129

CREATE OR REPLACE FUNCTION full\_name(employees) RETURNS varchar(62) as $$

SELECT $1.title || '' || $1.firstname || '' || $1.lastname;

$$ LANGUAGE SQL;

130

CREATE OR REPLACE FUNCTION highest\_inventory() RETURNS products AS $$

SELECT \*

FROM products

GROUP BY productid

ORDER BY SUM(unitprice\*unitsinstock) DESC

LIMIT 1

$$ LANGUAGE SQL;

131

CREATE OR REPLACE FUNCTION sum\_n\_product (x int, OUT squared int, OUT cubed int) AS $$

SELECT x\*x, x\*x\*x

$$ LANGUAGE SQL;

132

CREATE OR REPLACE FUNCTION sum\_n\_product (x int DEFAULT 10, OUT squared int, OUT cubed int) AS $$

SELECT x\*x, x\*x\*x

$$ LANGUAGE SQL;

133

SELECT productname, companyname

FROM highest\_inventory() as test

JOIN suppliers ON test.supplierid = suppliers.supplierid

134

CREATE OR REPLACE FUNCTION suppliers\_to\_reorder\_from() RETURNS SETOF suppliers AS $$

SELECT \* FROM suppliers

WHERE supplierid in(

SELECT supplierid FROM products

WHERE unitsinstock+unitsonorder>reorderlevel

)

$$ LANGUAGE SQL;

CREATE OR REPLACE FUNCTION excess\_inv(num int) RETURNS

TABLE(excess int, productid int, productname varchar(40)) AS $$

SELECT SUM((unitsinstock+unitsonorder)-(reorderlevel\*num/100))::int AS excess,

productid, productname FROM products

WHERE (unitsinstock+unitsonorder)-(reorderlevel\*num/100)>0

GROUP BY productid

$$ LANGUAGE SQL;

135

CREATE OR REPLACE PROCEDURE change\_supplier\_prices(sup\_id int) AS $$

UPDATE products

SET unitprice = unitprice+0.5

WHERE supplierid=sup\_id

$$ LANGUAGE SQL;

147

CREATE FUNCTION biggest\_order() RETURNS double precision AS $$

BEGIN

SELECT max(largest\_order) FROM

(

SELECT SUM(unitprice\*quantity) as largest\_order, orderid

FROM order\_details

GROUP BY orderid

)as order\_price;

END;

$$ LANGUAGE plpgsql;

148

CREATE FUNCTION square\_n\_cubed(in x int, OUT squared int, OUT cubed int) AS $$

BEGIN

squared:=x\*x;

cubed:=x\*x\*x;

END;

$$ LANGUAGE plpgsql;

149

CREATE FUNCTION suppliers\_to\_reorder() RETURNS SETOF suppliers AS $$

BEGIN

RETURN QUERY SELECT \* FROM suppliers

WHERE supplierid in(

SELECT supplierid FROM products

WHERE unitsinstock+unitsonorder>reorderlevel

);

END;

$$ LANGUAGE plpgsql;

150

CREATE OR REPLACE FUNCTION find\_price() RETURNS SETOF orders AS $$

DECLARE

avg\_price real;

BEGIN

SELECT avg(price\_avg) INTO avg\_price

FROM (SELECT SUM(unitprice\*quantity)AS price\_avg, orderid

FROM order\_details

GROUP BY orderid) as find\_prices;

RETURN QUERY SELECT \* FROM orders

WHERE orderid IN (

SELECT orderid FROM (

SELECT SUM(unitprice\*quantity)AS price\_avg, orderid

FROM order\_details

GROUP BY orderid

HAVING SUM(unitprice\*quantity) BETWEEN avg\_price\*0.75 AND avg\_price\*1.25

) AS find\_prices2

);

END;

$$ LANGUAGE plpgsql;

151

CREATE OR REPLACE FUNCTION avg\_square() RETURNS double precision as $$

DECLARE

square\_sum double precision :=0;

cnt int :=0;

product record;

avg\_sum double precision :=0;

BEGIN

FOR product IN SELECT \* FROM products LOOP

square\_sum:= square\_sum+SUM(product.unitprice\*product.unitprice);

cnt := cnt+1;

END LOOP;

avg\_sum:=square\_sum/cnt;

RETURN avg\_sum;

END;

$$ LANGUAGE plpgsql;

152

CREATE OR REPLACE FUNCTION time\_of\_year(day\_in TIMESTAMP) RETURNS text AS $$

BEGIN

IF EXTRACT(MONTH FROM day\_in) >=3 AND EXTRACT(MONTH FROM day\_in) <=5 THEN

RETURN 'Spring';

ELSIF EXTRACT(MONTH FROM day\_in) >5 AND EXTRACT(MONTH FROM day\_in) <=8 THEN

RETURN 'Summer';

ELSIF EXTRACT(MONTH FROM day\_in) >8 AND EXTRACT(MONTH FROM day\_in) <=11 THEN

RETURN 'Autumn';

ELSE RETURN 'Winter';

END IF;

END;

$$ LANGUAGE plpgsql;

153 – No Question

CREATE OR REPLACE FUNCTION factorial(factorial int) RETURNS int AS $$

DECLARE

fact\_sum bigint:=1;

cnt int:=factorial;

BEGIN

WHILE cnt>=1 LOOP

fact\_sum:= fact\_sum\*cnt;

cnt:=cnt-1;

END LOOP;

RETURN fact\_sum;

END;

$$ LANGUAGE plpgsql;

154

155

CREATE OR REPLACE FUNCTION first\_multiple(num\_array int[], divisor int) RETURNS int AS $$

DECLARE

element\_in\_array int;

first\_divisor int;

BEGIN

FOREACH element\_in\_array IN ARRAY num\_array LOOP

IF (element\_in\_array%divisor=0) THEN

RETURN element\_in\_array;

END IF;

END LOOP;

RETURN 0;

END;

$$ LANGUAGE plpgsql