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.
* The attribute “organization\_name” of the relation “End\_User” has been changed into “organization”.
* The attribute “status” of the relation “Orders” has been changed into “order\_status”.

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

Immagine che contiene testo

Descrizione generata automaticamente

Figure 1: Updated Relational Schema

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

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: Each producer can sell products that belong to the same categories he is associated with.

This stored procedure check if the inserted category\_id of the product matches one of the category\_id of the producer.

\*/

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();

/\*

Constraints 6 and 10:

Constraints 6 ensure that the quantity attribute of an order must be less or equal to the stock value of the relative products.

Constraints 10 ensure that when a order is correctly submited the stock attribute is updated

\*/

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: the promote relation have to be consistent, the same product\_code - email pair must exist also in the sell relation

\*/

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: when an order status is update from "Reserved" to "Canceled", the products stocks are restored

\*/

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

In the following, there are some examples of SQL instructions to insert data within each relation present in the relational schema.

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

\c localproductions

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

-- INSERT OPERATIONS --

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

-- Region Relation

INSERT INTO Region(name) VALUES

('Piemonte'),

('Valle d''Aosta'),

('Lombardia'),

('Trentino-Alto Adige'),

('Veneto'),

('Friuli-Venezia Giulia'),

('Liguria'),

('Emilia-Romagna'),

('Toscana'),

('Umbria'),

('Marche'),

('Lazio'),

('Abruzzo'),

('Molise'),

('Campania'),

('Puglia'),

('Basilicata'),

('Calabria'),

('Sicilia'),

('Sardegna');

-- Role Relation

INSERT INTO Role(role) VALUES

('Restaurateur'),

('Regional Manager'),

('Event Organizer'),

('Customer'),

('Producer');

-- Sales Channel Relation

INSERT INTO Sales\_Channel(type) VALUES

('Pay In store'),

('Cash On delivery');

-- Satus Relation

INSERT INTO Status(status) VALUES

('Reserved'),

('Completed'),

('Canceled');

-- Category Relation

INSERT INTO Category(category\_id, name, description) VALUES

('A1', 'Carne Rossa', 'Lavorazione di carni di ungulati domestici o selvatici per produzione e vendita di prodotti a base di carne'),

('A2', 'Carne Bianca', 'Macellazione e vendita carni di volatili da cortile, conigli, piccola selvaggina allevata o selvatica.'),

('A3', 'Miele', 'Produzione e vendita di MIELE, prodotti dolciari a base di miele con frutta secca o propoli; pappa reale o gelatina reale; polline; idromele; aceto di miele'),

('A4', 'Conserve', 'Produzione e vendita di CONSERVE ALIMENTARI VEGETALI, VEGETALI TOSTATI, VEGETALI ESSICCATI E FARINE, CONFETTURE, MARMELLATE, COMPOSTE, SCIROPPI E SUCCHI DI FRUTTA, VEGETALI FRESCHI ED ERBE ALIMURGICHE'),

('A5', 'Pane', 'Produzione, cottura e vendita di PANE e PRODOTTI DA FORNO'),

('A6', 'Olii', 'Produzione e vendita di OLIO EXTRAVERGINE DI OLIVA E OLIVE'),

('A7', 'Latte e derivati', 'Produzione, lavorazione e vendita di LATTE CRUDO, LATTE TRATTATO TERMICAMENTE E PRODOTTI LATTIERO CASEARI DI MALGA E DI PICCOLI CASEIFICI AZIENDALI'),

('A8', 'Chiocciole', 'Produzione, lavorazione e vendita di CHIOCCIOLE'),

('A9', 'Pesca e pescati', 'Produzione, lavorazione e vendita di PRODOTTI DELLA PESCA, DELL"ACQUACOLTURA E PRODOTTI TRASFORMATI'),

('A10', 'Pasta Secca','Produzione, lavorazione e vendita di: PASTA SECCA '),

('A11', 'Birra','Produzione, lavorazione e vendita di: BIRRA '),

('A12', 'Aceti','Produzione, lavorazione e vendita di: ACETI');

-- Product Relation

INSERT INTO Product(name, general\_description, category\_id) VALUES

('Sopressa Vicentina D.O.P.', 'La Soppressa Vicentina √® ottenuta dalla lavorazione di cosce, coppa, spalla, pancetta, grasso di gola e lombo di maiale.', 'A1'),

('Coppa di Testa di Este', 'La coppa di testa viene prodotta da tempo immemorabile dai contadini dell"estense, nel periodo invernale, immediatamente dopo la macellazione del maiale.', 'A1'),

('Gallina Padovana', 'Razza riconosciuta come "pura" da uno standard nazionale redatto dalla Federazione Italiana delle razze avicole.', 'A2'),

('Coniglio Veneto', 'Anticamente l"allevamento del coniglio costituiva la forma di reddito integrativo per le famiglie della mezzadria veneta.', 'A2'),

('Miele delle Dolomiti Bellunesi D.O.P.', 'Miele prodotto a partire dal nettare dei fiori del territorio montano bellunese, dall"ecotipo locale di ‚ÄúApis mellifera.‚Äù', 'A3'),

('Confettura di Mirtillo e Mela della Altopiano di Asiago', 'Esistono numerosi tipi di marmellate e confetture tipiche delle diverse zone del Veneto dovute alla ricca produzione di frutta.', 'A4'),

('Confettura di Ciliegia di Marostica I.G.P.', 'Esistono numerosi tipi di marmellate e confetture tipiche delle diverse zone del Veneto dovute alla ricca produzione di frutta.', 'A4'),

('Panada Veneta', 'Pancotto aromatizzato alla cannella.', 'A5'),

('Sopa Coada', 'Zuppa gratinata composta di strati di pane raffermo e piccione', 'A5'),

('Olio Extravergine Di Oliva D.O.P.', 'Olio tipico dei Colli Euganei.', 'A6'),

('Asiago D.O.P.', 'Formaggio tipico delle zone limitrofe all"Altopiano di Asiago.', 'A7'),

('Chiocciole D.O.P. Vicentine', 'Chiocciole tipiche della pianura vicino a Vicenza.', 'A8'),

('Calamari Veneziani', 'Pesce tipico della laguna di Venezia.', 'A9'),

('Bigoli Veneti', 'Ottima pasta da abbinare con il rag√π d''anatra.', 'A10'),

('Birra di Rovigo D.O.P.', 'La birra artigianale √® un prodotto non pastorizzato e non filtrato.', 'A11'),

('Aceto di Vino Euganeo', 'Aceto tipico dei Colli Euganei.', 'A12');

-- Restaurant Relation

INSERT INTO Restaurant(name, email, location, description, images, telephone\_number, region\_name) VALUES

('Le calandre', 'lecalandre@gmail.com', 'Padova', 'Traditional Italian restaurant', 'https://bit.ly/2LW54GS', '0490000', 'Veneto'),

('Da Orazio', 'daorazio@gmail.com', 'Treviso', 'Seafood restaurant', 'https://bit.ly/2QexFpo', '04221000', 'Veneto'),

('Da Luisa', 'daluisa@gmail.com', 'Venezia', 'Trattoria', 'https://bit.ly/2Eojo4P', '346013406', 'Veneto'),

('Da Pino', 'dapino@gmail.com', 'Verona', 'Pizzeria', 'https://bit.ly/2YE0sGX', '346713406', 'Veneto');

-- End User Relation

INSERT INTO End\_User(email, password, first\_name, last\_name, validated, organization, role, tax\_code) VALUES

('Giovanni.Aquila@gmail.com', md5('12345'), 'Giovanni', 'Aquila', NULL, NULL, 'Restaurateur', 'QLAGNN80P11G273B'),

('Orazio.Gatti@gmail.com', md5('12345'), 'Orazio', 'Gatti', NULL, NULL, 'Restaurateur', 'GTTRZO75P08D612O'),

('Luisa.Ferrara@gmail.com', md5('12345'), 'Luisa', 'Ferrara', NULL, NULL, 'Restaurateur', 'FRRLSU80M46G482J'),

('Enzo.Tumicelli@gmail.com', md5('12345'), 'Enzo', 'Tumicelli', NULL, NULL, 'Restaurateur', 'TMCNZE80H12G478Y'),

('Salvatore.Aloia@gmail.com', md5('12345'), 'Salvatore', 'Aloia', NULL, NULL, 'Regional Manager', NULL),

('Ginevra.Barsotti@gmail.com', md5('12345'), 'Ginevra', 'Barsotti', NULL, NULL, 'Regional Manager', NULL),

('Uberto.Innocenti@gmail.com', md5('12345'), 'Uberto', 'Innocenti', NULL, NULL, 'Regional Manager', NULL),

('Ambrogio.Sparacello@gmail.com', md5('12345'), 'Ambrogio', 'Sparacello', NULL, NULL, 'Regional Manager', NULL),

('Settimo.Albanesi@gmail.com', md5('12345'), 'Settimo', 'Albanesi', TRUE, 'Pro Loco Padova', 'Event Organizer', NULL),

('Antonio.Como@gmail.com', md5('12345'), 'Antonio', 'Como', TRUE, 'Antichi Sapori', 'Event Organizer', NULL),

('Michelina.Corti@gmail.com', md5('12345'), 'Michelina', 'Corti', TRUE, 'Chilometro zero', 'Event Organizer', NULL),

('Fioralba.Murgia@gmail.com', md5('12345'), 'Fioralba', 'Murgia', TRUE, 'Coldiretti', 'Event Organizer', NULL),

('Evelina.Piazza@gmail.com', md5('12345'), 'Evelina', 'Piazza', NULL, NULL, 'Customer', NULL),

('Gianpaolo.Abano@gmail.com', md5('12345'), 'Gianpaolo', 'Abano', NULL, NULL, 'Customer', NULL),

('Gualtiero.Aldebrandi@gmail.com', md5('12345'), 'Gualtiero', 'Aldebrandi', NULL, NULL, 'Customer', NULL),

('Nicola.Abelli@gmail.com', md5('12345'), 'Nicola', 'Abelli', NULL, NULL, 'Customer', NULL),

('Tatiana.Agnelli@gmail.com', md5('12345'), 'Tatiana', 'Agnelli ', TRUE, NULL, 'Producer', 'GNLTTN80D45D969Q'),

('Angelo.Antonini@gmail.com', md5('12345'), 'Angelo', 'Antonini', TRUE, NULL, 'Producer', 'NTNNGL80H08L219J'),

('Gualberto.Alescio@gmail.com', md5('12345'), 'Gualberto', 'Alescio', TRUE, NULL, 'Producer', 'LSCGBR80P11H501P'),

('Beatrice.Altoviti@gmail.com', md5('12345'), 'Beatrice', 'Altoviti', TRUE, NULL, 'Producer', 'LTVBRC80M48G224W');

-- Orders Relation

INSERT INTO Orders(total\_price, order\_timestamp, order\_status) VALUES

('20,20', '2019-03-24 14:13:25+02', 'Completed'),

('2,50', '2019-03-29 12:05:09+02', 'Completed'),

('43,75', '2019-04-5 17:47:35+02', 'Completed'),

('3,60', '2019-04-7 14:13:25+02', 'Canceled'),

('13,10', '2019-04-13 10:22:13+02', 'Completed'),

('12,00', '2019-04-19 18:44:30+02', 'Canceled'),

('8,05', '2019-04-22 12:15:00+02', 'Completed'),

('12,05', '2019-04-27 20:37:55+02', 'Completed'),

('12,35', '2019-05-01 19:18:57+02', 'Completed'),

('14,20', '2019-05-03 13:33:47+02', 'Canceled');

-- Contain Relation

INSERT INTO Contain(order\_id, product\_code, quantity, price) VALUES

(1, 1, 2, '12,05'),

(1, 5, 1, '8,15'),

(2, 7, 1, '2,50'),

(3, 3, 2, '12,00'),

(3, 11, 3, '6,75'),

(3, 12, 1, '25,00'),

(4, 6, 1, '3,60'),

(5, 8, 3, '1,05'),

(5, 1, 3, '12,05'),

(6, 3, 1, '12,00'),

(7, 5, 1, '8,05'),

(8, 12, 1, '12,05'),

(9, 1, 2, '12,35'),

(10, 10, 1, '14,20');

-- Producer Relation

INSERT INTO Producer(email, pec, activity\_description , location, telephone\_number , business\_name , vat\_number, region\_name ) VALUES

('Tatiana.Agnelli@gmail.com', 'Tatiana.Agnelli@legalmail.it', 'Allevamento di maiali e produzione propria di salumi', 'Conegliano', '3923085842', 'Le carni di Tatiana', '01906530983', 'Veneto'),

('Angelo.Antonini@gmail.com', 'Angelo.Antonini@legalmail.it', 'Produzione di olio dei Colli Euganei', 'Cinto Euganeo', '0429634030', 'Sapori dei Colli', '01835500940', 'Veneto'),

('Gualberto.Alescio@gmail.com', 'Gualberto.Alescio@legalmail.it', 'Piccolo allevamento di conigli', 'Padova', '0496588741', 'Allevamento Alescio', '02976538413', 'Veneto'),

('Beatrice.Altoviti@gmail.com', 'Beatrice.Altoviti@legalmail.it', 'Produzione di confetture', 'Asiago', '0424461475', 'Le confetture di Bea', '01984568450', 'Veneto');

-- Review Relation

INSERT INTO Review(email, product\_code, score, content, review\_timestamp) VALUES

('Evelina.Piazza@gmail.com', 1, 5, 'Sopressa di ottima fattura, con un buon rapporto qualit√†/prezzo.', '2019-05-10 11:23:54+02'),

('Nicola.Abelli@gmail.com', 5, 4, 'Miele molto buono, forse un po'' troppo caro.', '2019-05-13 10:23:54+02'),

('Gianpaolo.Abano@gmail.com', 7, 3, 'Marmellata un po'' troppo dolce, accettabile per il prezzo.', '2019-05-15 12:23:54+02'),

('Evelina.Piazza@gmail.com', 3, 5, 'Ottima qualit√† di carne, riacquister√≤ sicuramente in futuro.', '2019-05-16 10:43:54+02'),

('Evelina.Piazza@gmail.com', 11, 5, 'Asiago perfetto.', '2019-05-18 10:23:54+02'),

('Gualtiero.Aldebrandi@gmail.com', 12, 1, 'Chiocciole arrivare troppo vecchie, le ho buttate.', '2019-05-19 09:29:54+02');

-- Event Relation

INSERT INTO Event(name, location, start\_date, end\_date, description, email, region\_name) VALUES

('Festa di Primavera', 'Via Casoni, 31057 Casale sul Sile (TV)', '2019-07-20', '2019-07-23', 'Musica, vino e artigianato, tutti i giorni dalle 20 alle 23', 'Antonio.Como@gmail.com', 'Veneto'),

('In Vino Veritas', 'Prato della Valle, 35123 Padova (PD)', '2019-09-15', '2019-09-20', 'Gustate i vini dei colli, tutti i giorni dalle 10 alle 23', 'Settimo.Albanesi@gmail.com', 'Veneto'),

('Orto in Tavola', 'Piazza IV Novembre, 06100 Perugia (PG)', '2019-06-10', '2019-06-20', 'Le migliori produzioni locali in piazza, tutti i giorni dalle 10 alle 22', 'Fioralba.Murgia@gmail.com', 'Umbria'),

('Festa della tagliata', 'Piazza della Biade, 36100 Vicenza (VI)', '2019-07-01', '2019-07-05', 'Carne di qualit√† tutte le sere dalle 19 alle 00', 'Antonio.Como@gmail.com', 'Veneto'),

('Festa del Pane', 'Piazza del Duomo, 56126 Pisa (PI)', '2019-10-25', '2019-10-30', 'Cibo e musica tutte le sere dalle 19 alle 23', 'Michelina.Corti@gmail.com', 'Toscana');

-- Manage Relation

INSERT INTO Manage(email, region\_name) VALUES

('Salvatore.Aloia@gmail.com', 'Veneto'),

('Ginevra.Barsotti@gmail.com', 'Piemonte'),

('Uberto.Innocenti@gmail.com', 'Trentino-Alto Adige'),

('Ambrogio.Sparacello@gmail.com', 'Friuli-Venezia Giulia');

-- Offer Relation

INSERT INTO Offer(restaurant\_id, product\_code) VALUES

(1, 3),

(1, 4),

(2, 13),

(2, 15),

(3, 1),

(4, 10);

-- Own Relation

INSERT INTO Own(restaurant\_id, email) VALUES

(1, 'Giovanni.Aquila@gmail.com'),

(2, 'Orazio.Gatti@gmail.com'),

(3, 'Luisa.Ferrara@gmail.com'),

(4, 'Enzo.Tumicelli@gmail.com');

-- Make Relation

INSERT INTO Make(order\_id, type, customer\_email, producer\_email) VALUES

(1, 'Cash On delivery', 'Evelina.Piazza@gmail.com', 'Angelo.Antonini@gmail.com'),

(2, 'Pay In store', 'Gianpaolo.Abano@gmail.com', 'Tatiana.Agnelli@gmail.com'),

(3, 'Cash On delivery', 'Evelina.Piazza@gmail.com', 'Angelo.Antonini@gmail.com'),

(4, 'Pay In store', 'Gianpaolo.Abano@gmail.com', 'Gualberto.Alescio@gmail.com'),

(5, 'Pay In store', 'Gualtiero.Aldebrandi@gmail.com', 'Angelo.Antonini@gmail.com'),

(6, 'Pay In store', 'Gianpaolo.Abano@gmail.com', 'Beatrice.Altoviti@gmail.com'),

(7, 'Pay In store', 'Nicola.Abelli@gmail.com', 'Gualberto.Alescio@gmail.com'),

(8, 'Cash On delivery', 'Gualtiero.Aldebrandi@gmail.com', 'Tatiana.Agnelli@gmail.com'),

(9, 'Pay In store', 'Evelina.Piazza@gmail.com', 'Tatiana.Agnelli@gmail.com'),

(10, 'Cash On delivery', 'Gianpaolo.Abano@gmail.com', 'Angelo.Antonini@gmail.com');

-- Belong1 Relation

INSERT INTO Belong1(email, category\_id ) VALUES

('Tatiana.Agnelli@gmail.com','A1'),

('Angelo.Antonini@gmail.com','A6'),

('Gualberto.Alescio@gmail.com','A2'),

('Beatrice.Altoviti@gmail.com','A4');

-- Sell Relation

INSERT INTO Sell(email , product\_code , price, stock, image, producer\_description) VALUES

('Tatiana.Agnelli@gmail.com', 1, '12,05', 7, NULL, 'Soppressa Vicentina D.O.P di coppa e spalla aromatizata con rosmarino. Prezzo indicato per 800g di prodotto.'),

('Tatiana.Agnelli@gmail.com', 2, '8,10', 10, NULL, 'Cappa di testa tradizionale estense aromatizzata al timo. Prezzo indicato per 900g di prodotto.'),

('Angelo.Antonini@gmail.com', 10, '14,20', 5, NULL, 'Olio dei Colli Euganei spremuto a freddo,dal retrogusto piccante. Prezzo indicato per 1L di prodotto.'),

('Gualberto.Alescio@gmail.com', 4, '12,30', 37, NULL, 'Coniglio intero allevato all"aperto. Prezzo indicato per 1800g di prodotto.'),

('Beatrice.Altoviti@gmail.com', 6, '3,60', 40, NULL, 'Confettura di Mirtillo e Mela di coltivazioni secolari presenti nell" Altopiano. Prezzo indicato per 600g di prodotto.'),

('Beatrice.Altoviti@gmail.com', 7, '2,50', 38, NULL, 'Deliziosa confettura della rinomata Ciliegia di Marostica,famosa per il suo gusto caramelloso. Prezzo indicato per 450g di prodotto.');

-- Promote Relation

INSERT INTO Promote(email, product\_code, event\_id) VALUES

('Beatrice.Altoviti@gmail.com', 6, 1),

('Beatrice.Altoviti@gmail.com', 7, 1),

('Beatrice.Altoviti@gmail.com', 6, 2),

('Gualberto.Alescio@gmail.com', 4, 2),

('Angelo.Antonini@gmail.com', 10, 2),

('Tatiana.Agnelli@gmail.com', 2, 4),

('Gualberto.Alescio@gmail.com', 4, 4);

-- Sale Through Relation

INSERT INTO Sale\_Through(type, email) VALUES

('Pay In store', 'Tatiana.Agnelli@gmail.com'),

('Cash On delivery', 'Tatiana.Agnelli@gmail.com'),

('Pay In store', 'Angelo.Antonini@gmail.com'),

('Cash On delivery', 'Angelo.Antonini@gmail.com'),

('Pay In store', 'Gualberto.Alescio@gmail.com'),

('Pay In store', 'Beatrice.Altoviti@gmail.com');

Principal Queries

In this section, we report the four most frequent queries to navigate the database:

1. Visualize the order history of a customer (last 15 orders);
2. Visualize the details of an order given an order ID;
3. Visualize present and future events in a region
4. Visualize sell statistics for a given producer

-- connect to localProductions db

\c localProductions

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

-- QUERIES --

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

-- 1: Query to visualize Order History (last 15) of a customer

SELECT o.order\_id, o.order\_timestamp, o.total\_price, m.type FROM Make AS m

INNER JOIN Orders AS o ON m.order\_id = o.order\_id

WHERE customer\_email = 'Evelina.Piazza@gmail.com'

ORDER BY order\_id DESC

LIMIT 15;



-- 2: Query to visualize the details of an order given an order ID

SELECT c.order\_id, p.product\_code, name, quantity, price AS "Unit Price", business\_name FROM Contain as c

INNER JOIN Make AS m On c.order\_id = m.order\_id

INNER JOIN Product AS p ON c.product\_code = p.product\_code

INNER JOIN Producer AS prdcr ON m.producer\_email = prdcr.email

WHERE c.order\_id = 3 AND m.customer\_email = 'Evelina.Piazza@gmail.com'

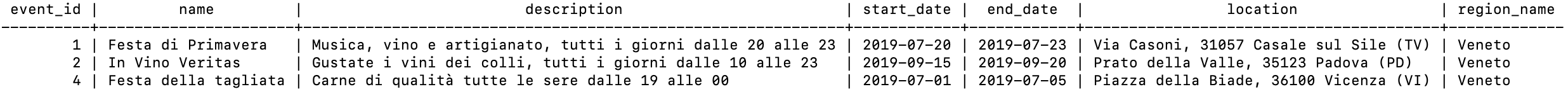
ORDER BY name ASC;



-- 3: Query to visualize present and future events in a region

SELECT e.event\_id, e.name, description, start\_date, end\_date, location, region\_name FROM Event AS e

WHERE end\_date >= CURRENT\_DATE AND region\_name = 'Veneto';



-- STATISTICS

-- 4: Query to visualize sell statistics for a given producer (email)

SELECT c.product\_code, p.name, SUM(c.quantity) AS "Total Sell" FROM Make AS m

INNER JOIN Orders AS o ON m.order\_id = o.order\_id

INNER JOIN Contain AS c ON m.order\_id = c.order\_id

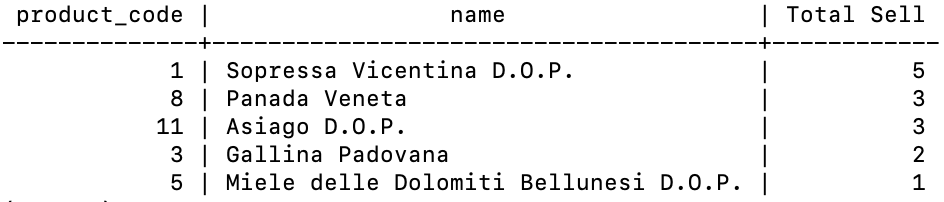
INNER JOIN Product AS p ON p.product\_code = c.product\_code

WHERE m.producer\_email = 'Angelo.Antonini@gmail.com' AND o.order\_status = 'Completed'

GROUP BY c.product\_code, p.name

ORDER BY "Total Sell" DESC

LIMIT 5;



JDBC Implementations of the Principal Queries and Visualization

Hereafter, we report a java class that simulates a user login and performs the principal queries.

import javax.xml.bind.DatatypeConverter;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.sql.\*;

import java.util.Scanner;

/\*\*

\* Simulate typical user (Customer and Producer) actions, through a textual interface

\*/

public class UserActions {

/\*\*

\* The JDBC driver to be used

\*/

private static final String DRIVER = "org.postgresql.Driver";

/\*\*

\* The URL of the database to be accessed

\*/

private static final String DATABASE = "jdbc:postgresql://localhost/localproductions";

/\*\*

\* The username for accessing the database

\*/

private static final String USER = "jarvis";

/\*\*

\* The password for accessing the database

\*/

private static final String PASSWORD = "";

/\*\*

\* The statement to list the last 15 orders done by a customer

\*/

private static final String LIST\_ORDERS = "SELECT o.order\_id, o.order\_timestamp, o.total\_price, m.type FROM Make AS m "

+ "INNER JOIN Orders AS o ON m.order\_id = o.order\_id "

+ "WHERE customer\_email = ? "

+ "ORDER BY order\_id DESC "

+ "LIMIT 15; ";

/\*\*

\* The statement to list the last 15 orders received by a producer

\*/

private static final String LIST\_PRODUCER\_ORDERS = "SELECT o.order\_id, o.order\_timestamp, o.total\_price, m.type FROM Make AS m "

+ "INNER JOIN Orders AS o ON m.order\_id = o.order\_id "

+ "WHERE producer\_email = ? "

+ "ORDER BY order\_id DESC "

+ "LIMIT 15; ";

/\*\*

\* The statement to list the details of an customer order

\*/

private static final String LIST\_ORDER\_DETAIL = "SELECT c.order\_id, p.product\_code, name, quantity, price AS \"Unit Price\", business\_name FROM Contain as c "

+ "INNER JOIN Make AS m On c.order\_id = m.order\_id "

+ "INNER JOIN Product AS p ON c.product\_code = p.product\_code "

+ "INNER JOIN Producer AS prdcr ON m.producer\_email = prdcr.email "

+ "WHERE c.order\_id::varchar = ? AND m.customer\_email = ? "

+ "ORDER BY name ASC; ";

/\*\*

\* The statement to list the details of an producer order

\*/

private static final String LIST\_PRODUCER\_ORDER\_DETAIL = "SELECT c.order\_id, p.product\_code, name, quantity, price AS \"Unit Price\", business\_name FROM Contain as c "

+ "INNER JOIN Make AS m On c.order\_id = m.order\_id "

+ "INNER JOIN Product AS p ON c.product\_code = p.product\_code "

+ "INNER JOIN Producer AS prdcr ON m.producer\_email = prdcr.email "

+ "WHERE c.order\_id::varchar = ? AND m.producer\_email = ? "

+ "ORDER BY name ASC; ";

/\*\*

\* The statement to list the future events that will take place in a region

\*/

private static final String LIST\_EVENTS = "SELECT DISTINCT e.event\_id, e.name, description, start\_date, end\_date, location, region\_name FROM Event AS e "

+ "WHERE end\_date >= CURRENT\_DATE AND region\_name = ?; ";

/\*\*

\* The statement to list the 5 most sold product by a producer

\*/

private static final String LIST\_PRODUCER\_STATS = "SELECT c.product\_code, p.name, SUM(c.quantity) AS \"Total Sell\" FROM Make AS m "

+ "INNER JOIN Orders AS o ON m.order\_id = o.order\_id "

+ "INNER JOIN Contain AS c ON m.order\_id = c.order\_id "

+ "INNER JOIN Product AS p ON p.product\_code = c.product\_code "

+ "WHERE m.producer\_email = ? AND o.order\_status = 'Completed' "

+ "GROUP BY c.product\_code, p.name "

+ "ORDER BY \"Total Sell\" DESC "

+ "LIMIT 5; ";

/\*\*

\* The statement to retrieve an End User by email

\*/

private static final String EMAIL\_CHECK = "SELECT \* FROM End\_User WHERE email = ?;";

/\*\*

\* The statement to select the End User password by email

\*/

private static final String SELECT\_PSW = "SELECT password FROM End\_User WHERE email = ?;";

/\*\*

\* The statement to list the End User rose by email

\*/

private static final String ROLE = "SELECT role FROM End\_User WHERE email = ?;";

/\*\*

\* The connection to the database

\*/

private Connection con;

/\*\*

\* The scanner to use to get data from the user

\*/

private Scanner scan;

/\*\*

\* The logged in user mail

\*/

private String email;

/\*\*

\* The logged in user password (md5)

\*/

private String password;

/\*\*

\* The method for printing the user orders

\*/

private void printCustomerOrders() throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

try{

pstmt = con.prepareStatement(LIST\_ORDERS);

pstmt.setString(1, email);

rs = pstmt.executeQuery();

System.out.println("Order ID\tOrder Timestamp\tTotal Price\tPayment Method");

while (rs.next()) {

System.out.printf("%s\t%s\t%s\t%s\n",

rs.getString("order\_id"),

rs.getString("order\_timestamp"),

rs.getString("total\_price"),

rs.getString("type"));

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method that prints the details of the order made by a customer selected by id

\* @param order\_id the id of the order to print

\*/

private void printCustomerOrderDetail(String order\_id) throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

try{

pstmt = con.prepareStatement(LIST\_ORDER\_DETAIL);

pstmt.setString(1, order\_id);

pstmt.setString(2, email);

rs = pstmt.executeQuery();

if (!rs.isBeforeFirst()){

System.out.println("You haven't any order with the provided ID, please retry.");

} else {

System.out.println("Order ID\tProduct Name\tProduct Quantity\tUnit Price\tProducer");

}

while (rs.next()) {

System.out.printf("%s\t%s\t%s\t%s\t%s\n",

rs.getString("order\_id"),

rs.getString("name"),

rs.getString("quantity"),

rs.getString("Unit Price"),

rs.getString("business\_name"));

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method for printing the current or future events that take place in a region

\*/

private void printEvents(String Region) throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

try{

pstmt = con.prepareStatement(LIST\_EVENTS);

pstmt.setString(1, Region);

rs = pstmt.executeQuery();

if (!rs.isBeforeFirst()){

System.out.println("There aren't scheduled events in "+Region+". :'(");

} else {

System.out.println("Event Name\tDescription\tStart Date\tEnd Date\tLocation\tRegion");

}

while (rs.next()) {

System.out.printf("%s\t%s\t%s\t%s\t%s\t%s\n",

rs.getString("name"),

rs.getString("description"),

rs.getString("start\_date"),

rs.getString("end\_date"),

rs.getString("location"),

rs.getString("region\_name"));

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* Method to logout the current user

\*/

private void logout() {

email = null;

password = null;

}

/\*\*

\* The method for printing the 5 most sold product by a producer

\*/

private void printStats() throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

try{

pstmt = con.prepareStatement(LIST\_PRODUCER\_STATS);

pstmt.setString(1, email);

rs = pstmt.executeQuery();

System.out.println("Product Code\tProduct Name\tTotal Sell");

while (rs.next()) {

System.out.printf("%s\t%s\t%s\n",

rs.getString("product\_code"),

rs.getString("name"),

rs.getString("Total Sell"));

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method for printing the orders received by the producer

\*/

private void printProducerOrders() throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

try{

pstmt = con.prepareStatement(LIST\_PRODUCER\_ORDERS);

pstmt.setString(1, email);

rs = pstmt.executeQuery();

System.out.println("Order ID\tOrder Timestamp\tTotal Price\tPayment Method");

while (rs.next()) {

System.out.printf("%s\t%s\t%s\t%s\n",

rs.getString("order\_id"),

rs.getString("order\_timestamp"),

rs.getString("total\_price"),

rs.getString("type"));

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method that prints the details of a order received by a producer selected by id

\* @param order\_id

\*/

private void printProducerOrderDetail(String order\_id) throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

try{

pstmt = con.prepareStatement(LIST\_PRODUCER\_ORDER\_DETAIL);

pstmt.setString(1, order\_id);

pstmt.setString(2, email);

rs = pstmt.executeQuery();

if (!rs.isBeforeFirst()){

System.out.println("You haven't received any order with the provided ID, please retry.");

} else {

System.out.println("Order ID\tProduct Name\tProduct Quantity\tUnit Price\tProducer");

}

while (rs.next()) {

System.out.printf("%s\t%s\t%s\t%s\n", rs.getString("order\_id"),

rs.getString("name"), rs.getString("quantity"),

rs.getString("Unit Price"), rs.getString("business\_name"));

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method that checks if the password inserted by the user is correct and correspond to the one saved in the database.

\* @param user\_psw the password inserted by the user

\* @return true if the insert password

\* @throws SQLException

\* @throws IllegalArgumentException

\* @throws IllegalStateException

\*/

private Boolean checkPassword(String user\_psw) throws SQLException, IllegalArgumentException, IllegalStateException , NoSuchAlgorithmException{

PreparedStatement pstmt = null;

ResultSet rs = null;

String real\_psw;

try {

pstmt = con.prepareStatement(SELECT\_PSW);

pstmt.setString(1, email);

rs = pstmt.executeQuery();

if(rs.next()){

real\_psw = rs.getString("password");

}

else {

throw new IllegalStateException();

}

if(user\_psw == null || !(user\_psw instanceof String) || user\_psw.length() == 0){

throw new IllegalArgumentException("Incorrect argument passed to the function");

}

MessageDigest md = MessageDigest.getInstance("MD5");

md.update(user\_psw.getBytes());

byte[] digest = md.digest();

String myHash = DatatypeConverter.printHexBinary(digest).toLowerCase();

if(myHash.equals(real\_psw) ){

password = real\_psw;

return true;

}else{

return false;

}

}

finally {

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method that checks if an user with a certain email exists

\* @return true if user is registered, false otherwise

\* @throws SQLException

\*/

private Boolean checkIfUserExists() throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

pstmt = con.prepareStatement(EMAIL\_CHECK);

pstmt.setString(1, email);

rs = pstmt.executeQuery();

return rs.next();

} finally {

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method that check if the logged in user is a customer

\* @return true if the logged in user is a customer

\* @throws SQLException

\*/

private Boolean is\_Customer() throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

String user\_role = "";

try {

pstmt = con.prepareStatement(ROLE);

pstmt.setString(1, email);

rs = pstmt.executeQuery();

if(rs.next()){

user\_role = rs.getString("role");

}

if(user\_role.equals("Customer")){

return true;

}

else{

return false;

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* The method that check if the logged in user is a producer

\* @return true if the logged in user is a producer

\* @throws SQLException

\*/

private Boolean is\_Producer() throws SQLException {

PreparedStatement pstmt = null;

ResultSet rs = null;

String user\_role = "";

try {

pstmt = con.prepareStatement(ROLE);

pstmt.setString(1, email);

rs = pstmt.executeQuery();

if(rs.next()){

user\_role = rs.getString("role");

}

if(user\_role.equals("Producer")){

return true;

}

else{

return false;

}

} finally{

if (rs != null) {

rs.close();

}

if (pstmt != null) {

pstmt.close();

}

}

}

/\*\*

\* Run the whole simulation process

\*/

private void runSimulation() throws IllegalArgumentException, IllegalStateException, NoSuchAlgorithmException {

scan = new Scanner(System.in);

connect();

try {

System.out.printf("Welcome!\n");

System.out.printf("Please insert your email to login.\n");

email = scan.nextLine();

while(!checkIfUserExists()) {

System.out.printf("Please insert a registered email.\n");

email = scan.nextLine();

}

System.out.printf("Please insert your password.\n");

String user\_password = scan.nextLine();

while(!checkPassword(user\_password)) {

System.out.printf("Please insert the correct password.\n");

user\_password = scan.nextLine();

}

Boolean loggedOut = false;

if(is\_Customer()){

while(!loggedOut) {

System.out.printf("\nWhat do you want to do?\n");

System.out.printf("Digit 1 for visualize Order History\n");

System.out.printf("Digit 2 for visualize a Order Details\n");

System.out.printf("Digit 3 for visualize a Future Events\n");

System.out.printf("Digit q to logout\n");

String choice = scan.nextLine();

if(choice.equals("1")) {

printCustomerOrders();

} else if (choice.equals("2")) {

System.out.printf("Insert your order ID\n");

String order\_id = scan.nextLine();

printCustomerOrderDetail(order\_id);

} else if (choice.equals("3")) {

System.out.printf("Insert the Region\n");

String region = scan.nextLine();

printEvents(region);

} else if (choice.equals("q")) {

logout();

loggedOut = true;

}

}

} else if(is\_Producer()) {

while(!loggedOut) {

System.out.printf("\nWhat do you want to do?\n");

System.out.printf("Digit 1 for visualize your sell stats\n");

System.out.printf("Digit 2 for visualize last 15 orders you received\n");

System.out.printf("Digit 3 for visualize a Order Details\n");

System.out.printf("Digit q to logout\n");

String choice = scan.nextLine();

if(choice.equals("1")) {

printStats();

} else if (choice.equals("2")) {

printProducerOrders();

} else if (choice.equals("3")) {

System.out.printf("Insert the order ID\n");

String order\_id = scan.nextLine();

printProducerOrderDetail(order\_id);

} else if (choice.equals("q")) {

logout();

loggedOut = true;

}

}

}

} catch (SQLException e) {

System.out.println("Database access error:");

// cycle in the exception chain

while (e != null) {

System.out.printf("- Message: %s%n", e.getMessage());

System.out.printf("- SQL status code: %s%n", e.getSQLState());

System.out.printf("- SQL error code: %s%n", e.getErrorCode());

System.out.println();

e = e.getNextException();

}

} finally {

try {

con.close();

} catch (SQLException e) {

System.out.printf("Could not close the connection to the database: %s.%n",

e.getMessage());

} finally {

con = null;

}

}

}

/\*\*

\* Connect to the database

\*/

private void connect() {

try {

Class.forName(DRIVER);

System.out.printf("Driver %s successfully registered.%n", DRIVER);

} catch (ClassNotFoundException e) {

System.out.printf(

"Driver %s not found: %s.%n", DRIVER, e.getMessage());

System.exit(-1);

}

try {

long time = -System.currentTimeMillis();

con = DriverManager.getConnection(DATABASE, USER, PASSWORD);

time += System.currentTimeMillis();

System.out.printf("Connection to database %s successfully established in %,d milliseconds.%n", DATABASE, time);

} catch (SQLException e) {

System.out.printf("Could not connect to the database: %s%n", e.getMessage());

System.exit(-2);

}

}

/\*\*

\* The function that starts running the order simulation process

\*/

public static void main(String args[]) throws IllegalArgumentException, IllegalStateException, NoSuchAlgorithmException {

UserActions ua = new UserActions();

ua.runSimulation();

}

}