DATABASE FOR A BAKERY

Kocsis Edina Katalin
JBWPAI

2017. fall

Project supervisor: Gábor Csaba Attila

Introduction

I would like to design a database for a bakery. It is both factory and wholesaler. This company has more factories and different customers, so it's mandatory to maintain a proper database. With this database the factories can easily manage the orders, the storeroom can properly handle the stock, and the finance department can simply log the orders.

Description

The data tables I'd like to store are the following:

Person - A human being, attributes are general informations like telephone number, e-mail address, and name. A person in this database is either an employee or a contact.

Employee - It is a subclass of person. An employee of the firm, who works in one of the factories. An employee has further attributes like position, date of birth, a unique ID, and in which factory he/she works at.

Contact - It is a subclass of person. Contact person of a supplier, the factories order the ingredients from the supplier companies.

Customer - It can be either a person or a company, with a contact person in both cases. (For example, if I'm a party organizer, the customer (buyer) itself is my customer, but i'm the contact, because I manage the order.)

Factory - the factory itself, with data like location (postal code, country, city, etc.), territory, and an ID which connects to the employees.

Placement - A customer's order, includes an ID number, comments, status and so on.

Order details - It is the expansion of Placement. Only the designated factory should care about it. It has further, detailed informations about the Placement, including the ID of the ordered products, and the price.

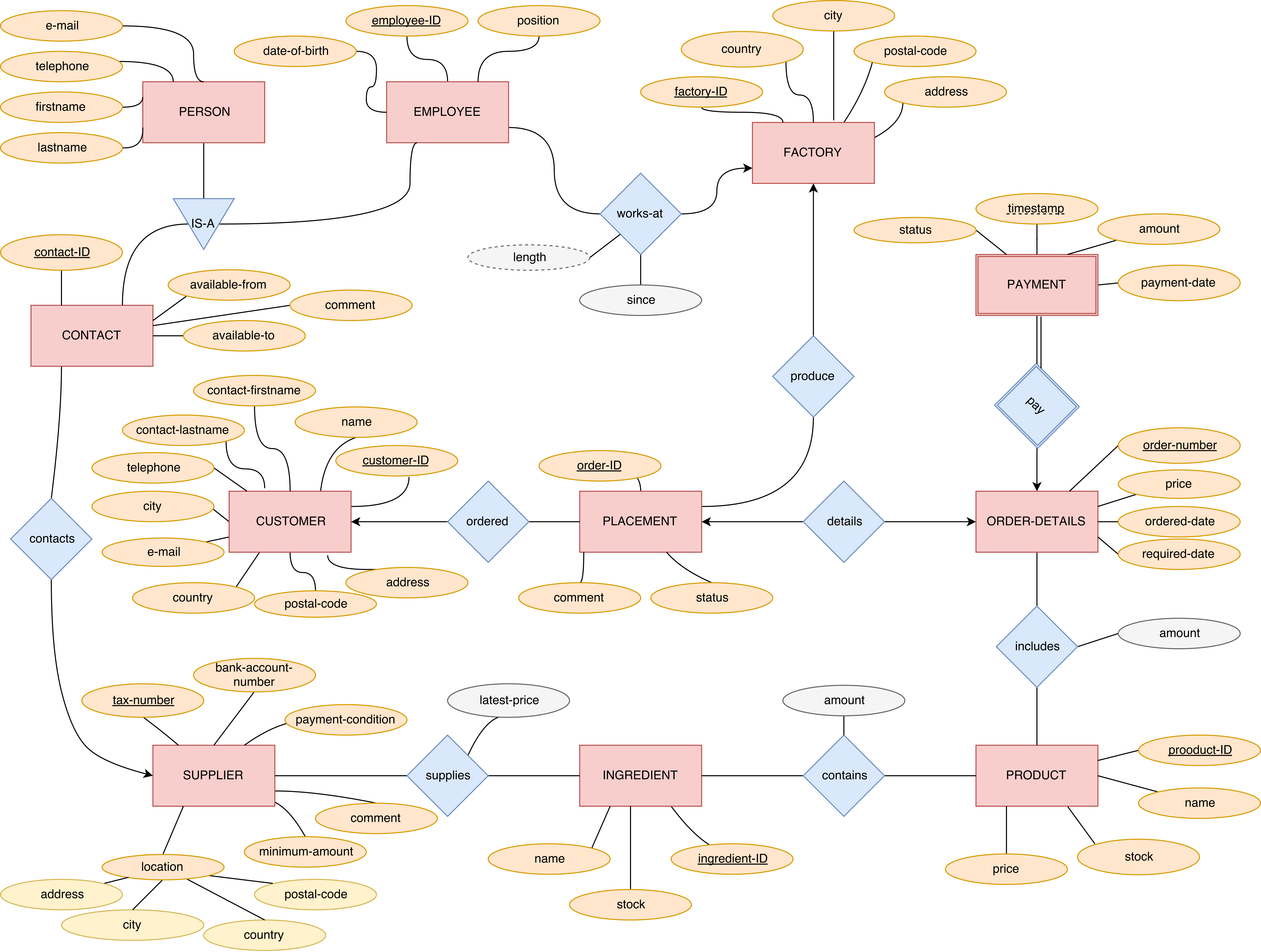
Payment - Every order have a payment data, but it has no identifying data, it's a weak entity of Order details. It stores a timestamp, amount, status and so on.

Product - A product of the company, stores data like name, stock, price, and ID of the ingredients.

Ingredient - Base material the producing needs, has an identifier ID, name, stock and ID of the possible suppliers.

Supplier - An independent company, who supplies the factories. Stores the order conditions (minimum amount, payment conditions, etc), bank account number, tax number, possible comments, ID of the contacts and further informations.

The tables have more attributes than I've listed, I hope it's not a problem that I haven't written them down. They are only obvious informations and they don't connect the entities.


e-mail
telephone
firstname
lastname
PERSON
date-of-birth
employee-ID
position
EMPLOYEE
city
country
postal-code
factory-ID
address
FACTORY
IS-A
works-at
length
since
contact-ID
available-from
comment
available-to
CONTACT
status
timestamp
amount
PAYMENT
payment-date
produce
pay
contact-firstname
contact-lastname
name
customer-ID
telephone
city
e-mail
CUSTOMER
address
country
postal-code
order-ID
ordered
PLACEMENT
comment
status
details
order-number
price
ORDER-DETAILS
ordered-date
required-date
contacts
includes
amount
bank-account-number
tax-number
payment-condition
latest-price
amount
SUPPLIER
supplies
INGREDIENT
contains
PRODUCT
prooduct-ID
name
comment
minimum-amount
location
address
postal-code
city
country
name
stock
ingredient-ID
price
stock

Relationships and ISAs

- contact and employee ISA with person
- employee has relationship with factory - many to one
- factory has relationship with placement - one to many
- customer has relationship with placement - one to many
- placement has relationship with order details - one to one
- order details has relationship with product - many to many
- product has relationship with ingredient - many to many
- ingredient has relationship with supplier - many to many
- supplier has relationship with contact - one to many

Relational model

Entities and attributes

EMPLOYEE (employee-ID, firstname, lastname, e-mail, telephone, date-of-birth, position)
CONTACT (contact-ID, firstname, lastname, e-mail, telephone, available-from, available-to, comment, tax-number)
CUSTOMER (customer-ID, country, city, postal-code, address, e-mail, telephone, contact-lastname, contact-firstname, name)
FACTORY (factory-ID, country, city, postal-code, address)
PLACEMENT (order-ID, comment, status, factory-ID, customer-ID)
ORDER-DETAILS (order-number, price, ordered-date, required-date, order-ID)
PRODUCT (product-ID, price, name, stock)
INGREDIENT (ingredient-ID, name, stock)
SUPPLIER (tax-number, bank-account-number, payment-condition, comment, minimum-amount, country, city, postal-code, address)
PAYMENT (order-number, timestamp, status, amount, payment-date)

Relations

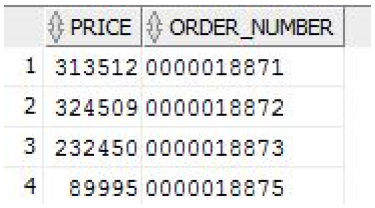
works-at (employee-ID, factoty-ID, since)
includes (order-number, product-ID, amount)
contains (product-ID, ingredient-ID, amount)
supplies (ingredient-ID, tax-number, latest-price)

Queries

Find every order and give their order-number and price, where the price is bigger than 50000!
Add meg minden olyan rendelés számát és értékét, ahol a rendelés értéke nagyobb, mint 50000!

$\Pi_{price,\ order\ number}(\sigma_{price>50000}(order - details))$

```
select price, order_number
from ORDER_DETAILS
where price>50000;
```

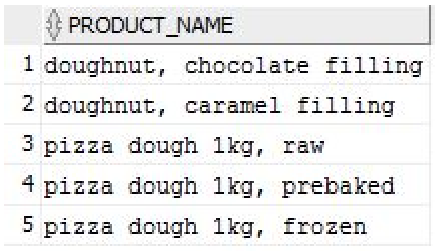


	PRICE	ORDER_NUMBER
1	313512	0000018871
2	324509	0000018872
3	232450	0000018873
4	89995	0000018875

Find every product and give their name, which contains yeast!
Add meg minden élesztőt tartalmazó termék nevét!

$\Pi_{produc-name}(product \bowtie contains \bowtie \sigma_{ingredient-name\ =\ 'yeast'}(ingredient))$

```
select product_name
from PRODUCT, CONTAINS, INGREDIENT
where PRODUCT.PRODUCT_ID = CONTAINS.PRODUCT_ID and
INGREDIENT.INGREDIENT_ID = CONTAINS.INGREDIENT_ID
and ingredient_name = 'yeast';
```

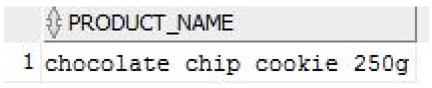


	PRODUCT_NAME
1	doughnut, chocolate filling
2	doughnut, caramel filling
3	pizza dough 1kg, raw
4	pizza dough 1kg, prebaked
5	pizza dough 1kg, frozen

Find every product and give their name and stock, which name contains the word “chocolate” and stock is bigger than 0! (List every product with chocolate, which is currently at stock…)
Adj meg minden olyan terméket, aminek a nevében benne van, hogy “chocolate”, és a készlete nagyobb, mint 0! (Listázd ki az összes csokis terméket, ami van készleten…)

$$\Pi_{product-name}(\sigma_{product-name\ like\ '\%chocolate\%'\ and\ stock>0}(product))$$

```
select product_name
from product
where product_name like '%chocolate%'
and stock >0;
```

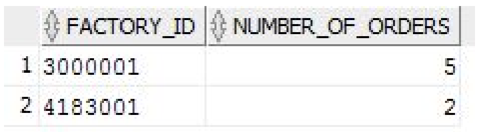


	PRODUCT_NAME
1	chocolate chip cookie 250g

Give the number of orders in every factory, in descending order!
Add meg csökkenő sorrendben, hogy melyik gyárhoz hány megrendelés tartozik!

$$_{factory-ID}g_{count(order-ID)}(placement)$$

```
select factory_ID, COUNT (order_ID) as number_of_orders
from PLACEMENT
GROUP BY factory_ID,
ORDER BY number_of_orders desc;
```



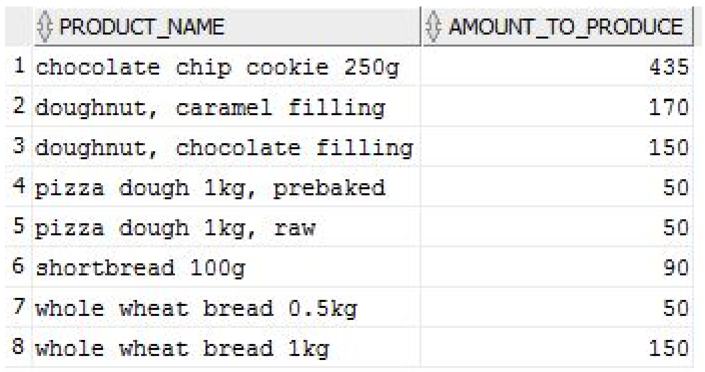
	FACTORY_ID	NUMBER_OF_ORDERS
1	3000001	5
2	4183001	2

Give the number and name of products waiting to be produced to fulfill every order, in ascending abc order!

Add meg betűrendben a még legyártandó termékek számát és nevét, ha minden rendelés telejsítéséhez szeretnénk eleget készíteni!

$\varrho_{product-name,\ amount-to-produce}\Pi_{myname,\ (-1)*mystock-mysum}(\sigma_{mystock-mysum<0}(\varrho_{sums(myname,\ mystock,\ mysum)}(\Pi_{product-name,\ stock,\ sum(amount)}({}_{product-name,\ stock}g_{sum(amount)}(includes \bowtie product)))$

```
select sums.myname as product_name,
(-1)*(sums.mystock-sums.mysum) as amount_to_produce
from(
select product_name as myname, stock as mystock, sum(amount)
as mysum
from includes, product
where includes.product_ID = product.product_ID
group by product_name, stock) sums
where sums.mystock - sums.mysum<0;
```



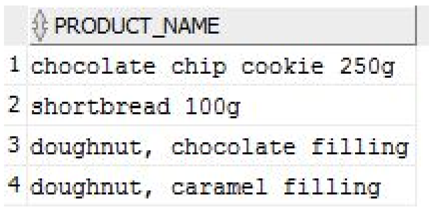
	PRODUCT_NAME	AMOUNT_TO_PRODUCE
1	chocolate chip cookie 250g	435
2	doughnut, caramel filling	170
3	doughnut, chocolate filling	150
4	pizza dough 1kg, prebaked	50
5	pizza dough 1kg, raw	50
6	shortbread 100g	90
7	whole wheat bread 0.5kg	50
8	whole wheat bread 1kg	150

Give the name of the products which are at stock, but does not contain any yeast. and which are not at stock, but contain egg!

Add meg azoknak a termékeknek a nevét, amik vannak készleten, de nem tartalmaznak élesztőt, és nincsenek készleten, de tartalmaznak tojást.

$$\Pi_{product-name}(\sigma_{stock>0}(product) \bowtie \Pi_{product-ID}(contains \bowtie$$
$$\Pi_{ingredient-ID}(\sigma_{ingredient-name\ like\ '\%yeast\%'}(ingredient))) +$$
$$\Pi_{product-name}(\sigma_{stock<0}(product) \bowtie \Pi_{product-ID}(contains \bowtie$$
$$\Pi_{ingredient-ID}(\sigma_{ingredient-name\ like\ '\%egg\%'}(ingredient)))$$

```
select product_name
from product
where(
stock>0 and product_id not in (
select product_id
from contains, ingredient
where ingredient.ingredient_ID = contains.ingredient_id
and ingredient_name like '%yeast%'))
or(
stock=0 and product_id in (
select product_id from contains, ingredient
where ingredient.ingredient_ID = contains.ingredient_id
and ingredient_name like '%egg%'));
```

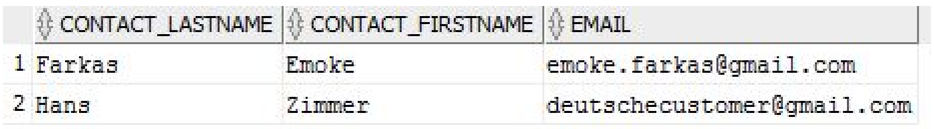


	PRODUCT_NAME
1	chocolate chip cookie 250g
2	shortbread 100g
3	doughnut, chocolate filling
4	doughnut, caramel filling

Find the name and e-mail address of contacts and customers from abroad!
Keresd meg a külföldi contact-ok és customer-ek nevét és e-mail címét!

$$\Pi_{contact-lastname,\ contact-firstname,\ email}(\sigma_{country\ not\ like\ '\%Hungary\%'}(customer)) \cup$$
$$\Pi_{lastname,\ firstname,\ email}(contact \bowtie \Pi_{tax-number}(\sigma_{country\ not\ like\ '\%Hungary\%'}(supplier)))$$

```
select contact_lastname, contact_firstname, email
from customer
where country not like '%Hungary%'
union
select lastname, firstname, email
from contact
where tax_number in (
select tax_number
from supplier
where country not like '%Hungary%');
```

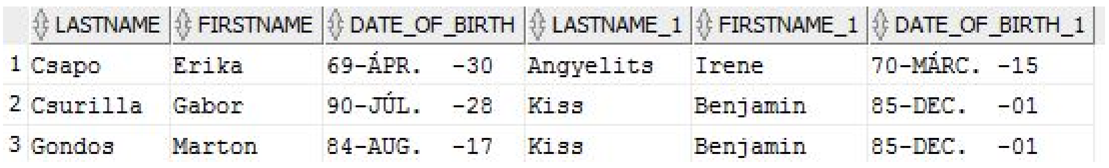


	CONTACT_LASTNAME	CONTACT_FIRSTNAME	EMAIL
1	Farkas	Emoke	emoke.farkas@gmail.com
2	Hans	Zimmer	deutschecustomer@gmail.com

Select employee pairs whose age differs less than 5 years!
Válaszd ki az olyan alkalmazottakat, akik között kevesebb, mint 5 év a korkülönbség.

$$\Pi_{e1.lastname,\ e1.firstname,\ e1.date\_of\_birth,\ e2.lastname,\ e2.firstname,\ e2.date\_of\_birth}($$
$$\sigma_{e1.employee-ID<e2.employee-ID\ and\ abs(e1.date\_of\_birth-e2.date\_of\_birth)/365<5}\ (\varrho_{e1}(employee)\times\varrho_{e2}(employee)))$$

```
select e1.lastname, e1.firstname, e1.date_of_birth,
e2.lastname, e2.firstname, e2.date_of_birth
FROM employee e1,employee e2
where e1.employee_ID < e2.employee_ID
and abs(e1.date_of_birth-e2.date_of_birth)/365<5
ORDER BY e1.lastname, e2.lastname;
```

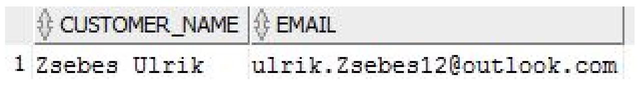


	LASTNAME	FIRSTNAME	DATE_OF_BIRTH	LASTNAME_1	FIRSTNAME_1	DATE_OF_BIRTH_1
1	Csapo	Erika	69-ÁPR. -30	Angyelits	Irene	70-MÁRC. -15
2	Csurilla	Gabor	90-JÚL. -28	Kiss	Benjamin	85-DEC. -01
3	Gondos	Marton	84-AUG. -17	Kiss	Benjamin	85-DEC. -01

Select the customers and their e-mail address, who ordered every kind of doughnut from the offer!
Válaszd ki azokat az ügyfeleket az e-mail címükkel együtt, akik a kínálatban szereplő összes fajta fánkból rendeltek!

$$\Pi_{customer-name,\ email}(customer \bowtie \Pi_{order-ID}(placement \bowtie \Pi_{order-number}(order\_details \bowtie \Pi_{product-ID}(includes \bowtie \Pi_{product-ID}(\sigma_{product-name\ like\ "\%doughnut\%"}(product))))) \div \Pi_{product-ID}(\sigma_{product-name\ like\ "\%doughnut\%"}(product))$$

```
select customer_name, email
from customer
where customer_ID in(
select customer_id from placement where order_id in(
select order_id from order_details where order_number in(
select order_number from includes where product_ID in (
select product_ID from product where product_name like
'%doughnut%')
group by order_number
having count(*) = (select count(*) from product where
product_name like '%doughnut%'))));
```

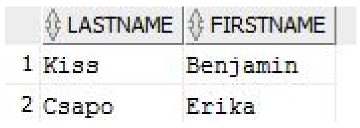


	CUSTOMER_NAME	EMAIL
1	Zsebes Ulrik	ulrik.Zsebes12@outlook.com

Select every employee, who works as janitor or storeman!
Válassz ki minden alkalmazottat, aki takarítóként vagy raktárosként dolgozik!

$$\Pi_{lastname,\ firstname}(\sigma_{pos='janitor' \vee pos='storeman'}(employee))$$

```
select lastname, firstname
from employee
where pos = any('janitor', 'storeman');
```



	LASTNAME	FIRSTNAME
1	Kiss	Benjamin
2	Csapo	Erika

View

In SQL, a view is a virtual table based on the result-set of an SQL statement.
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

```
create view loyal_doughnut_lovers as
select customer_name, email
from customer
where customer_ID in(
select customer_id from placement where order_id in(
select order_id from order_details where order_number in(
select order_number from includes where product_ID in (
select product_ID from product where product_name like
'%doughnut%')
group by order_number
having count(*) = (select count(*) from product where
product_name like '%doughnut%'))));
```

Update

The UPDATE statement is used to update existing records in a table.
I've created 2 updates, to demonstrate it.

Update the status to 'fulfilled' on every order, which required date is 2017. 10. 24.

```
update placement
set status = 'fulfilled'
where order_id in (
select order_id
from order_details
where required_date = TO_DATE('2017-10-24', 'YYYY-MM-DD')
);
```

The price of the egg insanely increases. Add +10 to the price of every product containing any egg!

```
update product
set price = price + 10
where product_id in (
select product_id
from contains
where ingredient_id in(
select ingredient_id
from ingredient
where ingredient_name = 'egg'
));
```

Delete

The DELETE statement is used to delete existing records in a table.
I’ve deleted the customer called 'Palne Rozsasi' with all her placements (and the other required dependencies).

```
delete from includes
where order_number in(
select order_number
from order_details
where order_id in(
select order_id
from placement
where customer_id in(
select customer_id
from customer
where customer_name = 'Palne Rozsasi')));

delete from payment
where order_number in(
select order_number
from order_details
where order_id in(
select order_id
from placement
where customer_id in(
select customer_id
from customer
where customer_name = 'Palne Rozsasi')));
```

```
delete from order_details
where order_id in(
select order_id
from placement
where customer_id in(
select customer_id
from customer
where customer_name = 'Palne Rozsasi'));

delete from placement
where customer_id in(
select customer_id
from customer
where customer_name = 'Palne Rozsasi');

delete from customer
where customer_name = 'Palne Rozsasi';
```

Normal forms

The normal forms and their criteria:

1NF

- Eliminate repeating groups in individual tables.
- Create a separate table for each set of related data.
- Identify each set of related data with a primary key

2NF

- The relation is in first normal form (1NF).
- every non-prime attribute of the relation is dependent on the whole of every candidate key.

3NF

- The relation is in second normal form (2NF).
- Every non-prime attribute of the relation is non-transitively dependent on every key of the relation.

BCNF

- The relation is in third normal form (3NF).
- Attributes depend only on any super key.

Normalization

I have checked the normal forms of every table. Firs I've listed all the dependencies, then I could decide which normal forms they fulfill. The goal is the BCNF, because there is always a lossless decomposition in BCNF.

EMPLOYEE (employee-ID, firstname, lastname, e-mail, telephone, date-of-birth, position)
$F_{EMPLOYEE}$(employee-ID → firstname, employee-ID → lastname, employee-ID → e-mail, employee-ID → telephone, employee-ID → date-of-birth, employee-ID → position)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

CONTACT (contact-ID, firstname, lastname, e-mail, telephone, available-from, available-to, comment, tax-number)
$F_{CONTACT}$(contact-ID → firstname, contact-ID → lastname, contact-ID → e-mail, contact-ID → telephone, contact-ID → available-from, contact-ID → available-to, contact-ID → comment, contact-ID → tax-number)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

CUSTOMER (customer-ID, country, city, postal-code, address, e-mail, telephone, contact-lastname, contact-firstname, name)
$F_{CUSTOMER}$(customer-ID → country, customer-ID → city, customer-ID → postal-code, customer-ID → address, customer-ID → e-mail, customer-ID → telephone, customer-ID → contact-lastname, customer-ID → contact-firstname, customer-ID → name)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

PRODUCT (product-ID, price, name, stock)
$F_{PRODUCT}$(product-ID → price, product-ID → name, product-ID → stock)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

FACTORY (factory-ID, country, city, postal-code, address)
$F_{FACTORY}$(factory-ID → country, factory-ID → city, factory-ID → postal-code, factory-ID → address)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

PLACEMENT (order-ID, comment, status, factory-ID, customer-ID)
$F_{PLACEMENT}$(order-ID → comment, order-ID → status, order-ID → factory-ID, order-ID → customer-ID)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

ORDER-DETAILS (order-number, price, ordered-date, required-date, order-ID)
$F_{ORDER\_DETAILS}$(order-number → price, order-number → ordered-date, order-number → required-date, order-number → order-ID, order-ID → order-number, order-ID → price, order-ID → ordered-date, order-ID → required-date)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

SUPPLIER (tax-number, bank-account-number, payment-condition, comment, minimum-amount, country, city, postal-code, address)

$F_{SUPPLIER}$(tax-number → bank-account-number, tax-number → payment-condition, tax-number → comment, tax-number → minimum-amount, tax-number → country, tax-number → city, tax-number → postal-code, tax-number → address)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

PAYMENT (order-number, timestamp, status, amount, payment-date)

$F_{PAYMENT}$(order-number & timestamp → status, order-number & timestamp → amount, order-number & timestamp → payment-date)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

INGREDIENT (ingredient-ID, name, stock)

$F_{INGREDIENT}$(ingredient-ID → name, ingredient-ID → stock)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

works-at (employee-ID, factoty-ID, since)
$F_{\text{works-at}}$(employee-ID & factory-ID $\rightarrow$ since)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

includes (order-number, product-ID, amount)
F_{includes}(order-number & product-ID $\rightarrow$ amount)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

contains (product-ID, ingredient-ID, amount)
F_{contains}(product-ID & ingredient-ID $\rightarrow$ amount)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

supplies (ingredient-ID, tax-number, latest-price)
F_{supplies}(ingredient-ID & tax-number $\rightarrow$ latest-price)

1NF	Fulfilled
2NF	Fulfilled
3NF	Fulfilled
BCNF	Fulfilled

Every table, so the whole database is in Boyce-Codd normal from.

Trigger

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

I've written a trigger for DML purposes. Since Christmas is coming, my company decided to give a -5% discount from every order with a price bigger than 50000.
This trigger makes sure that the discount gets applied automatically, when a row gets inserted or updated.

```
CREATE OR REPLACE TRIGGER christmas_discount
BEFORE INSERT OR UPDATE OF price ON ORDER_DETAILS
FOR EACH ROW
BEGIN
     IF(:NEW.price > 50000) THEN
     :NEW.price := :NEW.price*0.95;
     END IF;
END;
```