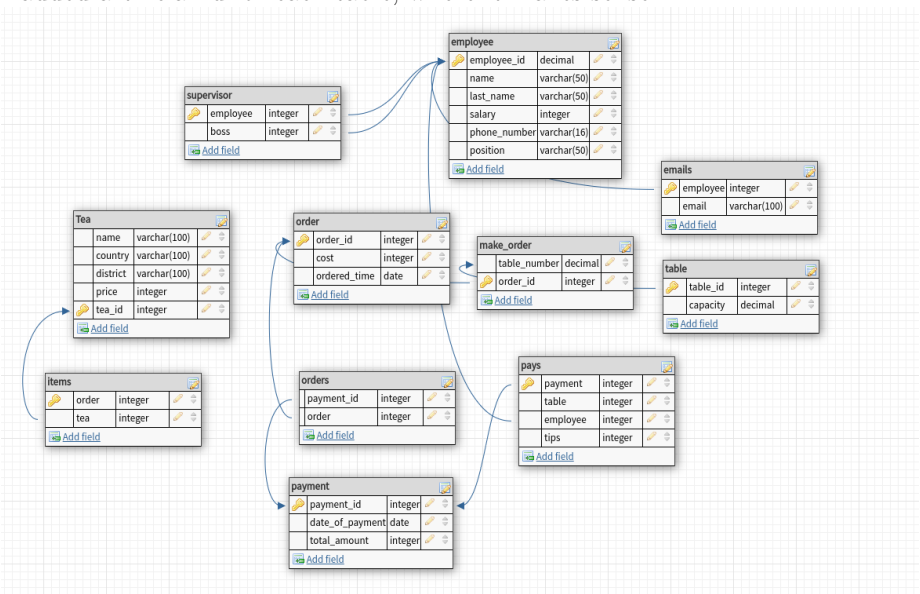
CP3 creating database

Danil Alshaev

April 2023

1 Scheme

I am going to use this scheme that I created in DB designer.
I added artificial id for each table, where it makes sense



2 Relational scheme

Relational scheme from CP2

tea(name, country, district,price)
order(order_number, price, ordered time)
Items(order number, name, country,district)
FK: (name, country, district) $\subset tea(name, country, district)$
$FK : (order\_number) \subset order(order\_number)$

$person(phone\_number, Birth\_number, name)$
$Emails(person, email)$
$FK : (person) \subset person(phone\_number, Birth\_number)$

$Employee(contract\_number, phone\_number, Birth\_number, salary, position)$
$FK : (contract\_number, phone\_number) \subset Person(contract\_number, phone\_number)$

$Supervisor(employee, boss)$:
$FK : (employee) \subset employee(contract\_number, phone\_number, Birth\_number)$
$FK : (boss) \subset employee(contract\_number, phone\_number, Birth\_number)$

$Take\_order(order\_number, employee)$:
$FK : (order) \subset order(order\_number)$
$FK : (employee) \subset employee(contract\_number, phone\_number, Birth\_number)$

$table(table\_number, capacity)$
$Make\_order(table\_number, order)$:
$FK : (table\_number) \subset table(table\_number)$
$FK : (order) \subset order(order\_number)$

$Served(employee, table)$
$FK : (employee) \subset employee(contract\_number, phone\_number, Birth\_number)$
$FK : (table) \subset table(table\_number)$

$payment(code, date, amount\_paid)$
$Contains(code, order)$
$FK : (code) \subset payment(code)$
$FK : (order) \subset order(order\_number)$

$pays(employee, table, payment, action, tip)$
$FK : (employee) \subset employee(contract\_number, phone\_number, Birth\_number)$
$FK : (table) \subset table(table\_number)$
$FK : (payment) \subset payment(code)$

3 Creating tables

I used generations, from DB designer, and added my constraints:

1. **Suitable types for individual attributes**

For country I used 3 symbols, to safe memory.
For ID almost everywhere I am using INTEGER.

```
CREATE TABLE IF NOT EXISTS public.tea
(
    name character varying(100) COLLATE pg_catalog."default" NOT NULL,
    country character varying(3) COLLATE pg_catalog."default" NOT NULL,
    district character varying(100) COLLATE pg_catalog."default",
    price integer NOT NULL,
    tea_id integer NOT NULL,
    CONSTRAINT "Tea_pk" PRIMARY KEY (tea_id),
    CONSTRAINT tea_district_key UNIQUE (district),
    CONSTRAINT tea_ch_price CHECK (price > 0)
)

CREATE TABLE IF NOT EXISTS public."table"
(
    table_id smallint NOT NULL,
    capacity integer NOT NULL,
    CONSTRAINT table_pk PRIMARY KEY (table_id)
)
```

2. **Integrity Constraints / Table integrity Constraints**

CHECK constraints are used for a few tables. For table employee checking whether salary is not less the minimum (arbitrary chosen number) and checking whether name and last name do make sense, if they have at least 2 letters

```
CREATE TABLE IF NOT EXISTS public.employee
(
    employee_id integer NOT NULL,
    name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    last_name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    salary integer NOT NULL,
    phone_number character varying(16) COLLATE pg_catalog."default" NOT NULL,
    "position" character varying(50) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT employee_pk PRIMARY KEY (employee_id),
    CONSTRAINT employee_ch_salary CHECK (salary > 8000 AND name::text ~~ '__%'::text AND last_name::text ~~ '___%'::text)
)

CREATE TABLE IF NOT EXISTS public.emails
(
    employee integer NOT NULL,
    email character varying(100) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT email_pk PRIMARY KEY (email, employee),
    CONSTRAINT emails_ch_email CHECK (email::text ~~ '_%@_%.__%'::text)
)
```

Also checking emails, if they have essential symbols as @, and ends with a dot and at least 2 letters. Emails have table constraint for as primary key for email it self and employee

3. **Foreign keys including the ON UPDATE/DELETE directive**

Used for a table connections. Important that, if the order have been deleted, it will be NULL in other tables. Changed emails are going to be changed in other, tables by using UPDATE ON CASCADE

```sql
CREATE TABLE IF NOT EXISTS public.items
(
    "order" integer NOT NULL,
    tea integer NOT NULL,
    CONSTRAINT items_fk0 FOREIGN KEY (tea)
        REFERENCES public.tea (tea_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT items_fk1 FOREIGN KEY ("order")
        REFERENCES public."order" (order_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE SET NULL
)
```

4 SQL queries

1. **Inner join and data condition**
For the 1 query I would like to join some information
for the teas that I want. I want to now about teas in the
orders that are more expensive then 60 Kč, for that
I am using data condition that cost is bigger then 60, also
I want orders from Indonesia or Japan, for some statistical purpose.

```sql
SELECT tea.name, tea.country, tea.district, tea.price, items.order
FROM public.tea tea
JOIN public.items items ON tea.tea_id = items.tea
WHERE (tea.price > 60) AND ((tea.country = 'IN') OR (tea.country = 'JP'));
LIMIT 5000;
```

Here is the following output from my database:

	name character varying (100)	country character varying (3)	district character varying (100)	price integer	order integer
1	Matcha	JP	Okinawa	150	21854
2	Chai	IN	Jakarta	80	19264
3	Gyokuro	JP	Uji	250	27233
4	Matcha	JP	Okinawa	150	246
5	Gyokuro	JP	Uji	250	15237
6	Hojicha	JP	Kyoto	90	27260
7	Darjeeling	IN	Darjeeling	70	22688
8	Darjeeling	IN	Darjeeling	70	2574
9	Sencha	JP	Tokyo	80	20554
10	Chai	IN	Jakarta	80	5422
11	Chai	IN	Jakarta	80	2033
12	Gyokuro	JP	Uji	250	30477
13	Darjeeling	IN	Darjeeling	70	1149

2. **Outer join**

For the 2 query I will use **LEFT OUTER JOIN**

I want to join tips and employee from the table "pays" and other information about payment,

payment doesn't count tips and now joining tables

I can get statistics how much tips for a total amount from payment

employer can expect

```
--Outer join
SELECT pays.payment, pays.table, pays.employee, pays.tips, payment.total_amount
FROM pays
LEFT OUTER JOIN payment ON pays.payment = payment.payment_id;
```

Here is the output:

	payment integer	table integer	employee integer	tips integer	total_amount integer
1	1	11	3	[null]	1697
2	2	34	8	217	3563
3	3	39	7	172	339
4	4	7	5	181	2629
5	5	40	4	139	1822
6	6	37	6	194	1862
7	7	28	6	144	3569
8	8	22	6	29	3049
9	9	2	3	180	262
10	10	17	4	234	908
11	11	30	3	293	4997
12	12	37	5	30	877
13	13	38	4	[null]	1757

3. **Aggregation**

For the 3 query I would like to know about orders costs during 1 working month from 2022.05 to 2022.06, I want to know what orders were more common and had the same cost and what orders are less common.

```
--Agregation
SELECT "cost", SUM("cost") as total_cost, COUNT(*) as total_orders
FROM public."order"
WHERE ordered_time >= '2022-05-01' AND ordered_time <= '2022-06-01'
GROUP BY "cost" HAVING (total_orders > 1)
```

And here is the output:

	cost integer	total_cost bigint	total_orders bigint
1	790	1580	2
2	1798	1798	1
3	1489	5956	4
4	1269	3807	3
5	1989	5967	3
6	652	652	1
7	273	273	1
8	51	51	1
9	1560	3120	2
10	1587	3174	2
11	1091	2182	2
12	951	1902	2
13	839	839	1

4. **Sorting and paging**

Limiting too big queries, for not waiting so long in a very big database

Ordering by total amount, from the lowest to the biggest

```
SELECT tea.name, tea.country, tea.district, tea.price, items.order
FROM public.tea tea
JOIN public.items items ON tea.tea_id = items.tea
WHERE (tea.price > 60) AND ((tea.country = 'IN') OR (tea.country = 'JP'));
LIMIT 5000;

SELECT payment_id, total_amount, COUNT(*) AS frequency
FROM payment
GROUP BY total_amount, payment_id
ORDER BY total_amount ASC;
```

here is the output for 2 query, the 1 query can be seen in part 1:

	payment_id [PK] integer	total_amount integer	frequency bigint
1	4474	50	1
2	773	51	1
3	2282	52	1
4	1950	52	1
5	1850	52	1
6	5205	53	1
7	3078	55	1
8	7314	55	1
9	1215	55	1
10	638	56	1
11	863	56	1
12	1610	58	1
13	989	58	1

5. **Set operations**

For set operations I am going to use UNION, because I don't want duplicates. Actually my query is pretty much the same as Outer Join from (2), Again I am integrating following tables as "pays" and "payment" and get tips and total amount:

```
SELECT payment AS payment_id, tips, NULL AS total_amount
FROM pays
UNION
SELECT payment_id,NULL AS tips, total_amount
FROM payment
ORDER BY payment_id LIMIT 100;
```

Here is the output:

	payment_id integer	tips integer	total_amount integer
1	1	[null]	[null]
2	1	[null]	1697
3	2	217	[null]
4	2	[null]	3563
5	3	172	[null]
6	3	[null]	339
7	4	181	[null]
8	4	[null]	2629
9	5	139	[null]
10	5	[null]	1822
11	6	[null]	1862
12	6	194	[null]
13	7	144	[null]

6. **Nested SELECT**

For this query I am going to modify my query from (1) Now I want to specify districts, not only country, in order to do that, I am going to use nested SELECT as joining my first tables on data conditions. I am choosing as districts 'Tokyo' and 'Jakarta':

```
SELECT t.name, t.country, t.district, t.price, i.order
FROM (
  SELECT name, country, district, price, tea_id
  FROM tea
  WHERE (price > 60) AND (country IN ('IN', 'JP'))
) t
JOIN items i ON t.tea_id = i.tea
WHERE t.district IN ('Tokyo', 'Jakarta')
ORDER BY i.order;
```

Here is the output, as you can see the difference from (1) is has only certain districts for orders

	name character varying (100)	country character varying (3)	district character varying (100)	price integer	order integer
1	Chai	IN	Jakarta	80	1
2	Sencha	JP	Tokyo	80	37
3	Chai	IN	Jakarta	80	43
4	Sencha	JP	Tokyo	80	45
5	Chai	IN	Jakarta	80	52
6	Sencha	JP	Tokyo	80	58
7	Chai	IN	Jakarta	80	90
8	Chai	IN	Jakarta	80	99
9	Chai	IN	Jakarta	80	115
10	Sencha	JP	Tokyo	80	116
11	Chai	IN	Jakarta	80	156
12	Chai	IN	Jakarta	80	178
13	Chai	IN	Jakarta	80	181

5 Additional

Data were generated by python scripts, using external libraries, 2 tables have 32000 rows, and other have from 10 to 8000 rows where it makes sense. All code for table creation, I am going to paste here, in order not to make pdf unreadable.

```
CREATE TABLE IF NOT EXISTS public.tea
(
    name character varying(100) COLLATE pg_catalog."default" NOT NULL,
    country character varying(3) COLLATE pg_catalog."default" NOT NULL,
    district character varying(100) COLLATE pg_catalog."default",
    price integer NOT NULL,
    tea_id integer NOT NULL,
    CONSTRAINT "Tea_pk" PRIMARY KEY (tea_id),
    CONSTRAINT tea_district_key UNIQUE (district),
    CONSTRAINT tea_ch_price CHECK (price > 0)
)

CREATE TABLE IF NOT EXISTS public."table"
(
    table_id smallint NOT NULL,
    capacity integer NOT NULL,
    CONSTRAINT table_pk PRIMARY KEY (table_id)
)

CREATE TABLE IF NOT EXISTS public.pays
(
    payment integer NOT NULL,
    "table" integer NOT NULL,
    employee integer NOT NULL,
    tips integer,
    CONSTRAINT pays_by_pk PRIMARY KEY (payment),
    CONSTRAINT pays_ch_tips CHECK (tips = NULL::integer OR tips > 0)
)

CREATE TABLE IF NOT EXISTS public.payment
(
    payment_id integer NOT NULL,
    total_amount integer NOT NULL,
    CONSTRAINT payment_pk PRIMARY KEY (payment_id)
)
```

```
CREATE TABLE IF NOT EXISTS public.orders_made_by
(
    table_number smallint NOT NULL,
    order_id integer NOT NULL,
    CONSTRAINT orders_made_by_pk PRIMARY KEY (order_id),
    CONSTRAINT orders_made_by_order_id_key UNIQUE (order_id)
)

CREATE TABLE IF NOT EXISTS public.orders
(
    payment_id integer NOT NULL,
    "order" integer NOT NULL,
    CONSTRAINT orders_pk PRIMARY KEY (payment_id, "order")
)

CREATE TABLE IF NOT EXISTS public."order"
(
    order_id integer NOT NULL,
    cost integer NOT NULL,
    ordered_time date,
    CONSTRAINT order_pk PRIMARY KEY (order_id),
    CONSTRAINT order_ch_cost CHECK (cost > 0)
)

CREATE TABLE IF NOT EXISTS public.items
(
    "order" integer NOT NULL,
    tea integer NOT NULL,
    CONSTRAINT items_fk0 FOREIGN KEY (tea)
        REFERENCES public.tea (tea_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT items_fk1 FOREIGN KEY ("order")
        REFERENCES public."order" (order_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE SET NULL
)
```

```
CREATE TABLE IF NOT EXISTS public.orders_made_by
(
    table_number smallint NOT NULL,
    order_id integer NOT NULL,
    CONSTRAINT orders_made_by_pk PRIMARY KEY (order_id),
    CONSTRAINT orders_made_by_order_id_key UNIQUE (order_id)
)

CREATE TABLE IF NOT EXISTS public.orders
(
    payment_id integer NOT NULL,
    "order" integer NOT NULL,
    CONSTRAINT orders_pk PRIMARY KEY (payment_id, "order")
)

CREATE TABLE IF NOT EXISTS public."order"
(
    order_id integer NOT NULL,
    cost integer NOT NULL,
    ordered_time date,
    CONSTRAINT order_pk PRIMARY KEY (order_id),
    CONSTRAINT order_ch_cost CHECK (cost > 0)
)

CREATE TABLE IF NOT EXISTS public.items
(
    "order" integer NOT NULL,
    tea integer NOT NULL,
    CONSTRAINT items_fk0 FOREIGN KEY (tea)
        REFERENCES public.tea (tea_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT items_fk1 FOREIGN KEY ("order")
        REFERENCES public."order" (order_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE SET NULL
)
```