

3 - GraphQL

1. Learning Outcomes

On completion of this lab you will have:

- Implemented a GraphQL interface to a PostgreSQL schema

2. Organisation

Please complete the exercises individually.

3. Grading

This worksheet is worth up to 10% of your overall module grade.

Note: You must attend and sign in at 10 labs in order to obtain full credit for your submitted worksheets. You may work on this worksheet during labs 3 and 4 with instructor assistance.

4. Submission

The deadline for submission is Sunday Mar 11, 2018 @23:59 through Webcourses.

5. Demonstration

You will demonstrate your solution to the lab instructor during the lab 5 session.

6. Requirements

For this lab you will need to

- Review the related module lecture material on Webcourses (lecture 19) and understood the theoretical concepts

7. Resources

You are free to research whatever you need to solve the problems in this lab. Some recommended resources include:

- <http://graphql.org/learn/>
- <https://www.graphile.org/postgraphile/>
- <https://github.com/graphile/postgraphile>

8. Set up

The following platform-independent set-up steps can be solved on Windows, Mac local Linux or Cloud Linux as you prefer

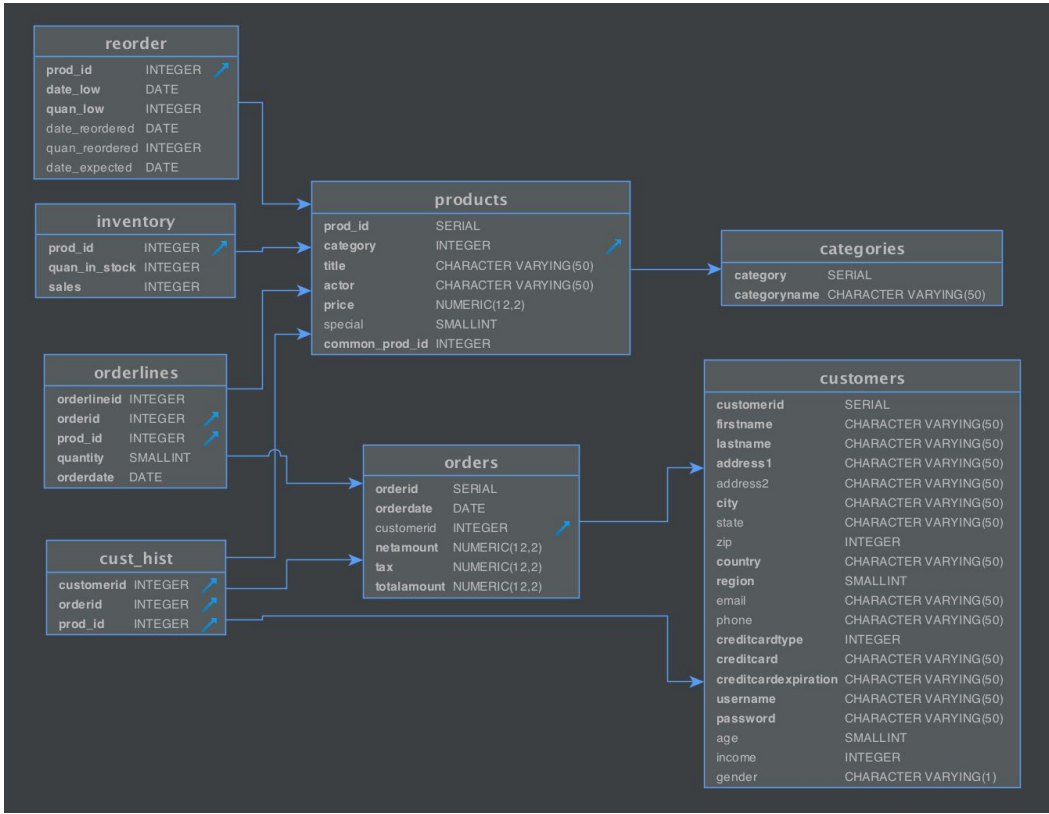
In the following steps, the database and schema **must be** named as instructed

1	Create a database called dellstore2	
2	Create a schema in your database called postgraphile	
3	Load the schema definitions and data from the dellstore2.sql file you downloaded from Webcourses	
4	Install postgraphile as per the instructions here: https://github.com/graphile/postgraphile	
5	Start the postgraphile server in watch-mode on your newly-created database/schema from the earlier steps	
6	Connect over a browser to http://localhost:5000/graphql	

9. Problem Sets

When submitting your GraphQL queries below, use a screen capture to show the query (or mutation) syntax and (at least part of) the output from your query

In the following problems, it is up to you to design and specify the GraphQL queries from the schema you have been provided. When doing so, try to think of a suitable real-world application for each query (e.g. a web/mobile client UI page, etc)

<p>1</p>	<p>Make your loaded schema exactly match the ERD below</p> <p>Hint: You will need to add several more foreign-key constraints which are missing from the <code>dellstore2.sql</code> file</p>  <pre> graph LR reorder --> products inventory --> products orderlines --> products orderlines --> orders cust_hist --> orders cust_hist --> products products --> categories orders --> customers </pre> <p>The ERD shows the following tables and their attributes:</p> <ul style="list-style-type: none"> reorder: prod_id (INTEGER), date_low (DATE), quan_low (INTEGER), date_reordered (DATE), quan_reordered (INTEGER), date_expected (DATE) inventory: prod_id (INTEGER), quan_in_stock (INTEGER), sales (INTEGER) orderlines: orderlineid (INTEGER), orderid (INTEGER), prod_id (INTEGER), quantity (SMALLINT), orderdate (DATE) cust_hist: customerid (INTEGER), orderid (INTEGER), prod_id (INTEGER) products: prod_id (SERIAL), category (INTEGER), title (CHARACTER VARYING(50)), actor (CHARACTER VARYING(50)), price (NUMERIC(12,2)), special (SMALLINT), common_prod_id (INTEGER) categories: category (SERIAL), categoryname (CHARACTER VARYING(50)) orders: orderid (SERIAL), orderdate (DATE), customerid (INTEGER), netamount (NUMERIC(12,2)), tax (NUMERIC(12,2)), totalamount (NUMERIC(12,2)) customers: customerid (SERIAL), firstname (CHARACTER VARYING(50)), lastname (CHARACTER VARYING(50)), address1 (CHARACTER VARYING(50)), address2 (CHARACTER VARYING(50)), city (CHARACTER VARYING(50)), state (CHARACTER VARYING(50)), zip (INTEGER), country (CHARACTER VARYING(50)), region (SMALLINT), email (CHARACTER VARYING(50)), phone (CHARACTER VARYING(50)), creditcardtype (INTEGER), creditcard (CHARACTER VARYING(50)), creditcardexpiration (CHARACTER VARYING(50)), username (CHARACTER VARYING(50)), password (CHARACTER VARYING(50)), age (SMALLINT), income (INTEGER), gender (CHARACTER VARYING(1)) 	<p>15 Marks</p>
<p>2</p>	<p>Build a GraphQL query which returns the attributes from a single database</p>	<p>15</p>

	relation. Have your query include one computed-field (which you can implement and a user-defined function in Postgres)	Marks
3	<p>Build a GraphQL query which returns the attributes from 3 joined database relations having 2 levels of nesting in the resultant output</p> <p>Describe an application of the query you have chosen to write</p>	20 Marks
4	Create a mutation to add a new order to the database. Your mutation should update the orders, orderlines and cust_hist relations	25 Marks
5	<p>In the previous problems you used postgraphql which conveniently introspects the database schema and dynamically builds the query and mutation objects and exports these over the graphql API</p> <p>In this problem, you should manually implement the query from problem 2 (above) directly using GraphQL and Express (or another server of your choice), i.e. not using postgraphql. For this you will probably need MassiveJS, Sequelize or similar to query Postgres as part of your resolver function</p> <p>See http://graphql.org/learn/ for more details</p>	25 Marks