ESHOP COMPANY

Project for Database Systems

This project was created as a term project for course: Database Systems – INFOA2 - ZS 23/24 – Wednesday 17:30.

Assignment is to create database model and database schema for topic of their own choosing. Project should have these parts:

- Topic – short description of what described database is about.
- Conceptual and logical (relational) diagrams
- Commands for creating schema in PostgreSQL database and filling it with sample data (CREATE TABLE, INSERT INTO)
- Sample queries in RA and SQL (24 in SQL, 10 in RA – commands can be same for RA and SQL). Try to get as many different categories of queries as possible list at the end. There is no set number of categories you need to do, but if you only do simple queries or queries of one type, it will influence your project final grade.

Yellow parts should be changed – white parts don't.

For completed project – this document must be sent to email of your LAB teacher. Your tables and sample data must also be created and filled into your assigned PostgreSQL database. Your database should be able to run all your queries (https://psqlc.db.kii.pef.czu.cz/SQLCLient).

Topic

The provided data appears to represent information related to a business involved in the production and sale of various confectionery production lines and equipment. The data includes details about different product categories such as wafer production lines, chocolate production lines, nougat production lines, and cake production lines. Additionally, information about customers, a producer, shipping records, products, product categories, reviews, order items, orders, and payments is present.

Key elements in the data include:

1. ***Categories**:* Descriptions of different production lines, such as wafer, chocolate, nougat, and cake production lines.

2. ***Customers**:* Information about customers, including their names, email addresses, and physical addresses.

3. ***Producer**:* Details about a producer, including the name and contact information.

4. ***Shipping**:* Records of shipping details, including addresses and shipment dates.

5. ***Products**:* Descriptions and prices of various production line equipment, such as ovens, cooling tunnels, cream spreading machines, and more.

6. ***Product Categories**:* Associations between products and their respective categories.

7. ***Reviews**:* Customer reviews with ratings and comments for specific products.

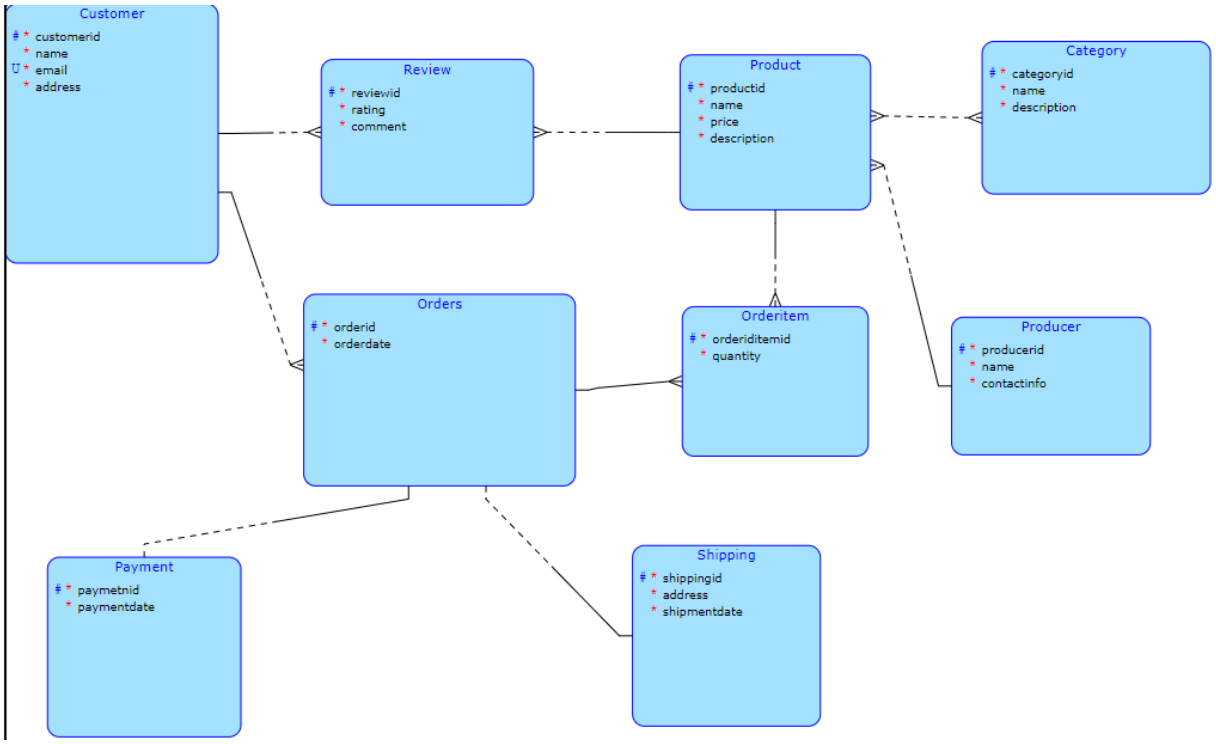
8. ***Order Items:**** Details about items in orders, including the quantity of each product.

9. ***Orders**:* Information about orders, including order IDs, customer IDs, payment IDs, order dates.

10. ***Payments**:* Records of payments made, including payment IDs, and payment dates.

Overall, the data seems to capture the relationships and transactions within a confectionery production business, encompassing product details, customer interactions, and financial transactions.

Conceptual diagram

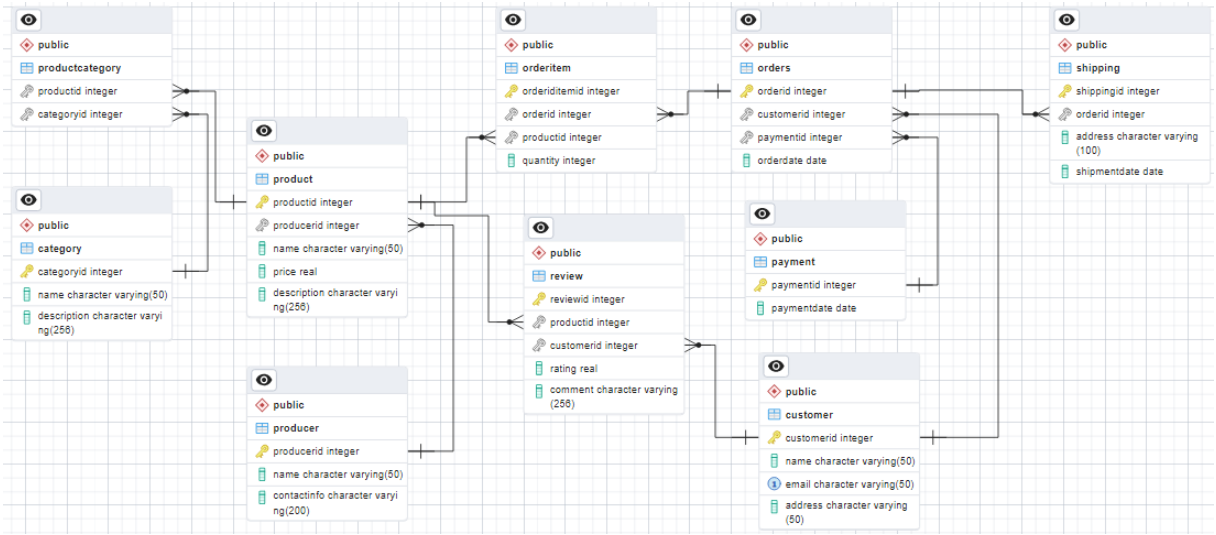


Loops discussion

The schema contains the loop CUSTOMER- ORDERS- ORDDERITEM-PRODUCT- REVİEW

The customer has an order, and the order has a specific id that they are buying and at the end, they can make a review.

Logical diagram



Tables (schema)

Normal text in this chapter should be in sql comments – so it could be all copied, pasted and run in sql query windows. For commenting one row use -- at start of row. For commenting paragraph use /* text */

Tables schema list:

```
-- Remove conflicting tables

DROP TABLE IF EXISTS category CASCADE;

DROP TABLE IF EXISTS customer CASCADE;

DROP TABLE IF EXISTS orderitem CASCADE;

DROP TABLE IF EXISTS orders CASCADE;

DROP TABLE IF EXISTS payment CASCADE;

DROP TABLE IF EXISTS producer CASCADE;

DROP TABLE IF EXISTS product CASCADE;

DROP TABLE IF EXISTS productcategory CASCADE;

DROP TABLE IF EXISTS review CASCADE;

DROP TABLE IF EXISTS shipping CASCADE;

-- End of removing


CREATE TABLE category (
   categoryid SERIAL NOT NULL,
   name VARCHAR(50) NOT NULL,
   description VARCHAR(256) NOT NULL
);
ALTER TABLE category ADD CONSTRAINT pk_category PRIMARY KEY (categoryid);


CREATE TABLE customer (
   customerid SERIAL NOT NULL,
   name VARCHAR(50) NOT NULL,
   email VARCHAR(50) NOT NULL,
   address VARCHAR(50) NOT NULL
);
ALTER TABLE customer ADD CONSTRAINT pk_customer PRIMARY KEY (customerid);
ALTER TABLE customer ADD CONSTRAINT uc_customer_email UNIQUE (email);


CREATE TABLE orderitem (
```

```
    orderiditemid SERIAL NOT NULL,
    orderid INTEGER NOT NULL,
    productid INTEGER,
    quantity INTEGER NOT NULL
);
ALTER TABLE orderitem ADD CONSTRAINT pk_orderitem PRIMARY KEY (orderiditemid);

CREATE TABLE orders (
    orderid SERIAL NOT NULL,
    customerid INTEGER,
    paymetnid INTEGER NOT NULL,
    orderdate DATE NOT NULL
);
ALTER TABLE orders ADD CONSTRAINT pk_orders PRIMARY KEY (orderid);
ALTER TABLE orders ADD CONSTRAINT u_fk_orders_payment UNIQUE (paymetnid);

CREATE TABLE payment (
    paymetnid SERIAL NOT NULL,
    paymentdate DATE NOT NULL
);
ALTER TABLE payment ADD CONSTRAINT pk_payment PRIMARY KEY (paymetnid);

CREATE TABLE producer (
    producerid SERIAL NOT NULL,
    name VARCHAR(50) NOT NULL,
    contactinfo VARCHAR(70) NOT NULL
);
ALTER TABLE producer ADD CONSTRAINT pk_producer PRIMARY KEY (producerid);

CREATE TABLE product (
    productid SERIAL NOT NULL,
    producerid INTEGER,
    name VARCHAR(50) NOT NULL,
    price REAL NOT NULL,
    description VARCHAR(256) NOT NULL
);
ALTER TABLE product ADD CONSTRAINT pk_product PRIMARY KEY (productid);
```

```
-- Warning: Missing primary key. It is recommended to explicitly define a primary key.
CREATE TABLE productcategory (
    productid INTEGER,
    categoryid INTEGER
);

CREATE TABLE review (
    reviewid SERIAL NOT NULL,
    productid INTEGER,
    customerid INTEGER,
    rating REAL NOT NULL,
    comment VARCHAR(140) NOT NULL
);
ALTER TABLE review ADD CONSTRAINT pk_review PRIMARY KEY (reviewid);

CREATE TABLE shipping (
    shippingid SERIAL NOT NULL,
    orderid INTEGER NOT NULL,
    address VARCHAR(100) NOT NULL,
    shipmentdate DATE NOT NULL
);
ALTER TABLE shipping ADD CONSTRAINT pk_shipping PRIMARY KEY (shippingid);
ALTER TABLE shipping ADD CONSTRAINT u_fk_shipping_orders UNIQUE (orderid);

ALTER TABLE orderitem ADD CONSTRAINT fk_orderitem_orders FOREIGN KEY (orderid) REFERENCES orders (orderid) ON DELETE CASCADE;
ALTER TABLE orderitem ADD CONSTRAINT fk_orderitem_product FOREIGN KEY (productid) REFERENCES product (productid) ON DELETE
CASCADE;

ALTER TABLE orders ADD CONSTRAINT fk_orders_customer FOREIGN KEY (customerid) REFERENCES customer (customerid) ON DELETE
CASCADE;
ALTER TABLE orders ADD CONSTRAINT fk_orders_payment FOREIGN KEY (paymetnid) REFERENCES payment (paymetnid) ON DELETE
CASCADE;

ALTER TABLE product ADD CONSTRAINT fk_product_producer FOREIGN KEY (producerid) REFERENCES producer (producerid) ON DELETE
CASCADE;

ALTER TABLE productcategory ADD CONSTRAINT fk_productcategory_product FOREIGN KEY (productid) REFERENCES product (productid)
ON DELETE CASCADE;
ALTER TABLE productcategory ADD CONSTRAINT fk_productcategory_category FOREIGN KEY (categoryid) REFERENCES category
(categoryid) ON DELETE CASCADE;
```

```
ALTER TABLE review ADD CONSTRAINT fk_review_product FOREIGN KEY (productid) REFERENCES product (productid) ON DELETE CASCADE;

ALTER TABLE review ADD CONSTRAINT fk_review_customer FOREIGN KEY (customerid) REFERENCES customer (customerid) ON DELETE CASCADE;


ALTER TABLE shipping ADD CONSTRAINT fk_shipping_orders FOREIGN KEY (orderid) REFERENCES orders (orderid) ON DELETE CASCADE;
```

Queries

There should be at least 24 queries in SQL total. Each query should have:

- Query Number.
- Set category from SQL Statements Cover chapter (last chapter).
- Short description.
- For 10 queries there should be variation in RA.
- Query itself – if there are multiple possibilities of how to write query, write it multiple times – write description in comment.

Normal text in this chapter should be in sql comments – so it could be all copied, pasted and run in sql query windows. For commenting one row use -- at start of row. For commenting paragraph use /* text */

Query list:

M VIEW used query
Description: customers who have placed more than 0 orders:

```
SELECT *

FROM CustomerOrderCount

WHERE total_orders > 0;
```

G4 Correlated subquery (EXISTS | NOT EXISTS)
Description: Retrieve customers who have made orders:

```
SELECT name AS customer_name

FROM customer c

WHERE EXISTS (

  SELECT 1

  FROM orders o

  WHERE o.customerid = c.customerid
```

```
);
```

F1 JOIN - JOIN ON
Description: Retrieve customers and their corresponding orders:

```
SELECT c.name AS customer_name, o.orderid, o.orderdate

FROM customer c

JOIN orders o ON c.customerid = o.customerid;
```

F4 LEFT or RIGHT OUTER JOIN
Description: Get products along with their associated reviews, including products without reviews:

```
SELECT review.rating, review.comment, product.name AS product_name, customer.name AS customer_name

FROM review

RIGHT OUTER JOIN product ON review.productid = product.productid

RIGHT OUTER JOIN customer ON review.customerid = customer.customerid;
```

G1 Subquery inside WHERE
Description: Retrieve customers who made orders after a certain date:

```
SELECT *

FROM customer

WHERE customerid IN (

   SELECT customerid

   FROM orders

   WHERE orderdate > '2023-01-01'

);
```

G1 Subquery inside WHERE
Description: List products that have reviews with ratings higher than a certain value:

```
SELECT *

FROM product

WHERE productid IN (

   SELECT productid

   FROM review

   WHERE rating > 8

);
```

F3 CROSS JOIN
Description: Combine customers with orders using CROSS JOIN:

```
SELECT customer.name AS customer_name, orders.orderid, orders.orderdate

FROM customer
```

```
CROSS JOIN orders;
```

G3 Subquery inside SELECT
Description: Retrieve customers along with their latest order dates:

```
SELECT c.name AS customer_name,
   (SELECT MAX(orderdate) FROM orders WHERE customerid = c.customerid) AS latest_order_date
FROM customer c;
```

H3 Set intersection - INTERSECT
Description: List products that have received reviews:

```
SELECT name AS product_name
FROM product
INTERSECT
SELECT p.name
FROM product p
JOIN review r ON p.productid = r.productid;
```

O UPDATE statement with subquery in WHERE/SET
Description: we want to update the price of a product for a specific customer (assume customer ID 6) to a new value (500) for all their orders:

```
UPDATE product
SET price = 500
WHERE productid IN (
   SELECT oi.productid
   FROM orderitem oi
   JOIN orders o ON oi.orderid = o.orderid
   WHERE o.customerid = 6
);
```

P DELETE statement with subquery in WHERE clause
Description: delete orders that belong to a particular customer

```
DELETE FROM orders
WHERE orderid IN (
   SELECT orderid
   FROM orders
   WHERE customerid = 6
);
```

F2 NATURAL JOIN or JOIN USING
Description: Retrieve shipping details and their associated orders using NATURAL JOIN:

```
SELECT shipping.shippingid, shipping.orderid, shipping.address, shipping.shipmentdate, orders.orderdate AS order_date

FROM shipping

NATURAL JOIN orders;
```

F5 FULL (OUTER) JOIN
Description: List all products along with their categories, including products or categories without matches:

```
SELECT product.name AS product_name, category.name AS category_name

FROM product

FULL OUTER JOIN productcategory ON product.productid = productcategory.productid

FULL OUTER JOIN category ON productcategory.categoryid = category.categoryid;
```

G2 Subquery inside FROM
Description: Retrieve customers along with the number of orders they have made:

```
SELECT c.name AS customer_name, COALESCE(o.order_count, 0) AS total_orders_made

FROM customer c

LEFT JOIN (

  SELECT customerid, COUNT(*) AS order_count

  FROM orders

  GROUP BY customerid

) o ON c.customerid = o.customerid;
```

G4 Correlated subquery (EXISTS | NOT EXISTS)
Description: List products that have not received any reviews:

```
SELECT name AS product_name

FROM product p

WHERE NOT EXISTS (

  SELECT 1

  FROM review r

  WHERE r.productid = p.productid

);
```

H1 Set union query - UNION
Description: Combine the names and emails of customers and producers:

```
SELECT name AS contact_name, email AS contact_email, 'Customer' AS contact_type

FROM customer

UNION
```

```
SELECT name, contactinfo, 'Producer'
FROM producer;
```

I1 Aggregation functions (count | sum | min | max| avg)
Description: Calculate the average price of products in each category:

```
SELECT pc.categoryid, AVG(p.price) AS avg_price
FROM product p
JOIN productcategory pc ON p.productid = pc.productid
GROUP BY pc.categoryid;
```

I2 Aggregations with GROUP BY (HAVING) clause
Description: Retrieve categories with more than 3 associated products:

```
SELECT pc.categoryid, COUNT(pc.productid) AS product_count
FROM productcategory pc
GROUP BY pc.categoryid
HAVING COUNT(pc.productid) > 3;
```

J The same query using three different formulations in SQL
Description: Using Derived Table in FROM clause:

```
SELECT *
FROM (
   SELECT o.customerid, COUNT(o.orderid) AS order_count
   FROM orders o
   GROUP BY o.customerid
) derived_table
WHERE derived_table.order_count >= 1;
```

J The same query using three different formulations in SQL
Description: Using Subquery in FROM clause:

```
SELECT *
FROM (
   SELECT o.customerid, COUNT(o.orderid) AS order_count
   FROM orders o
   GROUP BY o.customerid
) AS subq
WHERE subq.order_count >= 1;
```

K All select clauses - SELECT FROM WHERE GROUP BY HAVING ORDER BY
Description: average rating for products that have more than 10 reviews, ordered by the average rating in descending order:

```
SELECT productid, AVG(rating) AS avg_rating

FROM review

GROUP BY productid

HAVING COUNT(reviewid) > 10

ORDER BY avg_rating DESC;
```

N Insert statement for inserting list of records - INSERT without clause VALUES use, for example (add customer ID 6 all green boats reservation for all needed time intervals)
Description: assume the customer with customerid 6 is the target and the product with productid 1 is being shipped:

```
INSERT INTO shipping (orderid, address, shipmentdate)

SELECT o.orderid, c.address, '2024-12-01' AS shipmentdate

FROM orders o

JOIN customer c ON o.customerid = c.customerid

WHERE o.customerid = 6;

-- Updating order items to associate them with the product being shipped

UPDATE orderitem

SET productid = 1

WHERE orderid IN (SELECT orderid FROM orders WHERE customerid = 6);
```

K All select clauses - SELECT FROM WHERE GROUP BY HAVING ORDER BY

Description: Retrieves the names and addresses from the customer table where the customerid is not equal to 1.
customer(customerid != 1)[name, address]

```
SELECT DISTINCT name, address FROM  customer WHERE (customerid != 1)
```

F3 CROSS JOIN
Description: Joins the product and order tables based on productid, fetching name, orderdate as T1.

{ product[productid->T1, name]}*{orders[orderid ->T1,orderdate] }

```
SELECT DISTINCT *  FROM ( SELECT DISTINCT  productid AS T1, name FROM  product) AS T0 NATURAL JOIN ( SELECT DISTINCT orderid  AS T1,orderdate FROM orders) AS T2
```

J The same query using three different formulations in SQL
Description: {product[productid, name, description]} *{orders[orderid]}

```
SELECT DISTINCT *  FROM ( SELECT DISTINCT productid, name, description FROM product) AS T0 NATURAL JOIN ( SELECT DISTINCT orderid FROM orders) AS T2
```

J The same query using three different formulations in SQL
Description: Performs a join between orders and orderitem tables based on orderid, retrieving specific columns.

{orders[orderid,orderdate]}*{orderitem[orderiditemid,quantity]}

SELECT DISTINCT * FROM (SELECT DISTINCT orderid,orderdate FROM orders) AS T0 NATURAL JOIN (SELECT DISTINCT orderiditemid,quantity FROM orderitem) AS T2

J The same query using three different formulations in SQL
Description: Performs a join between product and orders tables based on productid, fetching columns related to products and order items.

{product[productid, name, description]} *{orders[orderid]}

SELECT DISTINCT * FROM (SELECT DISTINCT productid, name, description FROM product) AS T0 NATURAL JOIN (SELECT DISTINCT orderid FROM orders) AS T2

J The same query using three different formulations in SQL
Description: Relational join between orders and customer tables using customerid and orderid.

{orders[customerid,orderid]}*>{customer[customerid, name]}

SELECT DISTINCT customerid, name FROM (SELECT DISTINCT customerid,orderid FROM orders) AS T0 NATURAL JOIN (SELECT DISTINCT customerid, name FROM customer) AS T2

H2 Set substraction query - EXCEPT or MINUS (v Oracle)
Description: Fetches productid from the product table that does not exist in the orderitem table.

{product[productid]} \ {orderitem[orderiditemid]}

SELECT DISTINCT * FROM (SELECT DISTINCT productid FROM product) AS T0 EXCEPT SELECT DISTINCT * FROM (SELECT DISTINCT orderiditemid FROM orderitem) AS T2

F2 NATURAL JOIN or JOIN USING
Description: Retrieves orderid, shipping table for entries meeting the shipmentdate criteria in shipping.

{orders[orderid]}*

{shipping(shipmentdate > '5/3/2024' AND shipmentdate < '9/30/2024')[orderid]}

SELECT DISTINCT * FROM (SELECT DISTINCT orderid FROM orders) AS T0 NATURAL JOIN (SELECT DISTINCT orderid FROM shipping WHERE (shipmentdate > '5/3/2024' AND shipmentdate < '9/30/2024')) AS T2

F2 NATURAL JOIN or JOIN USING
Description: Relational join between orders and shipping tables using orderid and retrieving orderdate and shipment date.

{{orders[orderid,orderdate]}*{shipping[shippingid,shipmentdate]}}

SELECT DISTINCT * FROM (SELECT DISTINCT orderid,orderdate FROM orders) AS T0 NATURAL JOIN (SELECT DISTINCT shippingid,shipmentdate FROM shipping) AS T2

K All select clauses - SELECT FROM WHERE GROUP BY HAVING ORDER BY
Description: This query selects and returns unique product names from the PRODUCT table where the name matches Wafer Baken Oven

PRODUCT (name = 'Wafer Baken Oven')[product.name]

SELECT DISTINCT product.name FROM PRODUCT WHERE (name = 'Wafer Baken Oven')

(K)
Description: Retrieves the names and addresses from the customer table where the customerid is not equal to 1.

customer(customerid != 1)[name, address]

(F3)
Description: Joins the product and order tables based on productid, fetching name, orderdate as T1.

{ product[productid->T1, name]}*{orders[orderid ->T1,orderdate] }

(K)
Description: Retrieves shippingid, address from the shipping table where shipmentdate is after '11.01.2023'.

shipping(shipmentdate > '11.01.2023')[shippingid,address]

(J)
Description: Performs a join between orders and orderitem tables based on orderid, retrieving specific columns.

{orders[orderid,orderdate]}*{orderitem[orderiditemid,quantity]}

(H2)
Description: Fetches productid from the product table that does not exist in the orderitem table.

{product[productid]} \ {orderitem[orderiditemid]}

(J)
Description:)Performs a join between product and orders tables based on productid, fetching columns related to products and order items.

{product[productid, name, description]} *{orders[orderid]}

(J)
Description: Relational join between orders and customer tables using customerid and orderid.

{orders[customerid,orderid]}*>{customer[customerid, name]}

(F2,J)
Description: Retrieves orderid, shipping table for entries meeting the shipmentdate criteria in shipping.

{orders[orderid]}*
{shipping(shipmentdate > '5/3/2024' AND shipmentdate < '9/30/2024')[orderid]}

(J,F2)
Description: Relational join between orders and shipping tables using orderid and retrieving orderdate and shipment date.

{{orders[orderid,orderdate]}*{shipping[shippingid,shipmentdate]}}

(K)
Description: This query selects and returns unique product names from the PRODUCT table where the name matches Wafer Baken Oven

PRODUCT (name = 'Wafer Baken Oven')[product.name]

Sample data

Normal text in this chapter should be in sql comments – so it could be all copied, pasted and run in sql query windows. For commenting one row use -- at start of row. For commenting paragraph use /* text */

Sample data list:

CATEGORY

insert into category (categoryid, name, description) values (1, 'wafer production line', 'Wafer lines create crispy wafers from flour, water, precise baking, cutting, filling, coating, ensuring delicate handling, and packaging.');

insert into category (categoryid, name, description) values (2, 'chocolate production line', 'Chocolate lines grind cocoa beans, mix with sugar, cocoa butter, refine, conch, temper, mold, cool, and package to create various chocolate products, demanding precise temperature control and quality ingredients.');

insert into category (categoryid, name, description) values (3, 'Nougat production line', 'Nougat lines blend sugar/honey, nuts, fruits, heat to specific temperatures, stretch, shape, cut, and package the soft, chewy confection');

insert into category (categoryid, name, description) values (4, 'cake production line', 'A cake line mixes flour, sugar, eggs, flavors, automates mixing, baking, cooling, frosting, and packaging for diverse cake types.');

CUSTOMER

insert into customer (customerid, name, email, address) values (1, 'Zelda', 'zpechell0@cdbaby.com', '53199 Graceland Street');

insert into customer (customerid, name, email, address) values (2, 'Eddi', 'erodger1@slate.com', '737 Londonderry Hill');

insert into customer (customerid, name, email, address) values (3, 'Bryon', 'bhaggerstone2@github.io', '0 Troy Lane');

insert into customer (customerid, name, email, address) values (4, 'Carmela', 'cstanislaw3@sourceforge.net', '465 Westerfield Road');

insert into customer (customerid, name, email, address) values (5, 'Marybelle', 'mwhittam4@lycos.com', '203 Bartillon Court');

insert into customer (customerid, name, email, address) values (6, 'Fonz', 'flotze5@bizjournals.com', '7955 Birchwood Drive');

insert into customer (customerid, name, email, address) values (7, 'Suki', 'shassen6@hostgator.com', '51 Northridge Circle');

insert into customer (customerid, name, email, address) values (8, 'Cleveland', 'cchaloner7@tripod.com', '7807 Bunker Hill Avenue');

insert into customer (customerid, name, email, address) values (9, 'Peder', 'pmammatt8@house.gov', '2 Scoville Trail');

insert into customer (customerid, name, email, address) values (10, 'Stearne', 'smaneylaws9@amazon.co.uk', '1 Fremont Junction');

insert into customer (customerid, name, email, address) values (11, 'Cassandra', 'cruslina@soundcloud.com', '8078 Logan Lane');

insert into customer (customerid, name, email, address) values (12, 'Michele', 'mdobingb@typepad.com', '61 Del Mar Park');

insert into customer (customerid, name, email, address) values (13, 'Conan', 'ccestardc@java.com', '237 Bobwhite Crossing');

insert into customer (customerid, name, email, address) values (14, 'Daffi', 'dedmondsond@ft.com', '77164 Oak Valley Way');

insert into customer (customerid, name, email, address) values (15, 'Adeline', 'adhennine@theglobeandmail.com', '39 Mitchell Place');

PRODUCER

insert into producer (producerid, name, contactinfo) values (1, 'Atlas Machine', 'Atlas Machinery Organized Industrial Zone 21. St. No:8 70000, 70000 Karaman Center/Karaman');

SHIPPING

insert into shipping (shippingid, orderid, address, shipmentdate) values (1,1 ,'Apt 1421', '5/30/2024');

insert into shipping (shippingid, orderid, address, shipmentdate) values (2,2 ,'Room 935', '9/3/2024');

insert into shipping (shippingid, orderid, address, shipmentdate) values (3,3 ,'Suite 34', '5/17/2024');

insert into shipping (shippingid, orderid, address, shipmentdate) values (4,4 ,'Room 1374', '1/2/2024');

insert into shipping (shippingid, orderid, address, shipmentdate) values (5,5 ,'Apt 1815', '1/26/2024');

PRODUCT

insert into product (productid, producerid, name, price, description) values (1, 1, 'Wafer Baken Oven', 300000, 'Produces 280x380 to 350x500 sheets, molds 24-90, uses LPG/Natural Gas, efficient consumption, controlled by touch-screen PLC Panel.');

insert into product (productid, producerid, name, price, description) values (2, 1, 'Wafer Sheet Cooling Tunnel', 35000, 'Used to cool wafers in standard conditions, discarding incompatible sheets. Operates in sync with baking oven, humidification tunnel, and cream spreader.');

insert into product (productid, producerid, name, price, description) values (3, 1, 'Wafer Cream Spreading Machine', 60000, 'Enables precise cream spreading, customizable layers. Includes walled reservoir, synced weighing for accuracy. Controlled via touch panel.');

insert into product (productid, producerid, name, price, description) values (4, 1, 'Wafer Cutting Machine', 25000, 'Cuts creamed wafers to size with adjustable steel blades. Continuous process with auto-feeding, bidirectional cutting, touch panel control.');

insert into product (productid, producerid, name, price, description) values (5, 1, 'Chocolate pre-mixing', 30000, 'Gathers chocolate ingredients, auto-adjusts amounts. Heat-walled reservoir circulates hot water. Dual-motor-driven mixers, transfers mix to mixer.');

insert into product (productid, producerid, name, price, description) values (6, 1, 'Chocolate Preparation Mixer', 80000, 'Mixes chocolate uniformly in heated reservoir, uses steel balls for texture. Equipped with strainer, magnets, pump for circulation, transfers mix to stock tank.');

insert into product (productid, producerid, name, price, description) values (7, 1, 'Chocolate Tempering Machine', 100000, 'Chocolate stock tank maintains mixer's homogeneity. Heat-walled, circulates hot water. Allows pump transfer, customizable capacities.');

insert into product (productid, producerid, name, price, description) values (8, 1, 'Pattern Machine', 45000, 'Patterning machine adds designs to chocolate wafers. Speed control adjusts pattern density. Features own reservoir, pump');

insert into product (productid, producerid, name, price, description) values (9, 1, 'Syrup Preparing Tank', 37000, 'Nougat batter tank for homogeneous prep before cooking. Self-circulating, heat-walled. Automated recipe adjustment, easy wash with rotating water heads.');

insert into product (productid, producerid, name, price, description) values (10, 1, 'Nougat Cooking and Turbo Mixer', 49000, 'Dual reservoir system cooks under pressure/vacuum, prevents foam absorption. Enables continuous processing, actuated valves control inlets/outlets');

insert into product (productid, producerid, name, price, description) values (11, 1, 'Nougat Cylinders', 66000, 'Shapes cooked batter directly, handles cooling, forming, thickness. Maintains product structure, prevents adhesion. Syncs with cooling tunnel, versatile design');

insert into product (productid, producerid, name, price, description) values (12, 1, 'Floor Cooling Tunnel', 80000, 'Cools nougat/caramel with various systems. Reduces temp via water circulation. Adjustable, easy clean, precise band adjustment, dual-side refrigeration control.');

insert into product (productid, producerid, name, price, description) values (13, 1, 'Cake Baking Oven', 240000, 'Oven customizable in stainless/painted covers. Offers 2-4 heating cells. Wire/chain belt system, observation windows, steam chutes. Adjustable cooking time, variable fuel type');

insert into product (productid, producerid, name, price, description) values (14, 1, 'Cake Jelly Injection Machine', 58000, 'Injects jelly into finished products, adjustable for cake quantity. Automated transfer, servo-driven with dual-axis operation. Continuous jelly press without halting product on belt');

insert into product (productid, producerid, name, price, description) values (15, 1, 'Cake Vacuum Machine', 36000, 'Vacuum transfers cooked products between bands. Stainless/painted options. Dual-axis operation, fully automatic with sensors. Custom holders prevent product damage.');

PRODUCTCATEGORY

insert into productcategory (productid, categoryid) values (1, 1);

insert into productcategory (productid, categoryid) values (2, 1);

insert into productcategory (productid, categoryid) values (3, 1);

insert into productcategory (productid, categoryid) values (4, 1);

insert into productcategory (productid, categoryid) values (5, 2);

insert into productcategory (productid, categoryid) values (6, 2);

insert into productcategory (productid, categoryid) values (7, 2);

insert into productcategory (productid, categoryid) values (8, 2);

insert into productcategory (productid, categoryid) values (9, 3);

insert into productcategory (productid, categoryid) values (10, 3);

insert into productcategory (productid, categoryid) values (11, 3);

insert into productcategory (productid, categoryid) values (12, 3);

insert into productcategory (productid, categoryid) values (13, 4);

insert into productcategory (productid, categoryid) values (14, 4);

insert into productcategory (productid, categoryid) values (15, 4);

REVIEW

insert into review (reviewid, productid, customerid, rating, comment) values (1, 1, 1, 10, 'The wafer oven delivers performance that exceeds expectations! It is easy to use, wafers cook perfectly and cleaning is easy. 10/10 for its durable construction and great customer service! I definitely recommend it.');

insert into review (reviewid, productid, customerid, rating, comment) values (2, 2, 2, 10, 'Exceptional performance! This cooling tunnel ensures perfect wafer sheets consistently. User-friendly, durable, and energy-efficient. Top-notch customer service. A solid 10/10 for reliability and efficiency');

insert into review (reviewid, productid, customerid, rating, comment) values (3, 3, 3, 9, 'Efficient and precise, the Wafer Cream Spreading Machine elevates production. Minor usability tweaks aside, its exceptional performance warrants a solid 9/10!');

insert into review (reviewid, productid, customerid, rating, comment) values (4, 4, 4, 9, 'The Wafer Cutting Machine is impressive! Efficient, precise cuts, and user-friendly. Its reliable, but slight adjustments could enhance its speed. Overall, a great investment for quality production. Rated 9/10!');

insert into review (reviewid, productid, customerid, rating, comment) values (5, 5, 5, 7, 'Efficient but inconsistent. Simplifies the process but needs better uniformity. With improvements, it has potential for a higher score.');

insert into review (reviewid, productid, customerid, rating, comment) values (6, 6, 6, 7, 'Mixes well, lacks ease. Noise could improve. Responsive service. Decent, some tweaks needed');

insert into review (reviewid, productid, customerid, rating, comment) values (7, 7, 7, 5, 'The Chocolate Tempering Machine is decent, but it falls short on consistency. Improvement needed for better performance. ');

insert into review (reviewid, productid, customerid, rating, comment) values (8, 8, 8, 5, 'The Cake Vacuum Machine has potential but falls short. It struggles with larger cakes and needs better suction power. Room for improvement');

ORDER ITEM

ALTER TABLE orders DROP COLUMN totalamount;

insert into orderitem (orderiditemid, orderid, productid, quantity) values (1, 1, 1, 1);

insert into orderitem (orderiditemid, orderid, productid, quantity) values (2, 2, 2, 1);

insert into orderitem (orderiditemid, orderid, productid, quantity) values (3, 3, 3, 1);

insert into orderitem (orderiditemid, orderid, productid, quantity) values (4, 4, 4, 1);

insert into orderitem (orderiditemid, orderid, productid, quantity) values (5, 5, 5, 2);

ORDERS

insert into orders (orderid, customerid, paymentid, orderdate, totalamount) values (1, 1, 6, '9/10/2023', 1);

insert into orders (orderid, customerid, paymentid, orderdate, totalamount) values (2, 2, 7, '10/13/2023', 2);

insert into orders (orderid, customerid, paymentid, orderdate, totalamount) values (3, 3, 8, '7/28/2023', 3);

insert into orders (orderid, customerid, paymentid, orderdate, totalamount) values (4, 4, 9, '12/30/2022', 4);

insert into orders (orderid, customerid, paymentid, orderdate, totalamount) values (5, 5, 10, '4/26/2023', 5);

PAYMENT

ALTER TABLE payment DROP COLUMN amount;

insert into payment (paymentid, amount, paymentdate) values (6, 1, '6/7/2023');

insert into payment (paymentid, amount, paymentdate) values (7, 1, '9/27/2023');

insert into payment (paymentid, amount, paymentdate) values (8, 1, '5/29/2023');

insert into payment (paymentid, amount, paymentdate) values (9, 1, '3/27/2023');

insert into payment (paymentid, amount, paymentdate) values (10, 1, '9/7/2023');

SQL Statements Cover

Category	Category statement description
A	Simple (positive) query with at least two joined tables (list of patients - name, address who were examined by doctor Braun)
B	Simple (negative) query (list of patients - name, address who were not visiting doctor Braun) (select all doctors who had patients,...)
C	Choose all records, which have relation for "X" only …(list of patients - name, address who were examined by doctor Braun ONLY - the patients didn't have visit at another doctor).
D1	Select all records which are at the relation with all.General quantified query (list of doctors - name, address who were visited by EACH patient, who visited doctor Braun)
D2	The result validity check from category D1. For example if query {list of teachers, who had lecture during ALL semesters begin at winter 2001/2002 till summer semester 2008/2008 included} shows the teacher "123 Josef Pavlicek", so the validity check will be query {{list of all semester at the season, where was teaching Pavlicek }\ (minus){list of all semester at the season}} and the result must be empty.
F1	join - JOIN ON

F2	NATURAL JOIN | JOIN USING
F3	CROSS JOIN
F4	LEFT | RIGHT OUTER JOIN
F5	FULL (OUTER) JOIN
G1	Subquery inside WHERE
G2	Subquery inside FROM
G3	Subquery inside SELECT
G4	Correlated subquery (EXISTS | NOT EXISTS)
H1	Set union query - UNION
H2	Set substraction query - EXCEPT or MINUS (v Oracle)
H3	Set intersection - INTERSECT
I1	Aggregation functions (count | sum | min | max| avg)
I2	Aggregations with GROUP BY (HAVING) clause
J	The same query using three different formulations in SQL
K	All select clauses - SELECT FROM WHERE GROUP BY HAVING ORDER BY
L	Make VIEW
M	View used query
N	Insert statement for inserting list of records - INSERT without clause VALUES use, for example (add customer ID 6 all green boats reservation for all needed time intervals)
O	UPDATE statement with subquery in WHERE/SET
P	DELETE statement with subquery in WHERE clause