Variations to the Relational Schema

* The relation names “User” and “Order” have been changed into “End\_User” and “Orders” respectively because the previous names are reserved keywords.
* The attribute name “text” of the relation “Review” and the attribute name “name” of the relations “Manage” and “Producer” have been changed into “content” and “region\_name” respectively because the previous names are reserved keyword.
* The domain of the attribute “images” of the relations “Sell” and “Restaurant” has been changed from bytea to text.
* The domain of the attribute “product\_code” of the relation “Product” has been changed from text to serial.
* The domain of the attribute “restaurant\_id” of the relation “Restaurant” has been changed from text to serial.
* The domain of the attribute “order\_id” of the relation “Orders” has been changed from text to serial.
* The domain of the attribute “event\_id” of the relation “Event” has been changed from text to serial.
* All the foreign keys related to the above attributes that have been changed their domain into serial have now an integer domain.
* An attribute “description” has been added to the relation “Category”, with domain text.

Figure 1 shows the updated relational schema based on the ahead variations.

Physical Schema

In the following the SQL instructions to build the database that is reported in Figure 1.

-----------------------

-- DATABASE CREATION --

-----------------------

CREATE DATABASE localProductions OWNER POSTGRES ENCODING = 'UTF8';

-- Connect to thelocals db to create data for its 'public' schema

\c localproductions

-- Setting the monetary locale for the current session

SET lc\_monetary to "it\_IT";

---------------------

-- DOMAIN CREATION --

---------------------

CREATE DOMAIN emailD AS text --TODO: check how to write email domains

NOT NULL;

COMMENT ON DOMAIN emailD IS 'alphanumeric emailD domain';

CREATE DOMAIN passwordD AS character varying(254)

CONSTRAINT properpassword CHECK (((VALUE)::text ~\* '[A-Za-z0-9.\_%-]{5,}'::text));

COMMENT ON DOMAIN passwordD IS 'alphanumeric passwordD domain, max 254 characters';

CREATE DOMAIN reviewScoreD AS SMALLINT

NOT NULL

CONSTRAINT review\_interval CHECK (VALUE >= 1 AND VALUE <= 5);

COMMENT ON DOMAIN reviewScoreD IS 'A review score can be an int between 1 and 5';

------------------------

-- DATA TYPE CREATION --

------------------------

CREATE TYPE role\_type AS ENUM('Restaurateur','Regional Manager','Event Organizer','Customer','Producer');

COMMENT ON TYPE role\_type IS 'enum for role types';

CREATE TYPE order\_type AS ENUM('Reserved','Completed','Canceled'); --Constaint 7

COMMENT ON TYPE order\_type IS 'enum for order types';

CREATE TYPE channel\_type AS ENUM('Pay In store','Cash On delivery');

COMMENT ON TYPE channel\_type IS 'enum for sales channel types';

--------------------

-- TABLE CREATION --

--------------------

CREATE TABLE Region(

name text,

PRIMARY KEY (name)

);

COMMENT ON TABLE Region IS 'Regions in which each producer operates';

CREATE TABLE Role(

role role\_type,

PRIMARY KEY (role)

);

COMMENT ON TABLE Role IS 'Role of each user';

CREATE TABLE Sales\_Channel(

type channel\_type,

PRIMARY KEY (type)

);

COMMENT ON TABLE Sales\_Channel IS 'Types of sales channels a producer provides';

CREATE TABLE Status(

status order\_type,

PRIMARY KEY (status)

);

COMMENT ON TABLE Status IS 'Status of an order';

CREATE TABLE Category(

category\_id VARCHAR(4),

name text NOT NULL,

description text,

PRIMARY KEY (category\_id)

);

COMMENT ON TABLE Category IS 'Category a product belongs to';

CREATE TABLE Product(

product\_code SERIAL,

name text NOT NULL,

general\_description text,

category\_id VARCHAR(4),

PRIMARY KEY (product\_code),

FOREIGN KEY (category\_id) REFERENCES Category(category\_id)

);

COMMENT ON TABLE Product IS 'Good sold by the producer within the catalogue';

CREATE TABLE Restaurant(

restaurant\_id SERIAL,

name text NOT NULL,

email emailD,

location text NOT NULL,

description text,

images text,

telephone\_number text NOT NULL,

region\_name text NOT NULL,

PRIMARY KEY (restaurant\_id),

FOREIGN KEY (region\_name) REFERENCES Region(name)

);

COMMENT ON TABLE Restaurant IS 'Restaurant in which menu there is at least one local product';

CREATE TABLE End\_User(

email emailD,

password passwordD NOT NULL,

first\_name text NOT NULL,

last\_name text NOT NULL,

validated boolean,

organization text,

role role\_type NOT NULL,

tax\_code text,

PRIMARY KEY (email),

CONSTRAINT constr\_validation CHECK(--Constaint 1

(role = 'Producer' AND validated IS NOT NULL) OR (role = 'Event Organizer' AND validated IS NOT NULL) OR

(role != 'Producer' AND validated IS NULL) OR (role != 'Event Organizer' AND validated IS NULL)

),

CONSTRAINT constr\_tax\_code CHECK(--Constaint 2

(role = 'Producer' AND tax\_code IS NOT NULL) OR (role = 'Restaurateur' AND tax\_code IS NOT NULL) OR

(role != 'Producer' AND tax\_code IS NULL) OR (role != 'Restaurateur' AND tax\_code IS NULL)

),

CONSTRAINT constr\_organization CHECK(--Constaint 3

(role = 'Event Organizer' AND organization IS NOT NULL) OR (role != 'Event Organizer' AND organization IS NULL)

)

);

COMMENT ON TABLE End\_User IS 'Every end user who has registered in the database';

CREATE TABLE Orders(

order\_id SERIAL,

total\_price money NOT NULL,

order\_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,

order\_status order\_type NOT NULL,

PRIMARY KEY (order\_id),

FOREIGN KEY (order\_status) REFERENCES Status(status)

);

COMMENT ON TABLE Orders IS 'Summary of an order';

CREATE TABLE Contain(

order\_id INT,

product\_code INT,

quantity INT NOT NULL,

price money NOT NULL,

PRIMARY KEY (order\_id, product\_code),

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id) ON DELETE CASCADE,

FOREIGN KEY (product\_code) REFERENCES Product(product\_code)

);

COMMENT ON TABLE Contain IS 'List of products contained in an order';

CREATE TABLE Producer(

email emailD,

pec emailD,

activity\_description text,

location text NOT NULL,

telephone\_number text NOT NULL,

business\_name text NOT NULL,

vat\_number text NOT NULL,

region\_name text NOT NULL,

PRIMARY KEY (email),

FOREIGN KEY (email) REFERENCES End\_User(email),

FOREIGN KEY (region\_name) REFERENCES Region(name)

);

COMMENT ON TABLE Producer IS 'local producer of foodstuff';

CREATE TABLE Review(

email emailD,

product\_code INT,

score reviewScoreD,

content text,

review\_timestamp TIMESTAMP WITH TIME ZONE NOT NULL,

PRIMARY KEY (email, product\_code),

FOREIGN KEY (email) REFERENCES End\_User(email),

FOREIGN KEY (product\_code) REFERENCES Product(product\_code)

);

COMMENT ON TABLE Review IS 'Review of a bought product';

CREATE TABLE Event(

event\_id SERIAL,

name text NOT NULL,

location text NOT NULL,

start\_date date NOT NULL,

end\_date date NOT NULL,

description text,

email emailD,

region\_name text NOT NULL,

PRIMARY KEY(event\_id),

FOREIGN KEY(email) REFERENCES End\_User(email),

FOREIGN KEY(region\_name) REFERENCES Region(name)

);

COMMENT ON TABLE Event IS 'Event in which at least one local product is promoted';

CREATE TABLE Manage(

email emailD,

region\_name text NOT NULL,

PRIMARY KEY (email),

FOREIGN KEY (email) REFERENCES End\_User(email),

FOREIGN KEY (region\_name) REFERENCES Region(name)

);

COMMENT ON TABLE Manage IS 'Set of regions which a user of role\_type "regional manager" is assigned to';

CREATE TABLE Offer(

restaurant\_id INT,

product\_code INT,

PRIMARY KEY (restaurant\_id, product\_code),

FOREIGN KEY (restaurant\_id) REFERENCES Restaurant(restaurant\_id),

FOREIGN KEY (product\_code) REFERENCES Product(product\_code)

);

COMMENT ON TABLE Offer IS 'List of local products that a restaurant offers in its menu';

CREATE TABLE Own(

restaurant\_id INT,

email emailD,

PRIMARY KEY (restaurant\_id,email),

FOREIGN KEY (restaurant\_id) REFERENCES Restaurant(restaurant\_id),

FOREIGN KEY (email) REFERENCES End\_User(email)

);

COMMENT ON TABLE Own IS 'Restaurant owned by an user of role\_type "restaurateur"';

CREATE TABLE Make(

order\_id INT,

type channel\_type,

customer\_email emailD,

producer\_email emailD,

PRIMARY KEY (order\_id, type, customer\_email, producer\_email),

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id) ON DELETE CASCADE,

FOREIGN KEY (type) REFERENCES Sales\_Channel(type),

FOREIGN KEY (customer\_email) REFERENCES End\_User(email),

FOREIGN KEY (producer\_email) REFERENCES Producer(email)

);

COMMENT ON TABLE Make IS 'List of orders made by users of role\_type "costumer" from "producers"';

CREATE TABLE Sell(

email emailD,

product\_code INT NOT NULL,

price money NOT NULL,

stock INT NOT NULL,

image text,

producer\_description text,

PRIMARY KEY (email, product\_code),

FOREIGN KEY (email) REFERENCES Producer(email),

FOREIGN KEY (product\_code) REFERENCES Product(product\_code)

);

COMMENT ON TABLE Sell IS 'List of products sold by each "producer"';

CREATE TABLE Promote(

email emailD,

product\_code INT,

event\_id INT,

PRIMARY KEY (email, product\_code, event\_id),

FOREIGN KEY (email) REFERENCES End\_User(email),

FOREIGN KEY (product\_code) REFERENCES Product(product\_code),

FOREIGN KEY (event\_id) REFERENCES Event(event\_id)

);

COMMENT ON TABLE Promote IS 'List of products sold by a "producer"';

CREATE TABLE Belong1(

email emailD,

category\_id VARCHAR(4),

PRIMARY KEY (email, category\_id),

FOREIGN KEY (email) REFERENCES End\_User(email),

FOREIGN KEY (category\_id) REFERENCES Category(category\_id)

);

COMMENT ON TABLE Belong1 IS 'List of categories a "producer" belongs to';

CREATE TABLE Sale\_Through(

type channel\_type,

email emailD,

PRIMARY KEY (type, email),

FOREIGN KEY (type) REFERENCES Sales\_Channel(type),

FOREIGN KEY (email) REFERENCES Producer(email)

);

COMMENT ON TABLE Sale\_Through IS 'List of sales channels provided by each "producer"';

Trigger Function and Stored Procedure

The following triggers implement the external constraints 4, 6, 8, 10 and 11 defined in the previous homework. Constraint 5 is meant to be implemented at application level. The other constraints are implemented directly in the physical schema.

-- Connect to thelocals db to create data for its 'public' schema

\c localproductions

-----------------------

-- TRIGGERS CREATION --

-----------------------

--Constraint 4

CREATE FUNCTION category\_check() RETURNS TRIGGER AS $$

DECLARE

cat\_id text;

BEGIN

SELECT p.category\_id INTO cat\_id

FROM Product AS p

WHERE p.product\_code = NEW.product\_code;

PERFORM \* FROM Belong1 AS b

WHERE b.email = NEW.email AND b.category\_id = cat\_id;

IF NOT FOUND THEN -- if the query found 0 rows

RAISE EXCEPTION 'The product category is not associated to the producer';

END IF;

RETURN NEW; -- proceed to the insert

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER sell\_check BEFORE INSERT -- Constraint 4

ON Sell

FOR EACH ROW

EXECUTE PROCEDURE category\_check();

-- Constraint 6 and 10

CREATE FUNCTION cancel\_order(id INT) RETURNS void AS $$

BEGIN

DELETE FROM Orders

WHERE Orders.order\_id = id;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION decrease\_stock(p\_email text, qnt\_to\_decrease INT, p\_code INT) RETURNS void AS $$

BEGIN

UPDATE Sell

SET stock = stock - qnt\_to\_decrease

WHERE email = p\_email AND product\_code = p\_code;

END;

$$ LANGUAGE plpgsql;

CREATE FUNCTION quantity\_check() RETURNS TRIGGER AS $$

DECLARE

mystock INT;

p\_email text;

BEGIN

SELECT producer\_email INTO p\_email

FROM Make

WHERE order\_id = NEW.order\_id;

SELECT stock INTO mystock

FROM Sell

WHERE product\_code = NEW.product\_code AND email = p\_email;

-- If the quantity selected is greater than the available product cancel the order instance and all instances that reference to i

IF NEW.quantity > mystock THEN

PERFORM cancel\_order(NEW.order\_id);

RAISE EXCEPTION 'The quantity selected cannot be purchased. Please select a lower quantity';

ELSE

PERFORM decrease\_stock(p\_email, NEW.quantity, NEW.product\_code);

END IF;

RETURN NEW; -- proceed to the insert

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER contain\_check BEFORE INSERT

ON Contain

FOR EACH ROW

EXECUTE PROCEDURE quantity\_check();

--Constraint 8

CREATE FUNCTION promote\_check() RETURNS TRIGGER AS $$

BEGIN

PERFORM \* FROM Sell

WHERE email = NEW.email AND product\_code = NEW.product\_code;

IF NOT FOUND THEN -- if the query found 0 rows

RAISE EXCEPTION 'The producer does not sell that product';

END IF;

RETURN NEW; -- proceed to the insert

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER event\_promote\_check BEFORE INSERT

ON Promote

FOR EACH ROW

EXECUTE PROCEDURE promote\_check();

--Constraint 11

CREATE FUNCTION order\_status\_check() RETURNS TRIGGER AS $$

DECLARE

qnt\_to\_increase INT;

p\_email text;

p\_code INT;

BEGIN

IF NEW.status = "Canceled" AND OLD.status = "Reserved" THEN

SELECT product\_code, quantity INTO p\_code, qnt\_to\_increase

FROM Contain

WHERE order\_id = NEW.order\_id;

SELECT producer\_email INTO p\_email

FROM Make

WHERE order\_id = NEW.order\_id;

UPDATE Sell

SET stock = stock + qnt\_to\_increase

WHERE producer\_email = p\_email AND product\_code = p\_code;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER order\_status\_canceled AFTER UPDATE

ON Orders

FOR EACH ROW

EXECUTE PROCEDURE order\_status\_check();

Populate the Database: Example

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lacinia scelerisque euismod. Quisque rhoncus feugiat nisl. Pellentesque finibus feugiat tellus. Praesent in sodales lorem, et aliquam elit. Vivamus bibendum in mauris sodales lacinia. Donec et elementum justo. Praesent eget mauris consectetur, luctus massa vel, maximus ipsum. Sed quis quam sit amet magna congue tincidunt. Vivamus vel tortor eros. Nulla massa lacus, maximus quis orci quis, tristique cursus nisl.

Principal Queries

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lacinia scelerisque euismod. Quisque rhoncus feugiat nisl. Pellentesque finibus feugiat tellus. Praesent in sodales lorem, et aliquam elit. Vivamus bibendum in mauris sodales lacinia. Donec et elementum justo. Praesent eget mauris consectetur, luctus massa vel, maximus ipsum. Sed quis quam sit amet magna congue tincidunt. Vivamus vel tortor eros. Nulla massa lacus, maximus quis orci quis, tristique cursus nisl.

JDBC Implementations of the Principal Queries and Visualization

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lacinia scelerisque euismod. Quisque rhoncus feugiat nisl. Pellentesque finibus feugiat tellus. Praesent in sodales lorem, et aliquam elit. Vivamus bibendum in mauris sodales lacinia. Donec et elementum justo. Praesent eget mauris consectetur, luctus massa vel, maximus ipsum. Sed quis quam sit amet magna congue tincidunt. Vivamus vel tortor eros. Nulla massa lacus, maximus quis orci quis, tristique cursus nisl.

TO DO:

1. copiare il relational schema modificato
2. copiare codice
3. inserire sezione trigger

Domande

1. Trigger
2. Serial
3. Alcuni constraints a livello applicativo o azioni del database(4,5,6,11)
4. Enum per categorie gia’ fissate
5. Usare tipo bytea o riferimento url per le immagini (e come popolare per il bytea), come gestire piu’ immagini se usiamo url.
6. JDBC cosa deve contenere,se le stesse query che abbiamo concepito o no.
7. Dove va caricato il database? localhost o spazio sul server postgresql DEI?