Πανεπιστήμιο Θεσσαλίας
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πρόγραμμα Μεταπτυχιακών Σπουδών

Προχωρημένα Θέματα Βάσεων Δεδομένων 2015-16

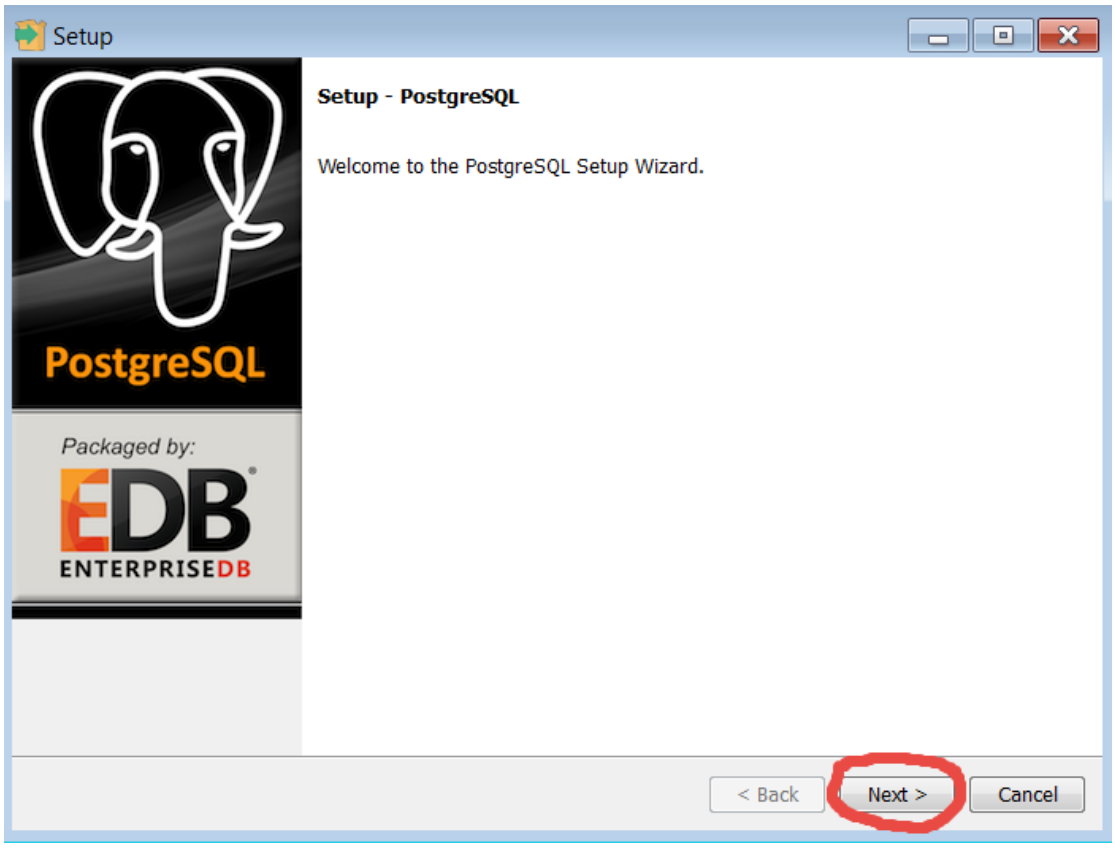
Online Shop Database

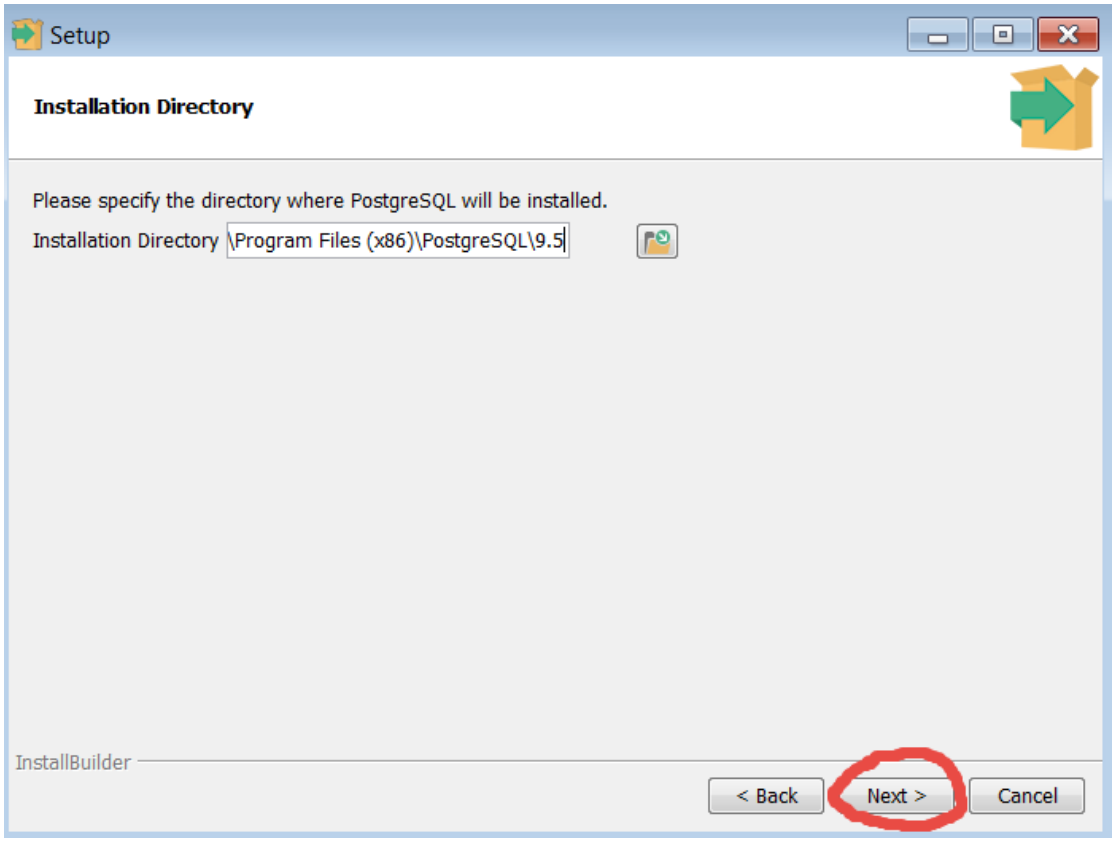
Θεοδώρου Γεώργιος

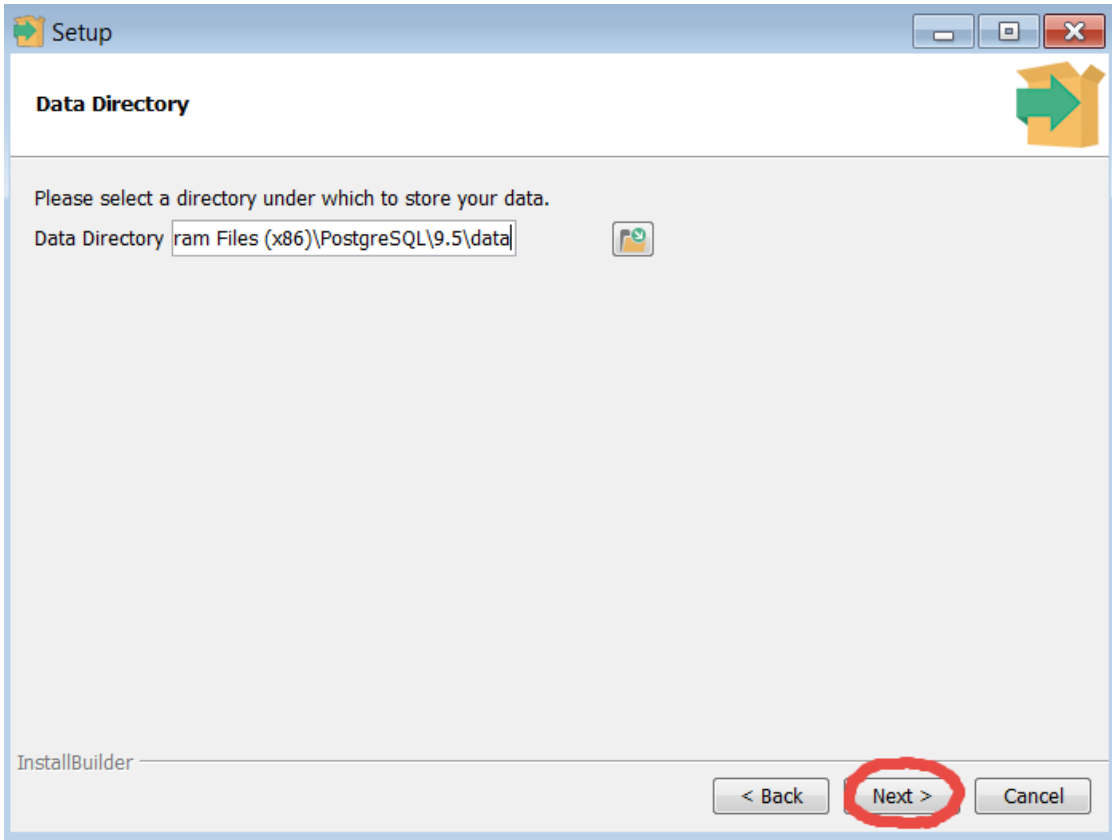
Περιεχόμενα

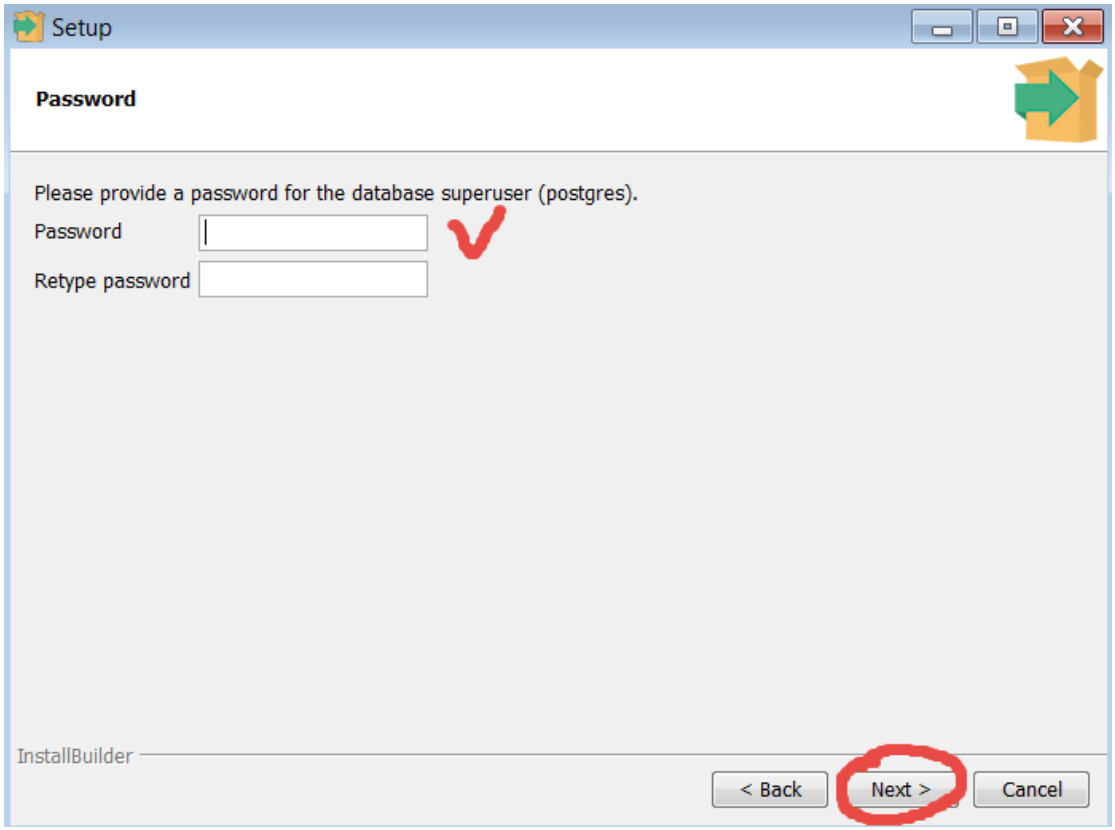
Πώς να εγκαταστήσουμε την PostgreSQL (σε Windows).

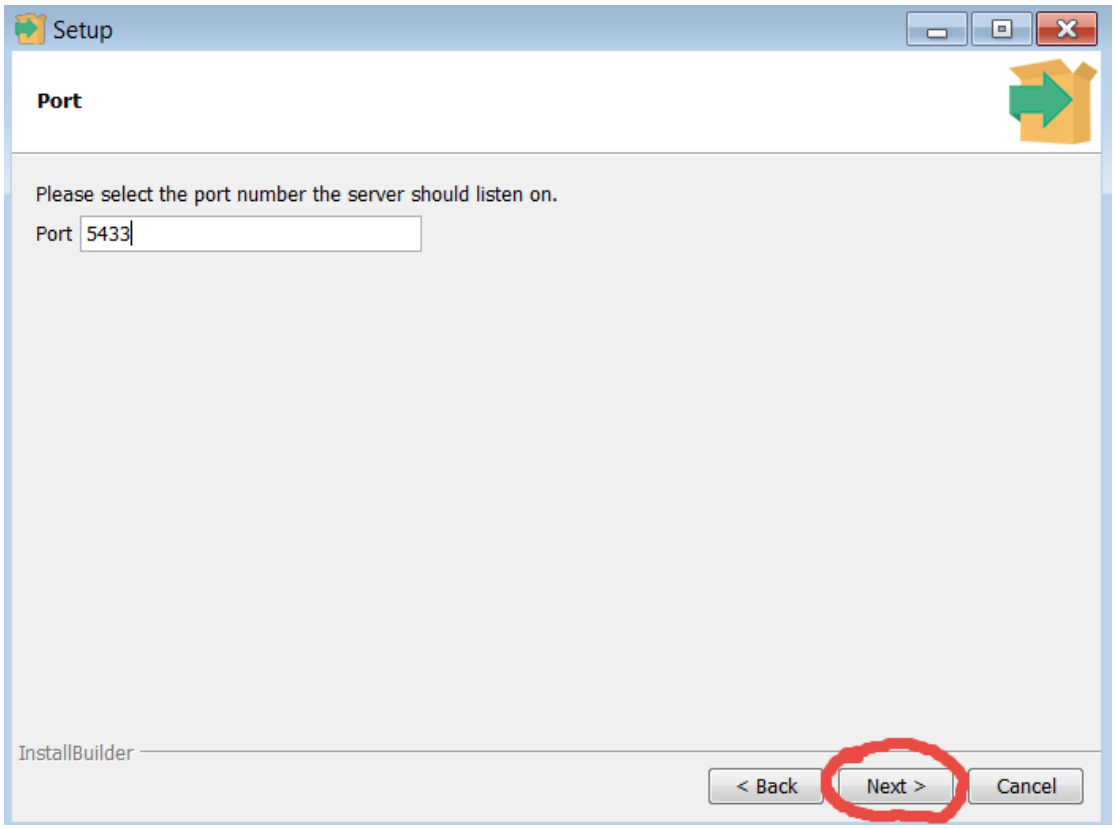
- Κατεβάζουμε την τελευταία έκδοση της PostgreSQL, ακολουθώντας το σύνδεσμο http://www.enterprisedb.com/products-services-training/pgdownload#windows.
- Έπειτα ακολουθούμε τις παρακάτω εικόνες:

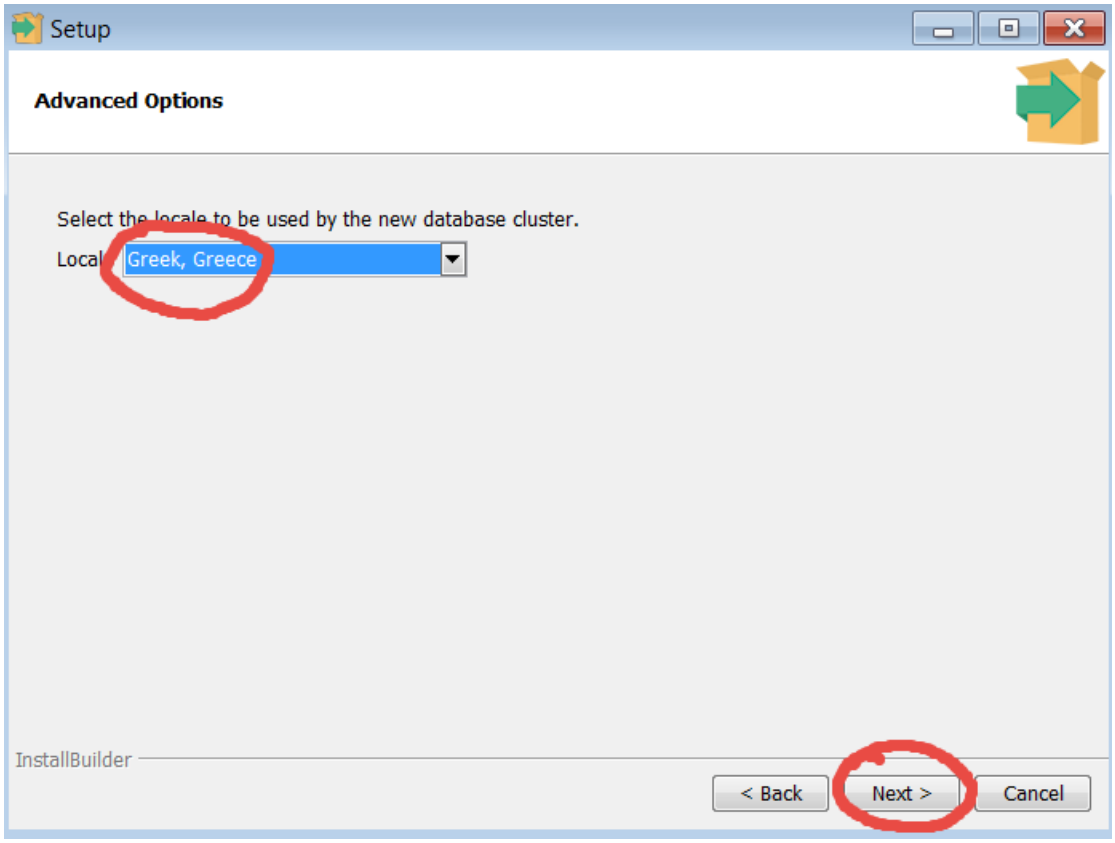


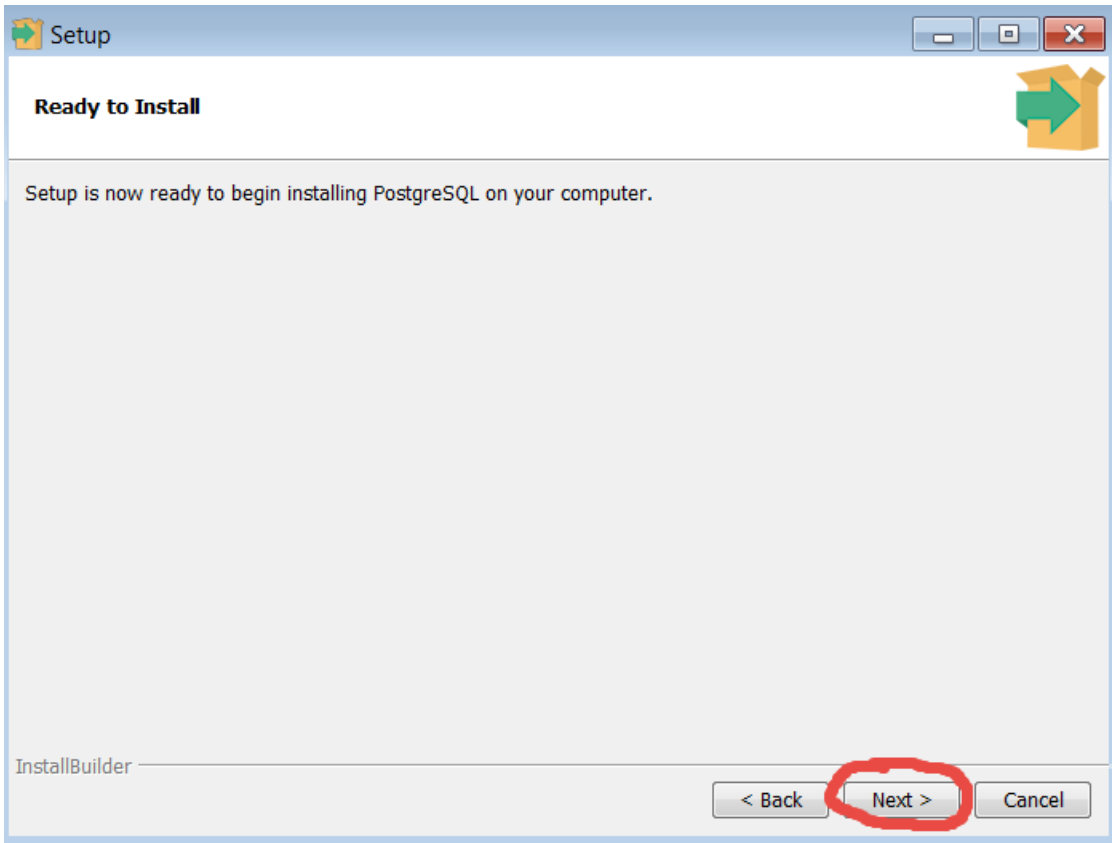

Setup
Installation Directory
Please specify the directory where PostgreSQL will be installed.
Installation Directory \Program Files (x86)\PostgreSQL\9.5
InstallBuilder
< Back
Next >
Cancel

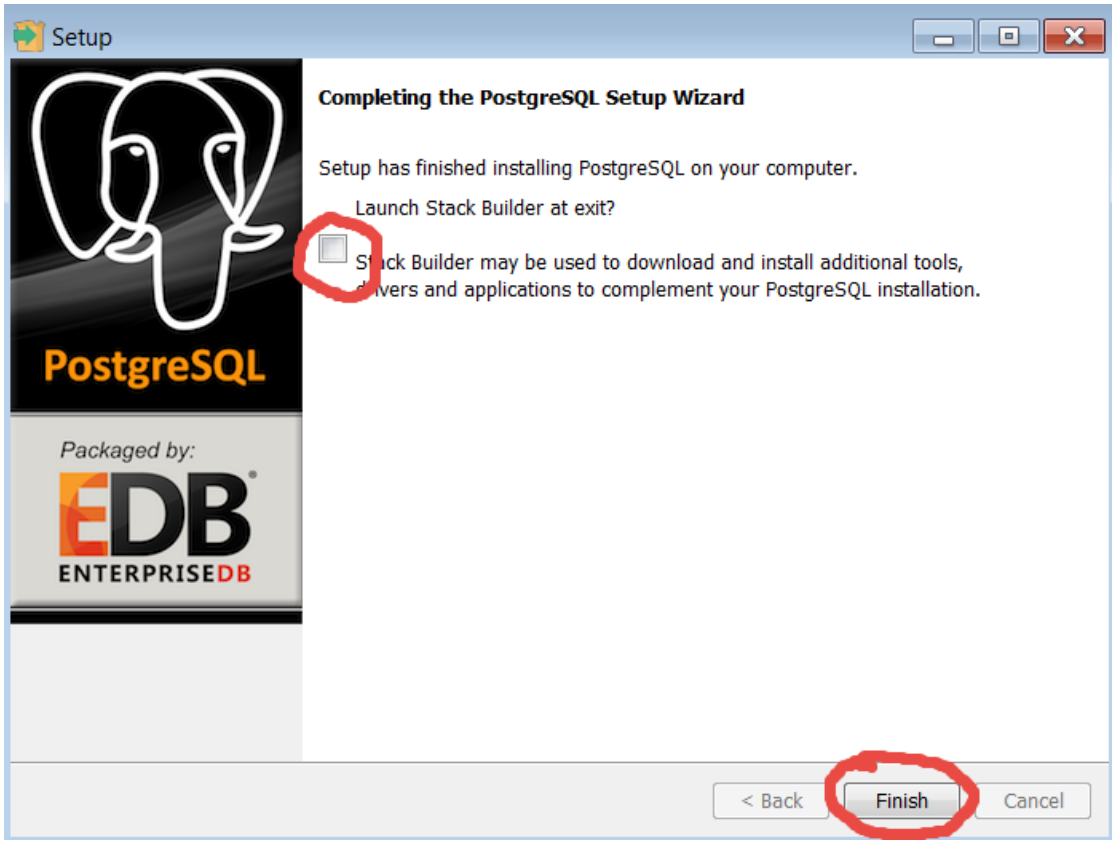

Setup
Data Directory
Please select a directory under which to store your data.
Data Directory ram Files (x86)\PostgreSQL\9.5\data
InstallBuilder
< Back
Next >
Cancel


Setup
Password
Please provide a password for the database superuser (postgres).
Password
Retype password
InstallBuilder
< Back
Next >
Cancel


Setup
Port
Please select the port number the server should listen on.
Port
5433
InstallBuilder
< Back
Next >
Cancel


Setup
Advanced Options
Select the locale to be used by the new database cluster.
Local
Greek, Greece
InstallBuilder
< Back
Next >
Cancel


Setup
Ready to Install
Setup is now ready to begin installing PostgreSQL on your computer.
InstallBuilder
< Back
Next >
Cancel


Setup
PostgreSQL
Packaged by:
EDB
ENTERPRISEDB
Completing the PostgreSQL Setup Wizard
Setup has finished installing PostgreSQL on your computer.
Launch Stack Builder at exit?
Stack Builder may be used to download and install additional tools,
drivers and applications to complement your PostgreSQL installation.
< Back
Finish
Cancel

Περιγραφή της βάσης δεδομένων.

Η βάση δεδομένων με την οποία θα ασχοληθούμε σε αυτό το άρθρο αφορά ένα διαδικτυακό κατάστημα. Η επιχείρηση θα έχει την δυνατότητα να αποθηκεύει και να αναζητά όλους τους πελάτες της και όλες τις παραγγελίες που έχουν κάνει κατά καιρούς. Επιπλέον θα είναι σε θέση πλέον να ελέγχει εάν υπάρχουν τα αντικείμενα που έχει παραγγείλει κάποιος πελάτης και μάλιστα και σε ποια υποκαταστήματα βρίσκονται. Η βάση μας έχει τις παρακάτω οντότητες:

Πελάτες, Παραγγελίες, Προϊόντα και Stock.

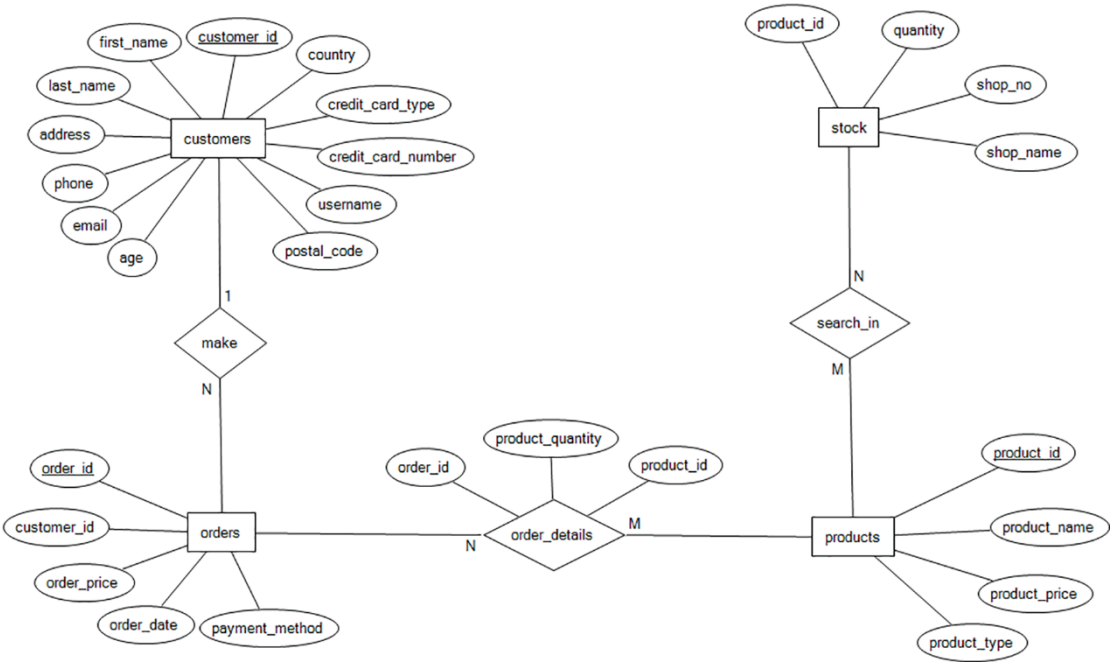
Οι συσχετίσεις είναι αρκετά προφανείς, κάθε πελάτης μπορεί να κάνει όσες παραγγελίες επιθυμεί, κάθε παραγγελία περιέχει κανένα, ένα ή και περισσότερα προϊόντα, και τέλος κάθε προϊόν μπορεί να βρίσκεται ή όχι σε κάποιο υποκατάστημα.

Σε κάθε οντότητα αντιστοιχεί και ένας πίνακας στη βάση δεδομένων μας. Θα πρέπει όμως να εισάγουμε έναν ακόμη πίνακα, τον Πληροφορίες_Παραγγελιών, διότι οι Παραγγελίες με τα Προϊόντα έχουν μία εξάρτυση «πολλά σε πολλά».

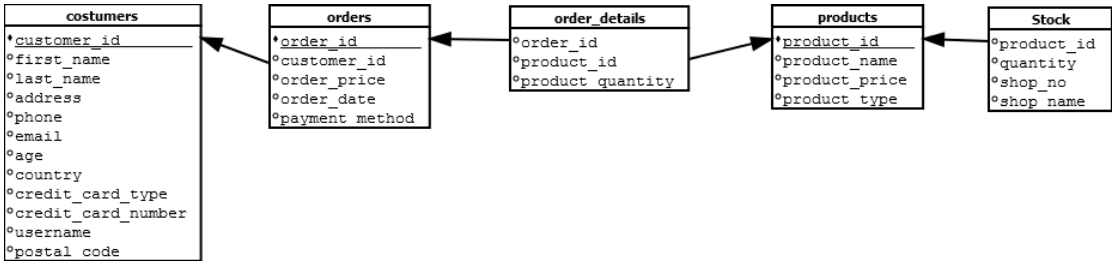
Τα πεδία των οντοτήτων απεικονίζονται λεπτομερώς στο διάγραμμα Οντοτήτων Συσχετίσεων που ακολουθεί.

Ως κύρια κλειδιά έχουμε ορίσει τους κωδικούς των πελατών, των παραγγελιών και των προϊόντων, στους πίνακες Πελάτες, Παραγγελίες και Προϊόντα αντίστοιχα. Ενώ ως ξένα κλειδιά στον πίνακα Παραγγελίες τον κωδικό των πελατών, στον πίνακα Πληροφορίες_Παραγγελιών τον κωδικό των παραγγελιών και των προϊόντων και στον πίνακα Υποκαταστήματα τον κωδικό των προϊόντων.

Διάγραμμα Οντοτήτων-Συσχετίσεων.



Σχεσιακό Σχήμα.



Κώδικας SQL για την κατασκευή της βάσης μας.

Ο ίδιος κώδικας υπάρχει και στο συνημμένο αρχείο με όνομα SQL Code, και μπορούμε να κάνουμε κατευθείαν import το αρχείο στην βάση μας.

Για να το κάνουμε import στο pgAdmin III ανοίγουμε το SQL Tool, πατάμε Open File, επιλέγουμε το αρχείο που επιθυμούμε και πατάμε Execute Query.

Εάν επιθυμούμε να κάνουμε import κατευθείαν την βάση μας, τότε κάνουμε create μία καινούργια βάση και κάνοντας δεξί click πάνω σε αυτή επιλέγουμε Restore και έπειτα το συνημμένο backup αρχείο.

- ```
  CREATE DATABASE "George"
    WITH OWNER = postgres
         ENCODING = 'UTF8'
         TABLESPACE = pg_default
         LC_COLLATE = 'Greek_Greece.1253'
         LC_CTYPE = 'Greek_Greece.1253'
         CONNECTION LIMIT = -1;
  ```

- ```
  CREATE TABLE public.customers
  (
    customer_id integer NOT NULL,
    first_name character varying(20),
    last_name character varying(40),
    address character varying(100),
    phone character varying(30),
    email character varying(50),
    country character varying(40),
    credit_card_type character varying(50),
    credit_card_number character varying(30),
    username character varying(20),
    postal_code character varying(20),
    age integer,
    CONSTRAINT "Customers_pkey" PRIMARY KEY (customer_id)
  )
  WITH (
    OIDS=FALSE
  );
  ALTER TABLE public.customers
    OWNER TO postgres;
  ```

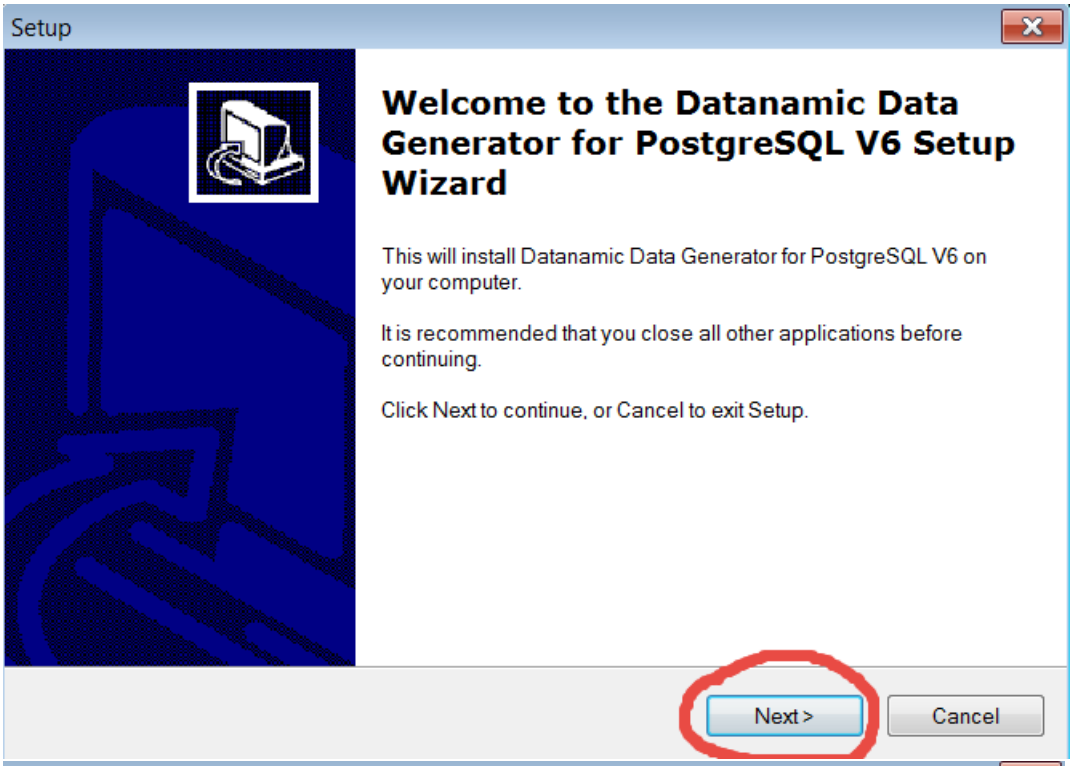
- ```
  CREATE TABLE public.orders
  (
    order_id integer NOT NULL,
    customer_id integer,
    order_price integer,
    order_date timestamp without time zone,
    payment_method character varying(30),
    CONSTRAINT orders_pkey PRIMARY KEY (order_id),
    CONSTRAINT orders_customer_id_fkey FOREIGN KEY (customer_id)
        REFERENCES public.customers (customer_id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION
  )
  WITH (
    OIDS=FALSE
  );
  ALTER TABLE public.orders
    OWNER TO postgres;
  ```

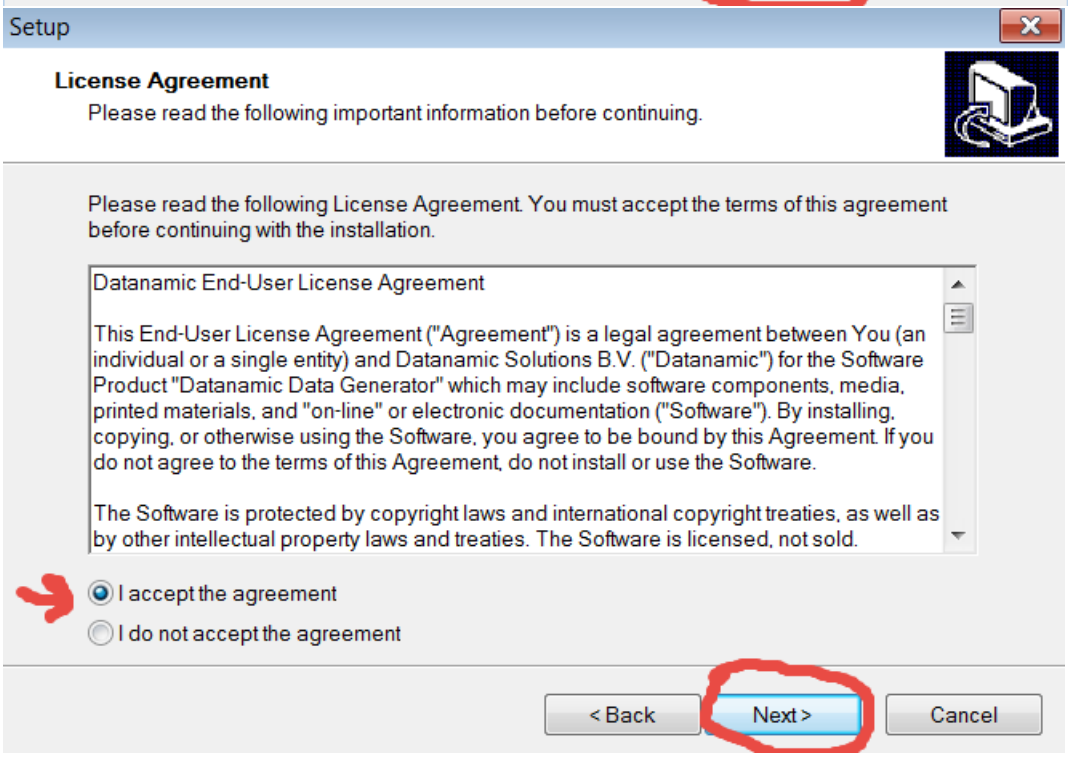
- ```
  CREATE TABLE public.order_details
  (
    product_id integer,
    order_id integer,
    product_quantity integer,
    CONSTRAINT "Order_Details_Product_ID_fkey" FOREIGN KEY
  (product_id)
        REFERENCES public.products (product_id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION
  )
  WITH (
    OIDS=FALSE
  );
  ALTER TABLE public.order_details
    OWNER TO postgres;
  ```

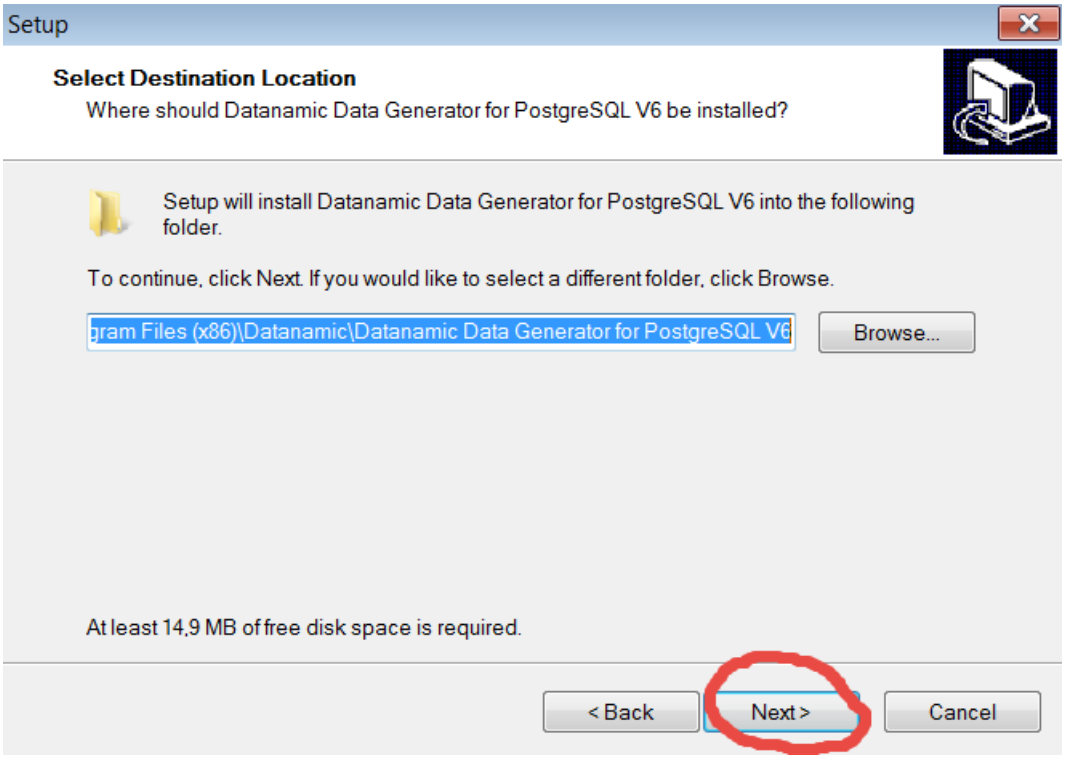
- ```
  CREATE TABLE public.products
  (
    product_id integer NOT NULL,
    product_name character varying(100),
    product_type character varying(50),
    product_price integer,
    CONSTRAINT "Products_pkey" PRIMARY KEY (product_id)
  )
  WITH (
    OIDS=FALSE
  );
  ALTER TABLE public.products
    OWNER TO postgres;
  ```
  ```

- ```
  CREATE TABLE public.stock
  (
    product_id integer,
    quantity integer,
    shop_no integer,
    shop_name character varying(50),
    CONSTRAINT "Stock_Product_ID_fkey" FOREIGN KEY (product_id)
        REFERENCES public.products (product_id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION
  )
  WITH (
    OIDS=FALSE
  );
  ALTER TABLE public.stock
    OWNER TO postgres;
  ```
  ```

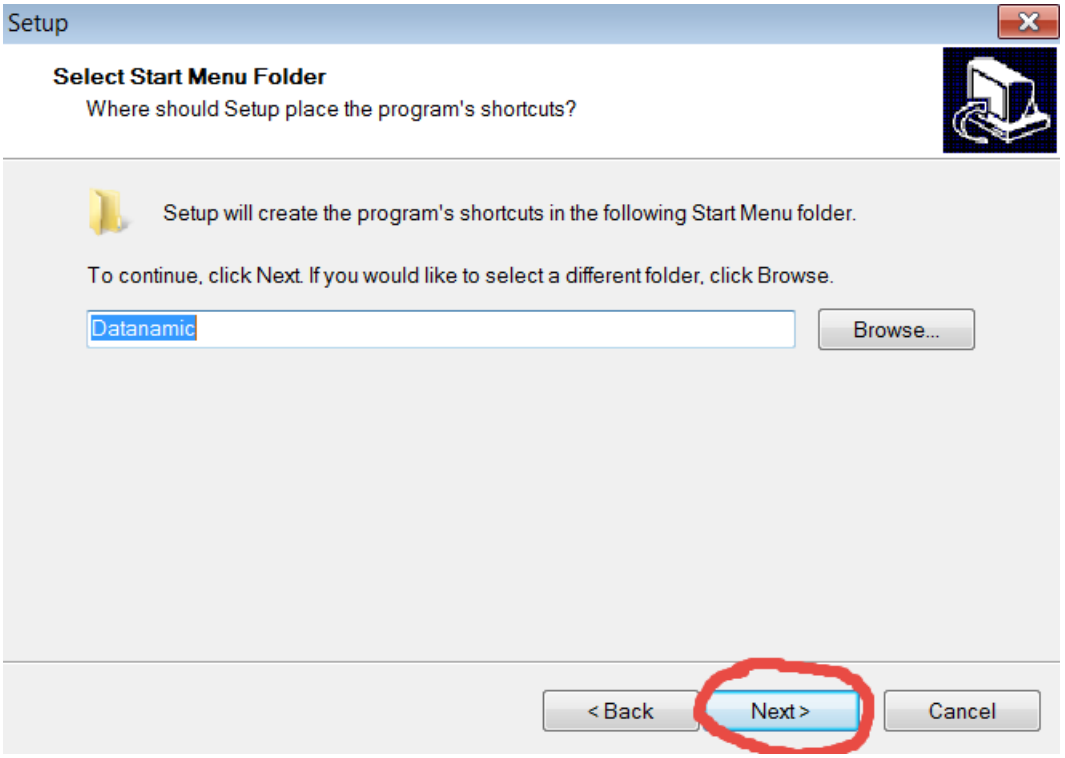
<u>Πώς να εγκαταστήσουμε το Datanamic Data Generator για PostgreSQL και πώς να κατασκευάσουμε τα «ψεύτικα» δεδομένα μας.</u>

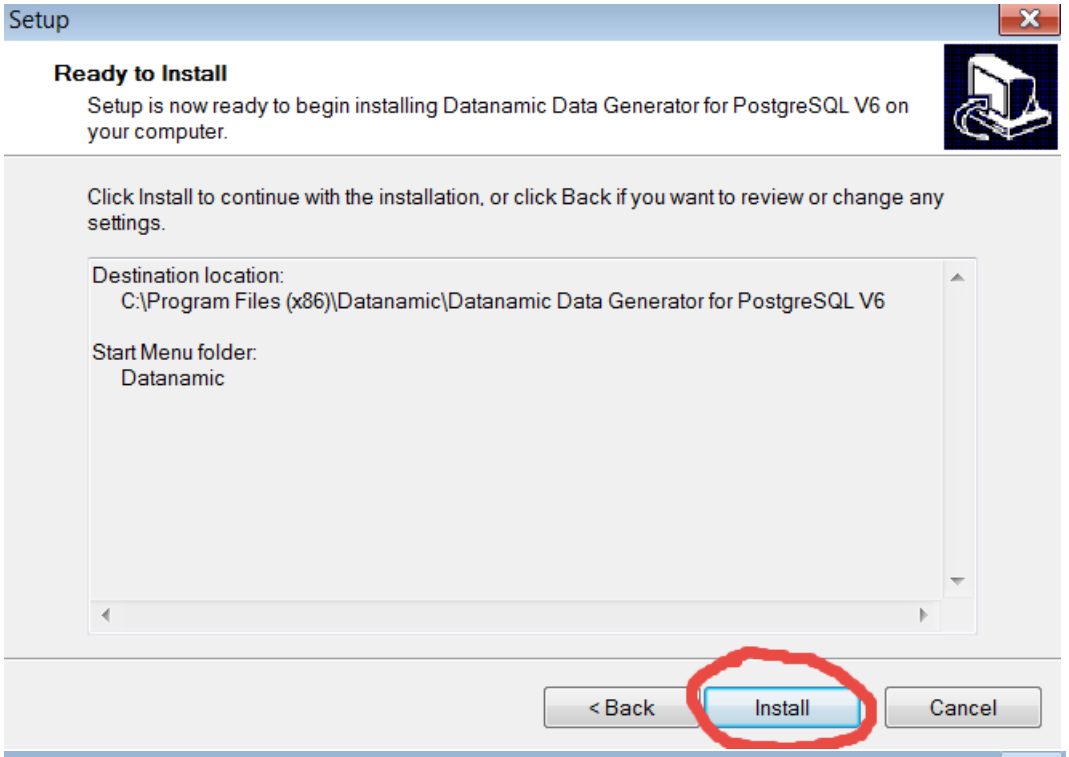
- Κατεβάζουμε την τελευταία έκδοση του Datanamic Data Generator, ακολουθώντας το σύνδεσμο http://datanamic.com/download/download-datagen-for-postgresql.html
- Έπειτα ακολουθούμε τις παρακάτω εικόνες:

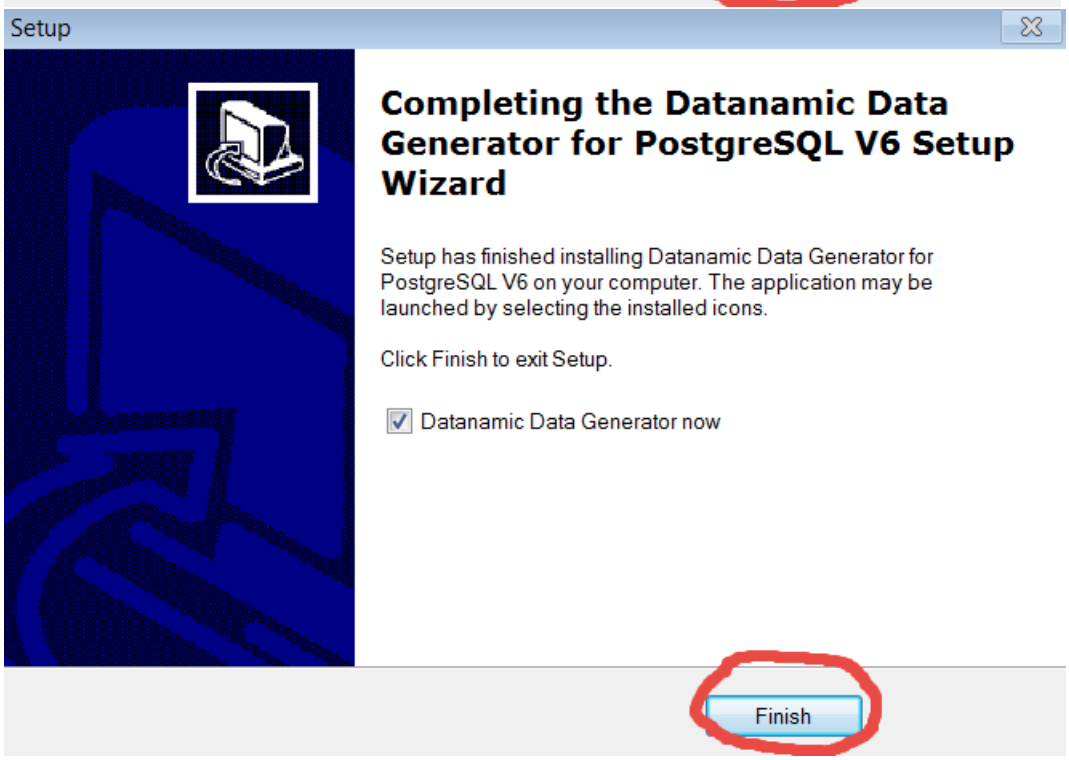


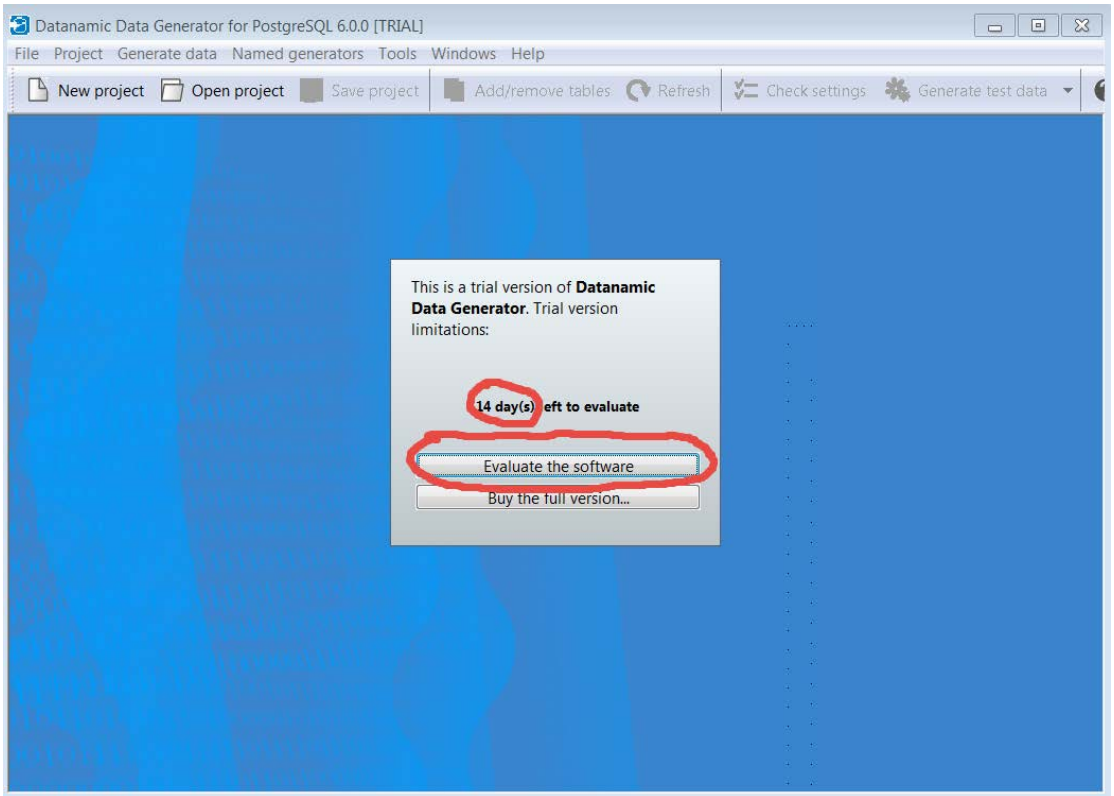


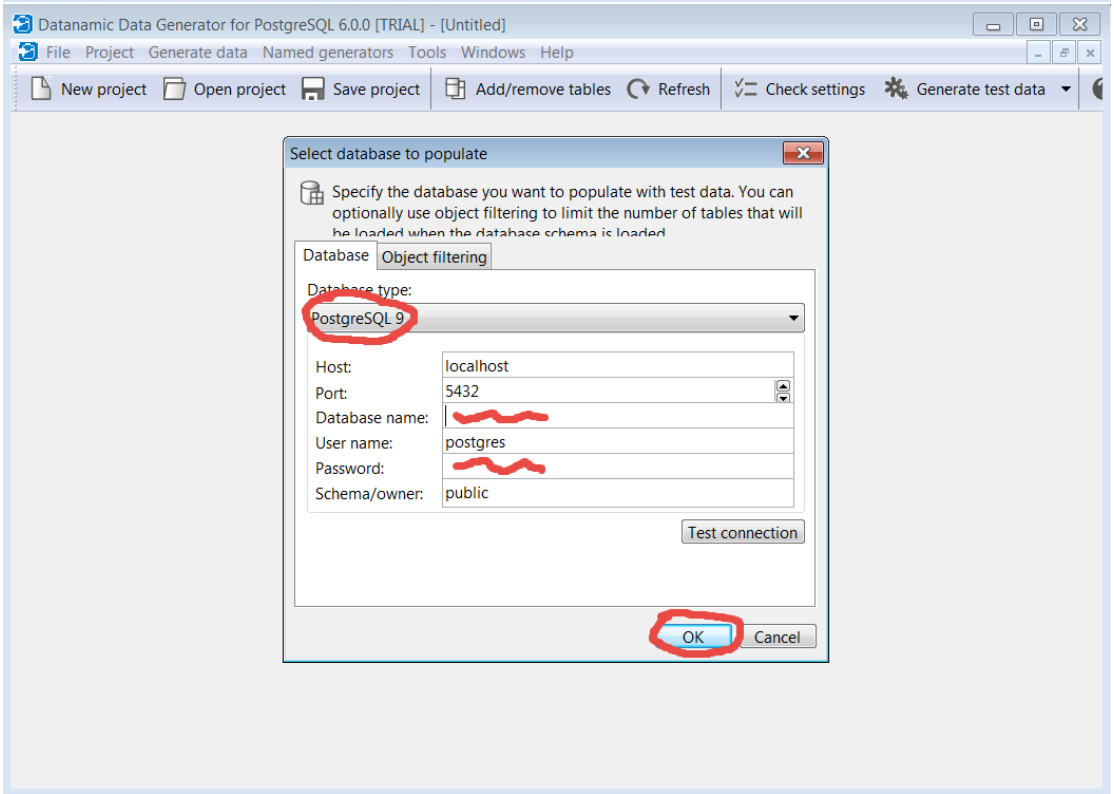

Setup
Select Destination Location
Where should Datanamic Data Generator for PostgreSQL V6 be installed?
Setup will install Datanamic Data Generator for PostgreSQL V6 into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
gram Files (x86)\Datanamic\Datanamic Data Generator for PostgreSQL V6
Browse...
At least 14,9 MB of free disk space is required.
< Back
Next >
Cancel

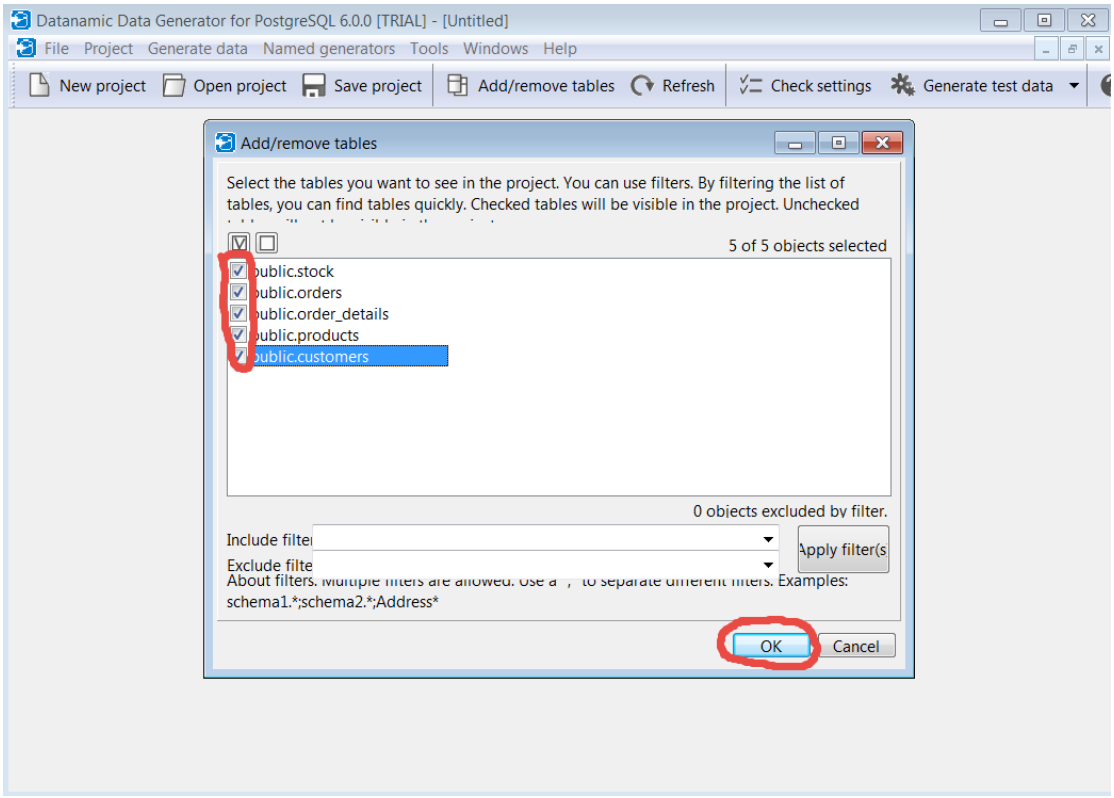

Setup
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Datanamic
Browse...
< Back
Next >
Cancel

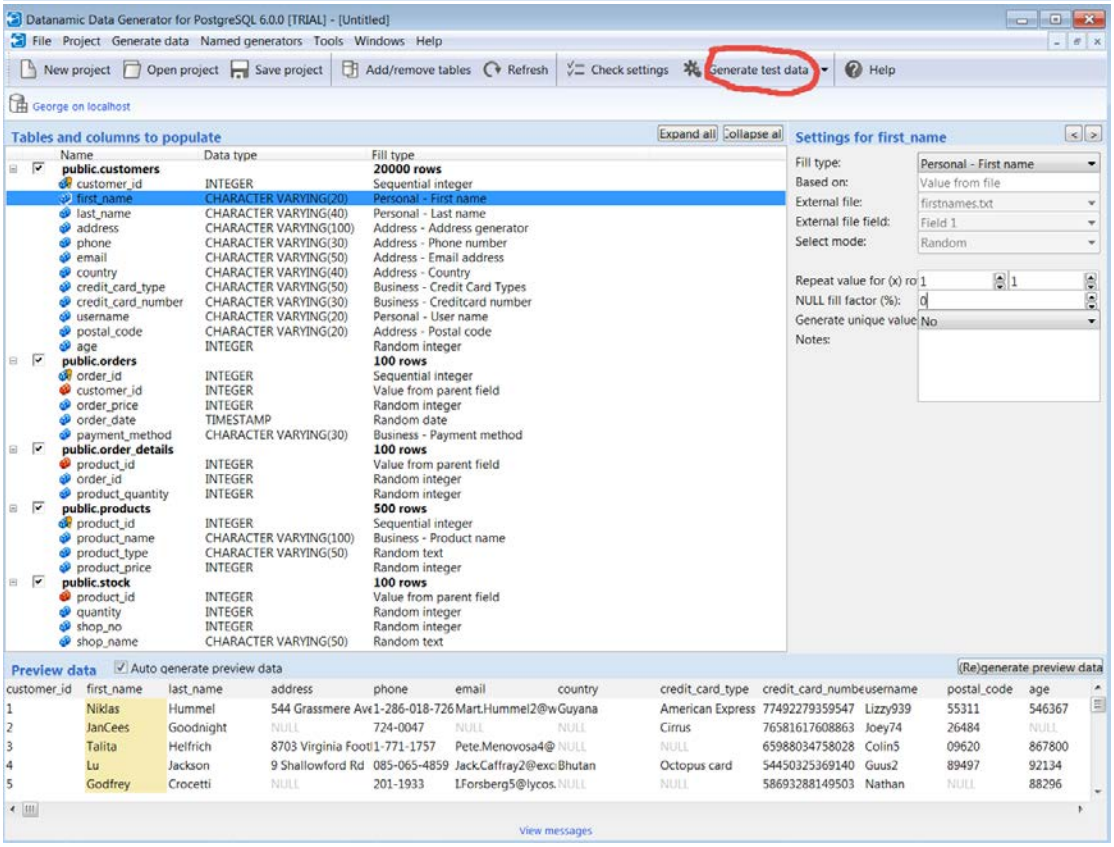

Setup
Ready to Install
Setup is now ready to begin installing Datanamic Data Generator for PostgreSQL V6 on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Destination location:
C:\Program Files (x86)\Datanamic\Datanamic Data Generator for PostgreSQL V6
Start Menu folder:
Datanamic
< Back
Install
Cancel


Setup
Completing the Datanamic Data Generator for PostgreSQL V6 Setup Wizard
Setup has finished installing Datanamic Data Generator for PostgreSQL V6 on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
Datanamic Data Generator now
Finish


Datanamic Data Generator for PostgreSQL 6.0.0 [TRIAL]
File Project Generate data Named generators Tools Windows Help
New project
Open project
Save project
Add/remove tables
Refresh
Check settings
Generate test data
This is a trial version of Datanamic Data Generator. Trial version limitations:
14 day(s) left to evaluate
Evaluate the software
Buy the full version...


Datanamic Data Generator for PostgreSQL 6.0.0 [TRIAL] - [Untitled]
File Project Generate data Named generators Tools Windows Help
New project
Open project
Save project
Add/remove tables
Refresh
Check settings
Generate test data
Select database to populate
Specify the database you want to populate with test data. You can optionally use object filtering to limit the number of tables that will
Database
Object filtering
Database type:
PostgreSQL 9
Host:
localhost
Port:
5432
Database name:
User name:
postgres
Password:
Schema/owner:
public
Test connection
OK
Cancel


Datanamic Data Generator for PostgreSQL 6.0.0 [TRIAL] - [Untitled]
File Project Generate data Named generators Tools Windows Help
New project Open project Save project Add/remove tables Refresh Check settings Generate test data
Add/remove tables
Select the tables you want to see in the project. You can use filters. By filtering the list of tables, you can find tables quickly. Checked tables will be visible in the project. Unchecked
5 of 5 objects selected
public.stock
public.orders
public.order_details
public.products
public.customers
0 objects excluded by filter.
Include filte
Exclude filte
Apply filter(s
About filters. Multiple filters are allowed. Use a ',' to separate different filters. Examples:
schema1.*;schema2.*;Address*
OK
Cancel


Datanamic Data Generator for PostgreSQL 6.0.0 [TRIAL] - [Untitled]
File Project Generate data Named generators Tools Windows Help
New project Open project Save project Add/remove tables Refresh Check settings Generate test data Help
George on localhost
Tables and columns to populate
Expand all Collapse all
Name Data type Fill type
public.customers 20000 rows
customer_id INTEGER Sequential integer
first_name CHARACTER VARYING(20) Personal - First name
last_name CHARACTER VARYING(40) Personal - Last name
address CHARACTER VARYING(100) Address - Address generator
phone CHARACTER VARYING(30) Address - Phone number
email CHARACTER VARYING(50) Address - Email address
country CHARACTER VARYING(40) Address - Country
credit_card_type CHARACTER VARYING(50) Business - Credit Card Types
credit_card_number CHARACTER VARYING(30) Business - Creditcard number
username CHARACTER VARYING(20) Personal - User name
postal_code CHARACTER VARYING(20) Address - Postal code
age INTEGER Random integer
public.orders 100 rows
order_id INTEGER Sequential integer
customer_id INTEGER Value from parent field
order_price INTEGER Random integer
order_date TIMESTAMP Random date
payment_method CHARACTER VARYING(30) Business - Payment method
public.order_details 100 rows
product_id INTEGER Value from parent field
order_id INTEGER Random integer
product_quantity INTEGER Random integer
public.products 500 rows
product_id INTEGER Sequential integer
product_name CHARACTER VARYING(100) Business - Product name
product_type CHARACTER VARYING(50) Random text
product_price INTEGER Random integer
public.stock 100 rows
product_id INTEGER Value from parent field
quantity INTEGER Random integer
shop_no INTEGER Random integer
shop_name CHARACTER VARYING(50) Random text
Settings for first_name
Fill type: Personal - First name
Based on: Value from file
External file: firstnames.txt
External file field: Field 1
Select mode: Random
Repeat value for (x) ro 1 1
NULL fill factor (%): 0
Generate unique value No
Notes:
Preview data
Auto generate preview data
(Re)generate preview data
customer_id first_name last_name address phone email country credit_card_type credit_card_numbe username postal_code age
1 Niklas Hummel 544 Grassmere Ave 1-286-018-726 Mart.Hummel2@w Guyana American Express 77492279359547 Lizzy939 55311 546367
2 JanCees Goodnight NULL 724-0047 NULL NULL Cirrus 76581617608863 Joey74 26484 NULL
3 Talita Helfrich 8703 Virginia Foot 1-771-1757 Pete.Menovosa4@ NULL NULL 65988034758028 Colin5 09620 867800
4 Lu Jackson 9 Shallowford Rd 085-065-4859 Jack.Caffray2@exc Bhutan Octopus card 54450325369140 Guus2 89497 92134
5 Godfrey Crocetti NULL 201-1933 LForsberg5@lycos. NULL NULL 58693288149503 Nathan NULL 88296
View messages

Ερωτήματα

Τα ερωτήματα υπάρχουν και στο συνημμένο αρχείο με όνομα Queries.

1) Εμφάνιση των πελατών με ηλικία από 20 μέχρι 30.

```
SELECT *
FROM customers
WHERE age BETWEEN 20 AND 30
```

2) Εμφάνιση των πελατών με ηλικία από 20 μέχρι 30, ταξινομημένοι κατά το επίθετο τους.

```
SELECT *
FROM customers
WHERE age BETWEEN 20 AND 30
ORDER BY last_name
```

3) Εμφάνιση όλων των παραγγελιών από 01/04/2016 μέχρι 30/04/2016.

```
SELECT last_name, first_name, order_date ,order_price
FROM customers, orders
WHERE order_date BETWEEN '2016-04-01' AND '2016-04-30' AND
        orders.customer_id = customers.customer_id
```

4) Εμφάνιση όλων των παραγγελιών από 01/04/2016 μέχρι 30/04/2016, ταξινομημένων κατά το επίθετο των πελατών.

```
SELECT last_name, first_name, order_date ,order_price
FROM customers, orders
WHERE order_date BETWEEN '2016-04-01' AND '2016-04-30' AND
        orders.customer_id = customers.customer_id
ORDER BY last_name
```

5) Εμφάνιση όλων των παραγγελιών ενός συγκεκριμένου πελάτη του “Stephen Love”, ταξινομημένων ημερολογιακά.

```
SELECT last_name, first_name, order_date ,order_price, payment_method
FROM customers, orders
WHERE last_name = 'Love' AND
        first_name = 'Stephen' AND
        orders.customer_id = customers.customer_id
ORDER BY order_date
```

6) Εμφάνιση της ποσότητας των διαφορετικών προϊόντων, μιας συγκεκριμένης παραγγελίας, ενός συγκεκριμένου πελάτη του “Stephen Love”, ταξινομημένων κατά τον κωδικό των προϊόντων.

```
SELECT last_name, first_name, order_date ,order_price, payment_method,
        product_quantity, product_id
FROM customers, orders, order_details
WHERE last_name = 'Love' AND
        first_name = 'Stephen' AND
        order_date = '2016-05-13 07:31:00' AND
        orders.customer_id = customers.customer_id AND
        orders.order_id = order_details.order_id
ORDER BY product_id
```

7) Αναλυτική εμφάνιση των προϊόντων του ερωτήματος 6, ταξινομημένων κατά το όνομα των προϊόντων.

```
SELECT last_name, first_name, order_date, product_name, product_type,
        product_quantity, product_price
FROM customers, orders, order_details, products
WHERE last_name = 'Love' AND
        first_name = 'Stephen' AND
        order_date = '2016-05-13 07:31:00' AND
        orders.customer_id = customers.customer_id AND
        orders.order_id = order_details.order_id AND
        order_details.product_id = products.product_id
ORDER BY product_name
```

8) Εμφάνιση των καταστημάτων στα οποία υπάρχουν τα προϊόντα του ερωτήματος 7, ταξινομημένων κατά το όνομα των προϊόντων.

```
SELECT last_name, first_name, order_date, product_name, product_type,
        product_quantity, product_price, shop_name, quantity
FROM customers, orders, order_details, products, stock
WHERE last_name = 'Love' AND
        first_name = 'Stephen' AND
        order_date = '2016-05-13 07:31:00' AND
        orders.customer_id =  customers.customer_id AND
        orders.order_id = order_details.order_id AND
        order_details.product_id = products.product_id AND
        products.product_id = stock.product_id
ORDER BY product_name
```

9) Αναζήτηση των προϊόντων που προτιμούν τα άτομα με ηλικίες από 20 έως 30.

```
SELECT DISTINCT product_name, product_type, product_price
FROM customers, orders, order_details, products
WHERE age BETWEEN 20 AND 30 AND
        orders.customer_id = customers.customer_id AND
        orders.order_id = order_details.order_id AND
        order_details.product_id = products.product_id
ORDER BY product_name
```

<u>Αποτελέσματα μέτρησης χρόνων των ερωτημάτων, χωρίς την χρήση ευρετηρίων.</u>

Οι μετρήσεις χρόνων έχουν γίνει με τη βοήθεια της SQL εντολής EXPLAIN ANALYZE.

1)

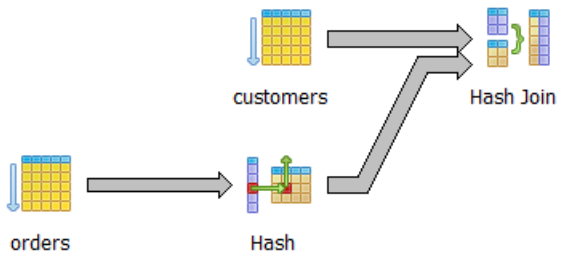
	QUERY PLAN text
1	Seq Scan on customers (cost=0.00..709.00 rows=2664 width=123) (actual time=0.019..6.565 rows=2664 loops=1)
2	Filter: ((age >= 20) AND (age <= 30))
3	Rows Removed by Filter: 17336
4	Planning time: 0.124 ms
5	Execution time: 6.666 ms

2)

	QUERY PLAN text
1	Sort (cost=860.57..867.23 rows=2664 width=123) (actual time=18.346..18.618 rows=2664 loops=1)
2	Sort Key: last_name
3	Sort Method: quicksort Memory: 781kB
4	-> Seq Scan on customers (cost=0.00..709.00 rows=2664 width=123) (actual time=0.021..7.310 rows=2664 loops=1)
5	Filter: ((age >= 20) AND (age <= 30))
6	Rows Removed by Filter: 17336
7	Planning time: 0.368 ms
8	Execution time: 19.076 ms

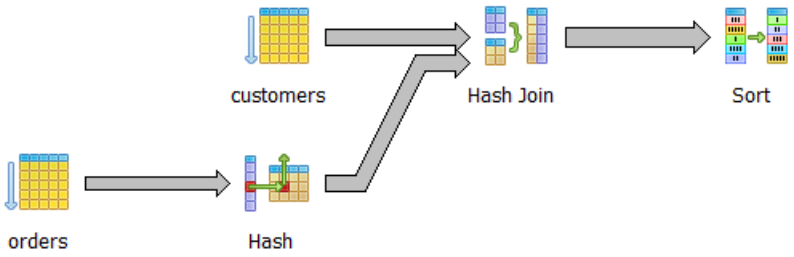
3)

	QUERY PLAN text
1	Hash Join (cost=190.68..1004.21 rows=454 width=25) (actual time=2.613..13.584 rows=452 loops=1)
2	Hash Cond: (customers.customer_id = orders.customer_id)
3	-> Seq Scan on customers (cost=0.00..609.00 rows=20000 width=17) (actual time=0.011..6.945 rows=20000 loops=1)
4	-> Hash (cost=185.00..185.00 rows=454 width=16) (actual time=2.573..2.573 rows=452 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 30kB
6	-> Seq Scan on orders (cost=0.00..185.00 rows=454 width=16) (actual time=0.018..2.410 rows=452 loops=1)
7	Filter: ((order_date >= '2016-04-01 00:00:00'::timestamp without time zone) AND (order_date <= '2016-04-30 00:00:
8	Rows Removed by Filter: 7548
9	Planning time: 0.410 ms
10	Execution time: 13.704 ms



4)

	QUERY PLAN text
1	Sort (cost=1024.25..1025.39 rows=454 width=25) (actual time=16.783..16.816 rows=452 loops=1)
2	Sort Key: customers.last_name
3	Sort Method: quicksort Memory: 60kB
4	-> Hash Join (cost=190.68..1004.21 rows=454 width=25) (actual time=2.756..13.874 rows=452 loops=1)
5	Hash Cond: (customers.customer_id = orders.customer_id)
6	-> Seq Scan on customers (cost=0.00..609.00 rows=20000 width=17) (actual time=0.011..7.068 rows=20000 loops=1)
7	-> Hash (cost=185.00..185.00 rows=454 width=16) (actual time=2.712..2.712 rows=452 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 30kB
9	-> Seq Scan on orders (cost=0.00..185.00 rows=454 width=16) (actual time=0.016..2.509 rows=452 loops=1)
10	Filter: ((order_date >= '2016-04-01 00:00:00'::timestamp without time zone) AND (order_date <= '2016-04-30 00:00:00'::timestamp without time zone))
11	Rows Removed by Filter: 7548
12	Planning time: 0.416 ms
13	Execution time: 16.963 ms

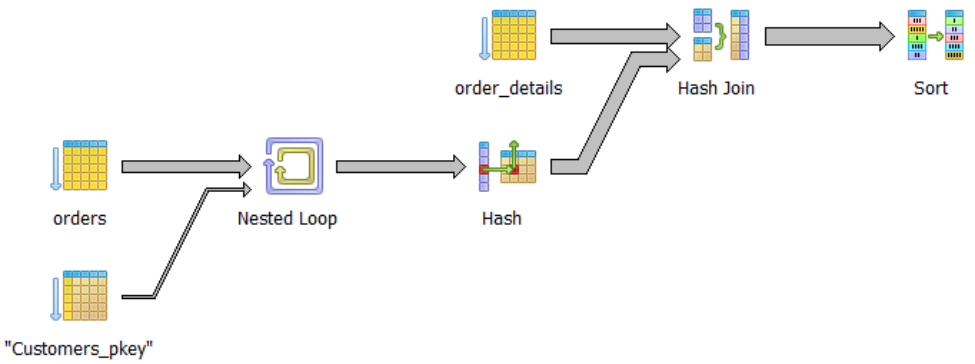


5)

	QUERY PLAN text
1	Sort (cost=884.03..884.04 rows=1 width=35) (actual time=10.033..10.033 rows=3 loops=1)
2	Sort Key: orders.order_date
3	Sort Method: quicksort Memory: 25kB
4	-> Hash Join (cost=709.01..884.02 rows=1 width=35) (actual time=6.937..10.011 rows=3 loops=1)
5	Hash Cond: (orders.customer_id = customers.customer_id)
6	-> Seq Scan on orders (cost=0.00..145.00 rows=8000 width=26) (actual time=0.014..1.322 rows=8000 loops=1)
7	-> Hash (cost=709.00..709.00 rows=1 width=17) (actual time=6.893..6.893 rows=1 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Seq Scan on customers (cost=0.00..709.00 rows=1 width=17) (actual time=0.017..6.886 rows=1 loops=1)
10	Filter: (((last_name)::text = 'Love'::text) AND ((first_name)::text = 'Stephen'::text))
11	Rows Removed by Filter: 19999
12	Planning time: 0.428 ms
13	Execution time: 10.116 ms

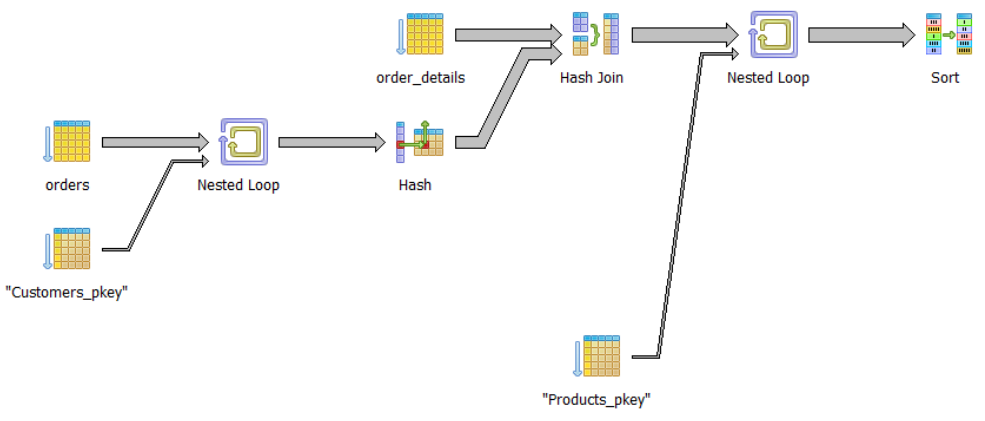
6)

```
QUERY PLAN
text
Sort  (cost=653.14..653.14 rows=3 width=43) (actual time=12.022..12.022 rows=2 loops=1)
  Sort Key: order_details.product_id
  Sort Method: quicksort  Memory: 25kB
  ->  Hash Join  (cost=173.33..653.11 rows=3 width=43) (actual time=5.619..12.004 rows=2 loops=1)
        Hash Cond: (order_details.order_id = orders.order_id)
        ->  Seq Scan on order_details  (cost=0.00..386.00 rows=25000 width=12) (actual time=0.016..4.344 rows=25000 loops=1)
        ->  Hash  (cost=173.32..173.32 rows=1 width=39) (actual time=2.236..2.236 rows=1 loops=1)
              Buckets: 1024  Batches: 1  Memory Usage: 9kB
              ->  Nested Loop  (cost=0.29..173.32 rows=1 width=39) (actual time=0.031..2.229 rows=1 loops=1)
                    ->  Seq Scan on orders  (cost=0.00..165.00 rows=1 width=30) (actual time=0.014..2.210 rows=1 loops=1)
                          Filter: (order_date = '2016-05-13 07:31:00'::timestamp without time zone)
                          Rows Removed by Filter: 7999
                    ->  Index Scan using "Customers_pkey" on customers  (cost=0.29..8.31 rows=1 width=17) (actual time=0.013..0.014 rows=1 loops=1)
                          Index Cond: (customer_id = orders.customer_id)
                          Filter: (((last_name)::text = 'Love'::text) AND ((first_name)::text = 'Stephen'::text))
Planning time: 0.909 ms
Execution time: 12.147 ms
```



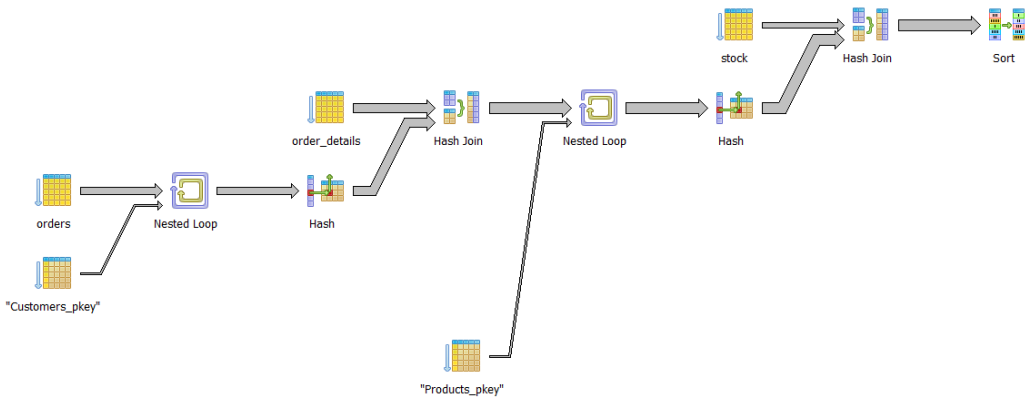
7)

```
QUERY PLAN
text
Sort  (cost=654.08..654.08 rows=3 width=49) (actual time=12.139..12.140 rows=2 loops=1)
  Sort Key: products.product_name
  Sort Method: quicksort  Memory: 25kB
  ->  Nested Loop  (cost=173.61..654.05 rows=3 width=49) (actual time=5.539..12.103 rows=2 loops=1)
        ->  Hash Join  (cost=173.33..653.11 rows=3 width=29) (actual time=5.525..12.081 rows=2 loops=1)
              Hash Cond: (order_details.order_id = orders.order_id)
              ->  Seq Scan on order_details  (cost=0.00..386.00 rows=25000 width=12) (actual time=0.016..4.194 rows=25000 loops=1)
              ->  Hash  (cost=173.32..173.32 rows=1 width=25) (actual time=2.229..2.229 rows=1 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 9kB
                    ->  Nested Loop  (cost=0.29..173.32 rows=1 width=25) (actual time=0.033..2.222 rows=1 loops=1)
                          ->  Seq Scan on orders  (cost=0.00..165.00 rows=1 width=16) (actual time=0.015..2.203 rows=1 loops=1)
                                Filter: (order_date = '2016-05-13 07:31:00'::timestamp without time zone)
                                Rows Removed by Filter: 7999
                          ->  Index Scan using "Customers_pkey" on customers  (cost=0.29..8.31 rows=1 width=17) (actual time=0.013..0.014 rows=1 loops=1)
                                Index Cond: (customer_id = orders.customer_id)
                                Filter: (((last_name)::text = 'Love'::text) AND ((first_name)::text = 'Stephen'::text))
        ->  Index Scan using "Products_pkey" on products  (cost=0.28..0.30 rows=1 width=28) (actual time=0.006..0.007 rows=1 loops=2)
              Index Cond: (product_id = order_details.product_id)
Planning time: 1.149 ms
Execution time: 12.294 ms
```



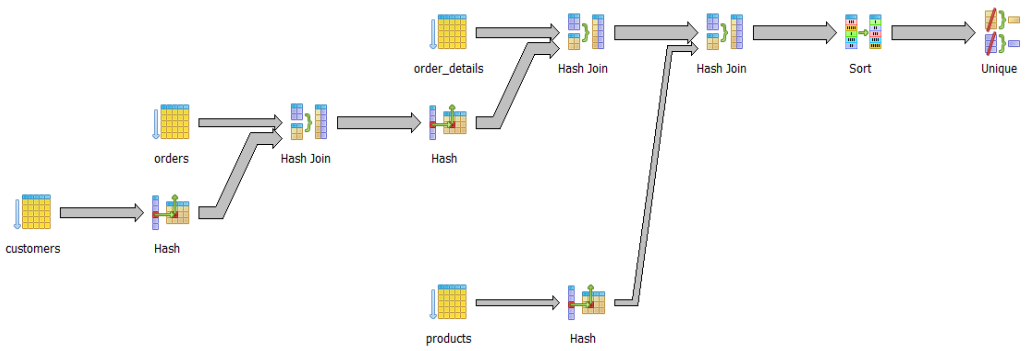
8)

	QUERY PLAN text
1	Sort (cost=757.95..757.96 rows=5 width=65) (actual time=14.156..14.157 rows=5 loops=1)
2	Sort Key: products.product_name
3	Sort Method: quicksort Memory: 25kB
4	-> Hash Join (cost=654.09..757.89 rows=5 width=65) (actual time=13.068..14.113 rows=5 loops=1)
5	Hash Cond: (stock.product_id = order_details.product_id)
6	-> Seq Scan on stock (cost=0.00..85.00 rows=5000 width=20) (actual time=0.014..0.836 rows=5000 loops=1)
7	-> Hash (cost=654.05..654.05 rows=3 width=57) (actual time=12.213..12.213 rows=2 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Nested Loop (cost=173.61..654.05 rows=3 width=57) (actual time=5.871..12.190 rows=2 loops=1)
10	-> Hash Join (cost=173.33..653.11 rows=3 width=29) (actual time=5.792..12.102 rows=2 loops=1)
11	Hash Cond: (order_details.order_id = orders.order_id)
12	-> Seq Scan on order_details (cost=0.00..386.00 rows=25000 width=12) (actual time=0.012..4.188 rows=25000 loops=1)
13	-> Hash (cost=173.32..173.32 rows=1 width=25) (actual time=2.230..2.230 rows=1 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB
15	-> Nested Loop (cost=0.29..173.32 rows=1 width=25) (actual time=0.032..2.224 rows=1 loops=1)
16	-> Seq Scan on orders (cost=0.00..165.00 rows=1 width=16) (actual time=0.014..2.204 rows=1 loops=1)
17	Filter: (order_date = '2016-05-13 07:31:00'::timestamp without time zone)
18	Rows Removed by Filter: 7999
19	-> Index Scan using "Customers_pkey" on customers (cost=0.29..8.31 rows=1 width=17) (actual time=0.013..0.014 rows=1 loops=1)
20	Index Cond: (customer_id = orders.customer_id)
21	Filter: (((last_name)::text = 'Love'::text) AND ((first_name)::text = 'Stephen'::text))
22	-> Index Scan using "Products_pkey" on products (cost=0.28..0.30 rows=1 width=28) (actual time=0.023..0.024 rows=1 loops=2)
23	Index Cond: (product_id = order_details.product_id)
24	Planning time: 2.080 ms
25	Execution time: 14.403 ms



9)

	QUERY PLAN text
1	Unique (cost=1785.54..1818.85 rows=3000 width=24) (actual time=43.561..44.550 rows=1684 loops=1)
2	-> Sort (cost=1785.54..1793.87 rows=3331 width=24) (actual time=43.557..43.688 rows=3592 loops=1)
3	Sort Key: products.product_name, products.product_type, products.product_price
4	Sort Method: quicksort Memory: 375kB
5	-> Hash Join (cost=1031.78..1590.65 rows=3331 width=24) (actual time=22.551..29.226 rows=3592 loops=1)
6	Hash Cond: (order_details.product_id = products.product_id)
7	-> Hash Join (cost=941.28..1454.34 rows=3331 width=4) (actual time=19.727..25.038 rows=3592 loops=1)
8	Hash Cond: (order_details.order_id = orders.order_id)
9	-> Seq Scan on order_details (cost=0.00..386.00 rows=25000 width=8) (actual time=0.011..2.067 rows=25000 loops=1)
10	-> Hash (cost=927.96..927.96 rows=1066 width=4) (actual time=19.650..19.650 rows=1144 loops=1)
11	Buckets: 2048 Batches: 1 Memory Usage: 57kB
12	-> Hash Join (cost=742.30..927.96 rows=1066 width=4) (actual time=16.839..19.352 rows=1144 loops=1)
13	Hash Cond: (orders.customer_id = customers.customer_id)
14	-> Seq Scan on orders (cost=0.00..145.00 rows=8000 width=8) (actual time=0.010..1.014 rows=8000 loops=1)
15	-> Hash (cost=709.00..709.00 rows=2664 width=4) (actual time=16.755..16.755 rows=2664 loops=1)
16	Buckets: 4096 Batches: 1 Memory Usage: 126kB
17	-> Seq Scan on customers (cost=0.00..709.00 rows=2664 width=4) (actual time=0.013..15.372 rows=2664 loops=1)
18	Filter: ((age >= 20) AND (age <= 30))
19	Rows Removed by Filter: 17336
20	-> Hash (cost=53.00..53.00 rows=3000 width=28) (actual time=2.780..2.780 rows=3000 loops=1)
21	Buckets: 4096 Batches: 1 Memory Usage: 218kB
22	-> Seq Scan on products (cost=0.00..53.00 rows=3000 width=28) (actual time=0.010..1.030 rows=3000 loops=1)
23	Planning time: 1.239 ms
24	Execution time: 44.891 ms



Ορισμός Ευρετηρίων

Αρχικά ορίζουμε ευρετήρια σε όλα τα βασικά και ξένα κλειδία. Επιπλέον έχουμε ορίσει ευρετήρια και σε άλλα πεδία, όπως είναι τα πεδία επώνυμο και ηλικία του πίνακα πελατών, το πεδίο της ημερομηνίας των παραγγελιών και το πεδίο του ονόματος κάθε προϊόντος.

Για να ορίσουμε ένα ευρετήριο στο pgAdmin III, ανοίγουμε από τα πλάγια τον πίνακα που επιθυμούμε και κάνουμε δεξί click πάνω στα Indexes και επιλέγουμε New Index.

Πιο απλά μπορείς να ανοίξεις το SQL Tool, να κάνεις Open File το συννημένο αρχείο, με όνομα «Indexes SQL Code» και Execute Query.

Αποτελέσματα μέτρησης χρόνων των ερωτημάτων, με χρήση ευρετηρίων.

1)

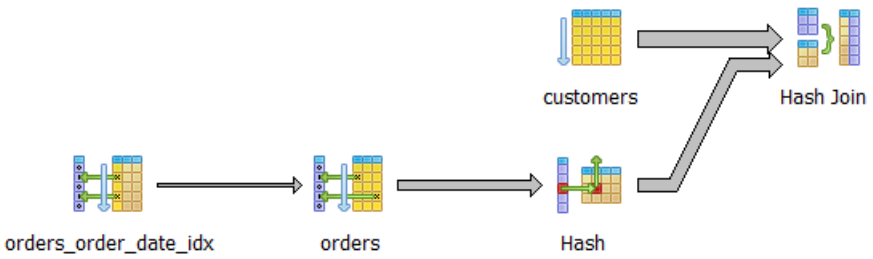
	QUERY PLAN text
1	Bitmap Heap Scan on customers (cost=59.59..488.55 rows=2664 width=123) (actual time=0.813..2.181 rows=2664 loops=1)
2	Recheck Cond: ((age >= 20) AND (age <= 30))
3	Heap Blocks: exact=388
4	-> Bitmap Index Scan on customers_age_idx (cost=0.00..58.93 rows=2664 width=0) (actual time=0.685..0.685 rows=2664 loops=1)
5	Index Cond: ((age >= 20) AND (age <= 30))
6	Planning time: 0.197 ms
7	Execution time: 2.385 ms

2)

	QUERY PLAN text
1	Sort (cost=640.13..646.79 rows=2664 width=123) (actual time=17.215..17.344 rows=2664 loops=1)
2	Sort Key: last_name
3	Sort Method: quicksort Memory: 781kB
4	-> Bitmap Heap Scan on customers (cost=59.59..488.55 rows=2664 width=123) (actual time=0.860..2.320 rows=2664 loops=1)
5	Recheck Cond: ((age >= 20) AND (age <= 30))
6	Heap Blocks: exact=388
7	-> Bitmap Index Scan on customers_age_idx (cost=0.00..58.93 rows=2664 width=0) (actual time=0.737..0.737 rows=2664 loops=1)
8	Index Cond: ((age >= 20) AND (age <= 30))
9	Planning time: 0.237 ms
10	Execution time: 17.736 ms

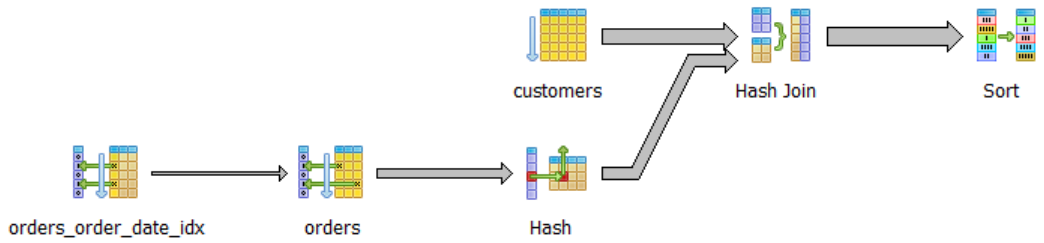
3)

	QUERY PLAN text
1	Hash Join (cost=90.42..883.96 rows=454 width=25) (actual time=0.718..9.311 rows=452 loops=1)
2	Hash Cond: (customers.customer_id = orders.customer_id)
3	-> Seq Scan on customers (cost=0.00..589.00 rows=20000 width=17) (actual time=0.010..3.873 rows=20000 loops=1)
4	-> Hash (cost=84.75..84.75 rows=454 width=16) (actual time=0.666..0.666 rows=452 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 30kB
6	-> Bitmap Heap Scan on orders (cost=12.94..84.75 rows=454 width=16) (actual time=0.152..0.486 rows=452 loops=1)
7	Recheck Cond: ((order_date >= '2016-04-01 00:00:00'::timestamp without time zone) AND (order_date <= '2016-04-30 00:00:00'::timestamp without time zone))
8	Heap Blocks: exact=65
9	-> Bitmap Index Scan on orders_order_date_idx (cost=0.00..12.82 rows=454 width=0) (actual time=0.126..0.126 rows=452 loops=1)
10	Index Cond: ((order_date >= '2016-04-01 00:00:00'::timestamp without time zone) AND (order_date <= '2016-04-30 00:00:00'::timestamp without time zo
11	Planning time: 0.672 ms
12	Execution time: 9.442 ms



4)

	QUERY PLAN text
1	Sort (cost=904.00..905.13 rows=454 width=25) (actual time=12.141..12.175 rows=452 loops=1)
2	Sort Key: customers.last_name
3	Sort Method: quicksort Memory: 60kB
4	-> Hash Join (cost=90.42..883.96 rows=454 width=25) (actual time=0.728..9.304 rows=452 loops=1)
5	Hash Cond: (customers.customer_id = orders.customer_id)
6	-> Seq Scan on customers (cost=0.00..589.00 rows=20000 width=17) (actual time=0.011..3.910 rows=20000 loops=1)
7	-> Hash (cost=84.75..84.75 rows=454 width=16) (actual time=0.682..0.682 rows=452 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 30kB
9	-> Bitmap Heap Scan on orders (cost=12.94..84.75 rows=454 width=16) (actual time=0.151..0.510 rows=452 loops=1)
10	Recheck Cond: ((order_date >= '2016-04-01 00:00:00'::timestamp without time zone) AND (order_date <= '2016-04-30 00:00:00'::timestamp without time zone
11	Heap Blocks: exact=65
12	-> Bitmap Index Scan on orders_order_date_idx (cost=0.00..12.82 rows=454 width=0) (actual time=0.126..0.126 rows=452 loops=1)
13	Index Cond: ((order_date >= '2016-04-01 00:00:00'::timestamp without time zone) AND (order_date <= '2016-04-30 00:00:00'::timestamp without time
14	Planning time: 0.638 ms
15	Execution time: 12.313 ms

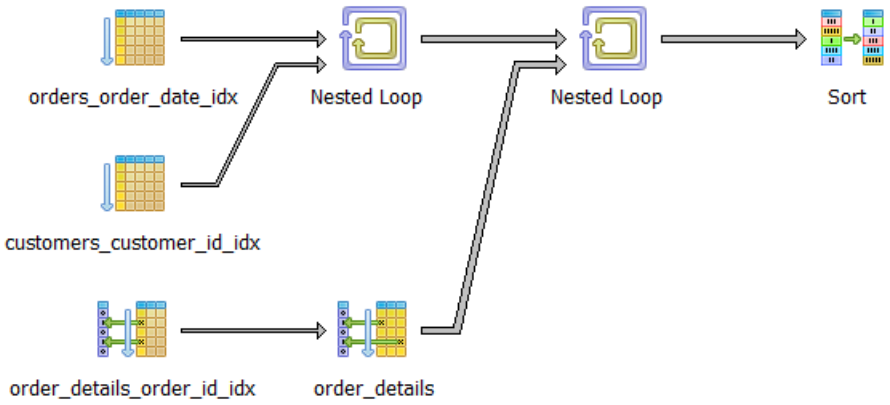


5)

	QUERY PLAN text
1	Sort (cost=165.32..165.33 rows=1 width=35) (actual time=0.212..0.212 rows=3 loops=1)
2	Sort Key: orders.order_date
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=4.98..165.31 rows=1 width=35) (actual time=0.098..0.188 rows=3 loops=1)
5	-> Bitmap Heap Scan on customers (cost=4.69..154.10 rows=1 width=17) (actual time=0.084..0.170 rows=1 loops=1)
6	Recheck Cond: ((last_name)::text = 'Love'::text)
7	Filter: ((first_name)::text = 'Stephen'::text)
8	Rows Removed by Filter: 44
9	Heap Blocks: exact=43
10	-> Bitmap Index Scan on customers_last_name_idx (cost=0.00..4.69 rows=54 width=0) (actual time=0.058..0.058 rows=45 loops=1)
11	Index Cond: ((last_name)::text = 'Love'::text)
12	-> Index Scan using orders_customer_id_idx on orders (cost=0.28..11.19 rows=2 width=26) (actual time=0.008..0.011 rows=3 loops=1)
13	Index Cond: (customer_id = customers.customer_id)
14	Planning time: 0.764 ms
15	Execution time: 0.335 ms

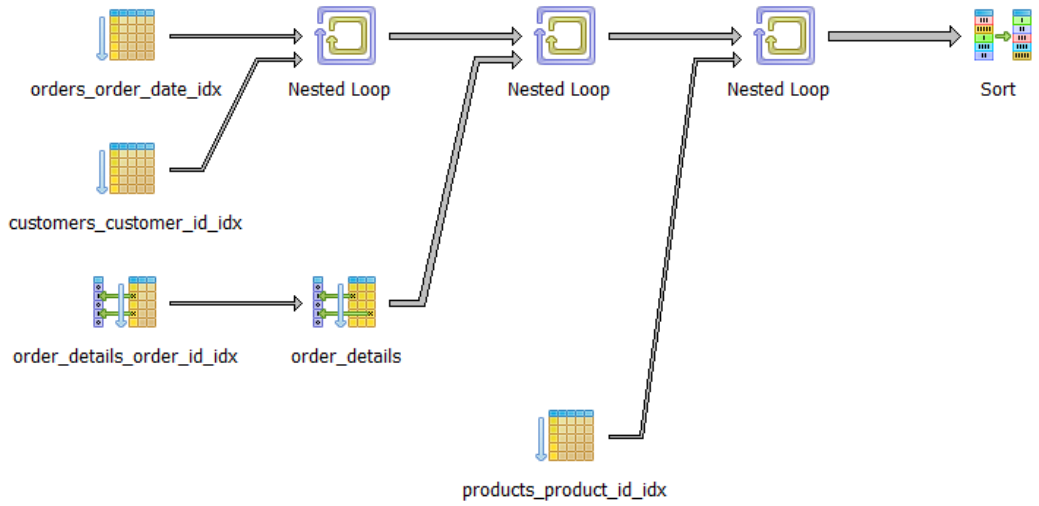
6)

```
QUERY PLAN
text
Sort  (cost=34.99..35.00 rows=3 width=43) (actual time=0.079..0.080 rows=2 loops=1)
  Sort Key: order_details.product_id
  Sort Method: quicksort  Memory: 25kB
  ->  Nested Loop  (cost=4.89..34.97 rows=3 width=43) (actual time=0.058..0.061 rows=2 loops=1)
        ->  Nested Loop  (cost=0.57..16.62 rows=1 width=39) (actual time=0.036..0.038 rows=1 loops=1)
              ->  Index Scan using orders_order_date_idx on orders  (cost=0.28..8.30 rows=1 width=30) (actual time=0.017..0.018 rows=1 loops=1)
                    Index Cond: (order_date = '2016-05-13 07:31:00'::timestamp without time zone)
              ->  Index Scan using customers_customer_id_idx on customers  (cost=0.29..8.31 rows=1 width=17) (actual time=0.013..0.014 rows=1 loops=1)
                    Index Cond: (customer_id = orders.customer_id)
                    Filter: (((last_name)::text = 'Love'::text) AND ((first_name)::text = 'Stephen'::text))
        ->  Bitmap Heap Scan on order_details  (cost=4.32..18.31 rows=4 width=12) (actual time=0.011..0.012 rows=2 loops=1)
              Recheck Cond: (order_id = orders.order_id)
              Heap Blocks: exact=1
              ->  Bitmap Index Scan on order_details_order_id_idx  (cost=0.00..4.32 rows=4 width=0) (actual time=0.007..0.007 rows=2 loops=1)
                    Index Cond: (order_id = orders.order_id)
Planning time: 1.336 ms
Execution time: 0.240 ms
```



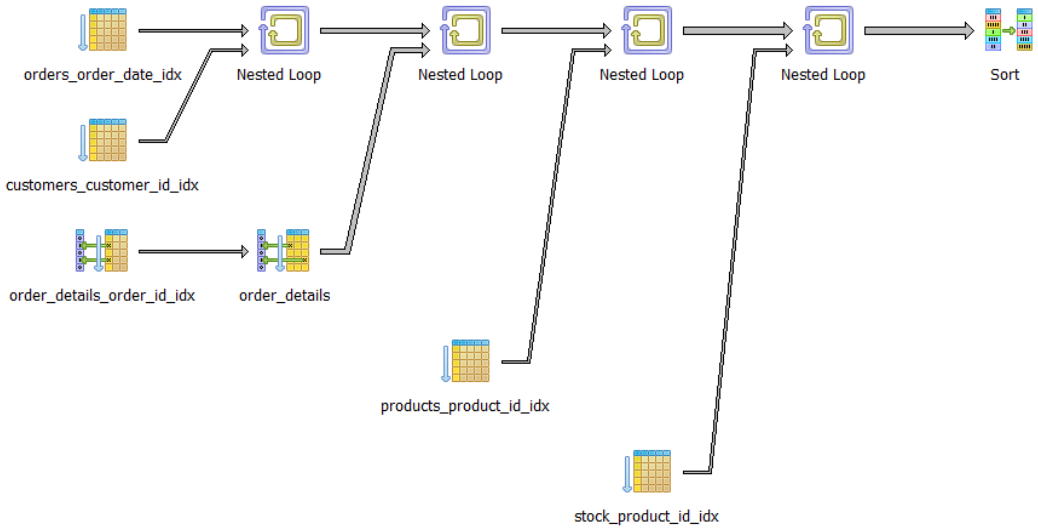
7)

```
QUERY PLAN
text
Sort  (cost=35.93..35.94 rows=3 width=49) (actual time=0.108..0.109 rows=2 loops=1)
  Sort Key: products.product_name
  Sort Method: quicksort  Memory: 25kB
  ->  Nested Loop  (cost=5.17..35.91 rows=3 width=49) (actual time=0.064..0.073 rows=2 loops=1)
        ->  Nested Loop  (cost=4.89..34.97 rows=3 width=29) (actual time=0.056..0.058 rows=2 loops=1)
              ->  Nested Loop  (cost=0.57..16.62 rows=1 width=25) (actual time=0.035..0.036 rows=1 loops=1)
                    ->  Index Scan using orders_order_date_idx on orders  (cost=0.28..8.30 rows=1 width=16) (actual time=0.015..0.016 rows=1 loops=1)
                          Index Cond: (order_date = '2016-05-13 07:31:00'::timestamp without time zone)
                    ->  Index Scan using customers_customer_id_idx on customers  (cost=0.29..8.31 rows=1 width=17) (actual time=0.013..0.014 rows=1 loops=1)
                          Index Cond: (customer_id = orders.customer_id)
                          Filter: (((last_name)::text = 'Love'::text) AND ((first_name)::text = 'Stephen'::text))
              ->  Bitmap Heap Scan on order_details  (cost=4.32..18.31 rows=4 width=12) (actual time=0.011..0.012 rows=2 loops=1)
                    Recheck Cond: (order_id = orders.order_id)
                    Heap Blocks: exact=1
                    ->  Bitmap Index Scan on order_details_order_id_idx  (cost=0.00..4.32 rows=4 width=0) (actual time=0.007..0.007 rows=2 loops=1)
                          Index Cond: (order_id = orders.order_id)
        ->  Index Scan using products_product_id_idx on products  (cost=0.28..0.30 rows=1 width=28) (actual time=0.005..0.005 rows=1 loops=2)
              Index Cond: (product_id = order_details.product_id)
Planning time: 1.788 ms
Execution time: 0.266 ms
```



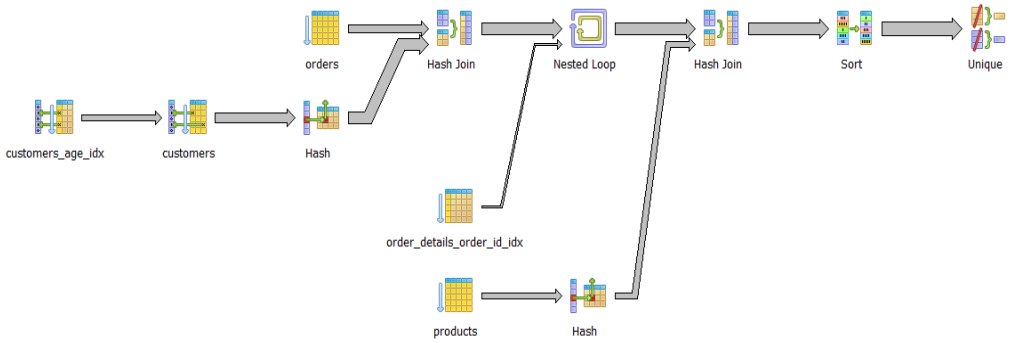
8)

	QUERY PLAN text
1	Sort (cost=37.00..37.02 rows=5 width=65) (actual time=0.126..0.127 rows=5 loops=1)
2	Sort Key: products.product_name
3	Sort Method: quicksort Memory: 25kB
4	-> Nested Loop (cost=5.45..36.95 rows=5 width=65) (actual time=0.067..0.090 rows=5 loops=1)
5	-> Nested Loop (cost=5.17..35.91 rows=3 width=57) (actual time=0.060..0.069 rows=2 loops=1)
6	-> Nested Loop (cost=4.89..34.97 rows=3 width=29) (actual time=0.053..0.055 rows=2 loops=1)
7	-> Nested Loop (cost=0.57..16.62 rows=1 width=25) (actual time=0.032..0.033 rows=1 loops=1)
8	-> Index Scan using orders_order_date_idx on orders (cost=0.28..8.30 rows=1 width=16) (actual time=0.014..0.015 rows=1 loops=1)
9	Index Cond: (order_date = '2016-05-13 07:31:00'::timestamp without time zone)
10	-> Index Scan using customers_customer_id_idx on customers (cost=0.29..8.31 rows=1 width=17) (actual time=0.012..0.012 rows=1 loops=1)
11	Index Cond: (customer_id = orders.customer_id)
12	Filter: (((last_name)::text = 'Love'::text) AND ((first_name)::text = 'Stephen'::text))
13	-> Bitmap Heap Scan on order_details (cost=4.32..18.31 rows=4 width=12) (actual time=0.011..0.011 rows=2 loops=1)
14	Recheck Cond: (order_id = orders.order_id)
15	Heap Blocks: exact=1
16	-> Bitmap Index Scan on order_details_order_id_idx (cost=0.00..4.32 rows=4 width=0) (actual time=0.006..0.006 rows=2 loops=1)
17	Index Cond: (order_id = orders.order_id)
18	-> Index Scan using products_product_id_idx on products (cost=0.28..0.30 rows=1 width=28) (actual time=0.004..0.005 rows=1 loops=2)
19	Index Cond: (product_id = order_details.product_id)
20	-> Index Scan using stock_product_id_idx on stock (cost=0.28..0.33 rows=2 width=20) (actual time=0.005..0.008 rows=3 loops=2)
21	Index Cond: (product_id = order_details.product_id)
22	Planning time: 3.050 ms
23	Execution time: 0.279 ms



9)

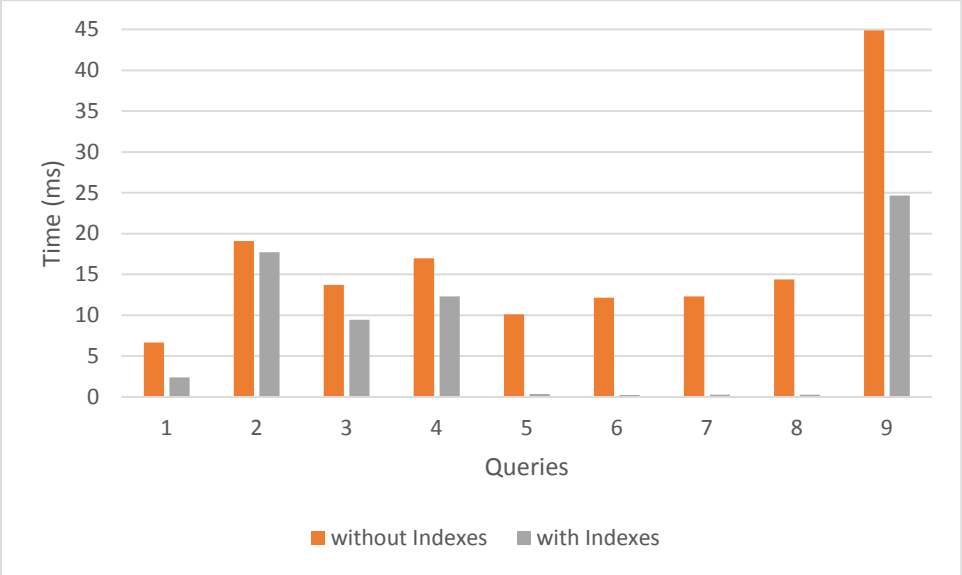
	QUERY PLAN text
1	Unique (cost=1572.77..1606.08 rows=3000 width=24) (actual time=23.679..24.367 rows=1684 loops=1)
2	-> Sort (cost=1572.77..1581.10 rows=3331 width=24) (actual time=23.675..23.789 rows=3592 loops=1)
3	Sort Key: products.product_name, products.product_type, products.product_price
4	Sort Method: quicksort Memory: 375kB
5	-> Hash Join (cost=612.64..1377.88 rows=3331 width=24) (actual time=4.254..11.027 rows=3592 loops=1)
6	Hash Cond: (order_details.product_id = products.product_id)
7	-> Nested Loop (cost=522.14..1241.58 rows=3331 width=4) (actual time=2.992..9.216 rows=3592 loops=1)
8	-> Hash Join (cost=521.85..707.51 rows=1066 width=4) (actual time=2.971..4.021 rows=1144 loops=1)
9	Hash Cond: (orders.customer_id = customers.customer_id)
10	-> Seq Scan on orders (cost=0.00..145.00 rows=8000 width=8) (actual time=0.006..0.755 rows=8000 loops=1)
11	-> Hash (cost=488.55..488.55 rows=2664 width=4) (actual time=2.921..2.921 rows=2664 loops=1)
12	Buckets: 4096 Batches: 1 Memory Usage: 126kB
13	-> Bitmap Heap Scan on customers (cost=59.59..488.55 rows=2664 width=4) (actual time=0.415..2.313 rows=2664 loops=1)
14	Recheck Cond: ((age >= 20) AND (age <= 30))
15	Heap Blocks: exact=388
16	-> Bitmap Index Scan on customers_age_idx (cost=0.00..58.93 rows=2664 width=0) (actual time=0.357..0.357 rows=2664 loops=1)
17	Index Cond: ((age >= 20) AND (age <= 30))
18	-> Index Scan using order_details_order_id_idx on order_details (cost=0.29..0.46 rows=4 width=8) (actual time=0.002..0.003 rows=3 loops=1144)
19	Index Cond: (order_id = orders.order_id)
20	-> Hash (cost=53.00..53.00 rows=3000 width=28) (actual time=1.240..1.240 rows=3000 loops=1)
21	Buckets: 4096 Batches: 1 Memory Usage: 218kB
22	-> Seq Scan on products (cost=0.00..53.00 rows=3000 width=28) (actual time=0.012..0.510 rows=3000 loops=1)
23	Planning time: 1.644 ms
24	Execution time: 24.660 ms



Αναλυτικοί πίνακες αποτελεσμάτων

Execution Time (ms)				Total
queries	without Indexes	with Indexes	Improvement	Improvement
1	6,666	2,385	64%	62%
2	19,076	17,736	7%	8%
3	13,704	9,442	31%	28%
4	16,963	12,313	27%	25%
5	10,116	0,335	97%	90%
6	12,147	0,24	98%	88%
7	12,294	0,266	98%	85%
8	14,403	0,279	98%	80%
9	44,891	24,66	45%	43%

Planning Time (ms)		
queries	without Indexes	with Indexes
1	0,124	0,197
2	0,368	0,237
3	0,41	0,672
4	0,416	0,638
5	0,428	0,764
6	0,909	1,336
7	1,149	1,788
8	2,08	3,05
9	1,239	1,644



Λίστα με τα συνημμένα αρχεία.

i. Ένα backup αρχείο της βάσης μας, χωρίς ευρετήρια (Online_Shop.backup).
ii. Ένα backup αρχείο της βάσης μας, με ευρετήρια (Online_Shop_Indexes.backup).
iii. Μία εικόνα με το διάγραμμα Οντοτήτων Συσχετίσεων (Online_Shop_ER).
iv. Μία png εικόνα με το Σχεσιακό Σχήμα (Online_Shop_RS).
v. Ένα αρχείο κειμένου με τον SQL κώδικα για την δημιουργία της βάσης μας (SQL Code).
vi. Ένα αρχείο κειμένου με όλα τα ερωτήματα (Queries).
vii. Ένα αρχείο Excel με τους πίνακες των αποτελεσμάτων (Time Results).
viii. Ένα αρχείο κειμένου με τον SQL κώδικα για την δημιουργία των ευρετηρίων (Indexes SQL Code).