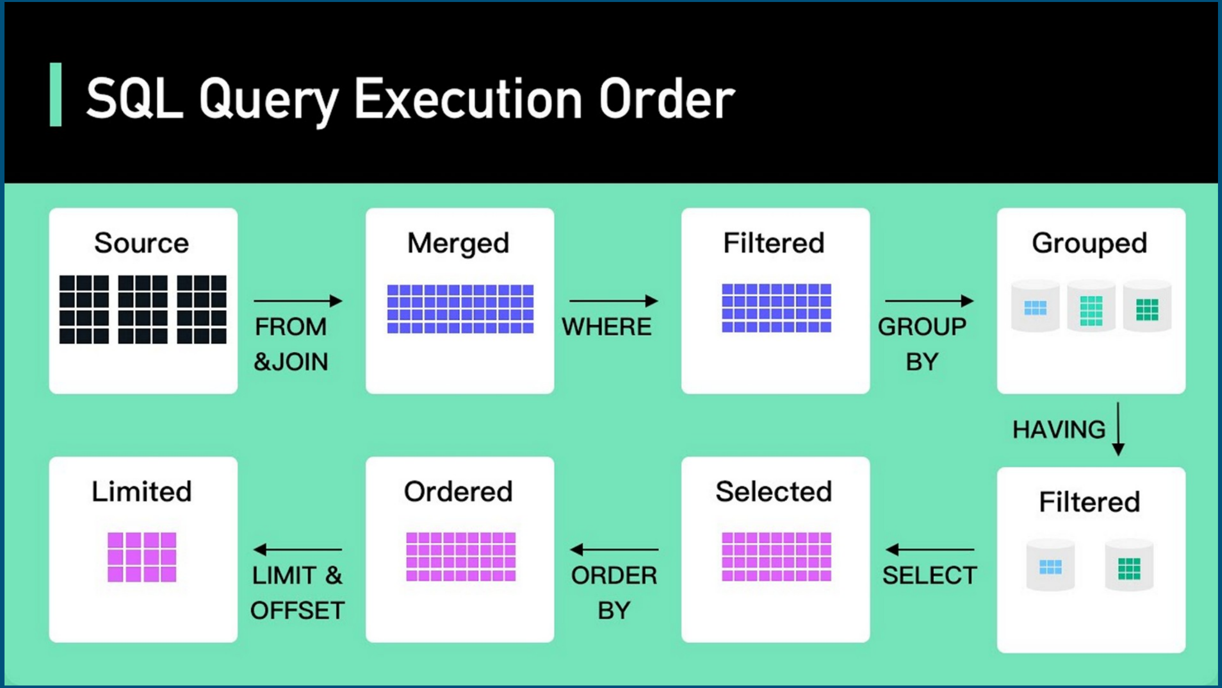
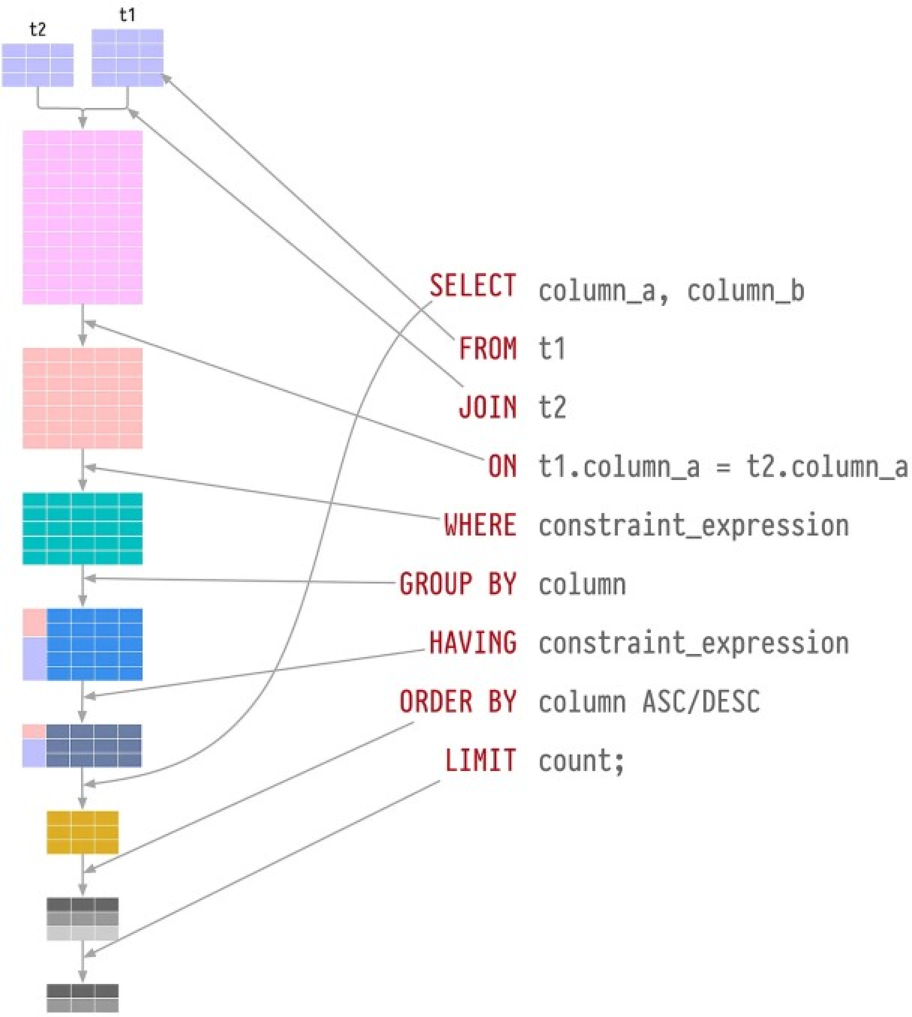
Databases

Lecture 4
DML. Database design.

Reminder: the order of execution

1. FROM & JOIN
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT & OFFSET





```sql
SELECT column_a, column_b
  FROM t1
  JOIN t2
    ON t1.column_a = t2.column_a
 WHERE constraint_expression
GROUP BY column
HAVING constraint_expression
ORDER BY column ASC/DESC
 LIMIT count;
```

DML: aggregation functions

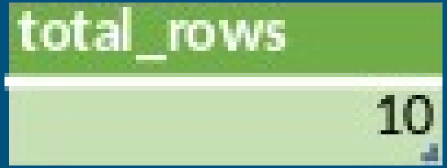
- An aggregate function reduces many input values to a single output value, such as a sum or average
- Most aggregate functions ignore NULL values, so rows for which expressions produce one or more NULL values are discarded

SELECT: using aggregation functions

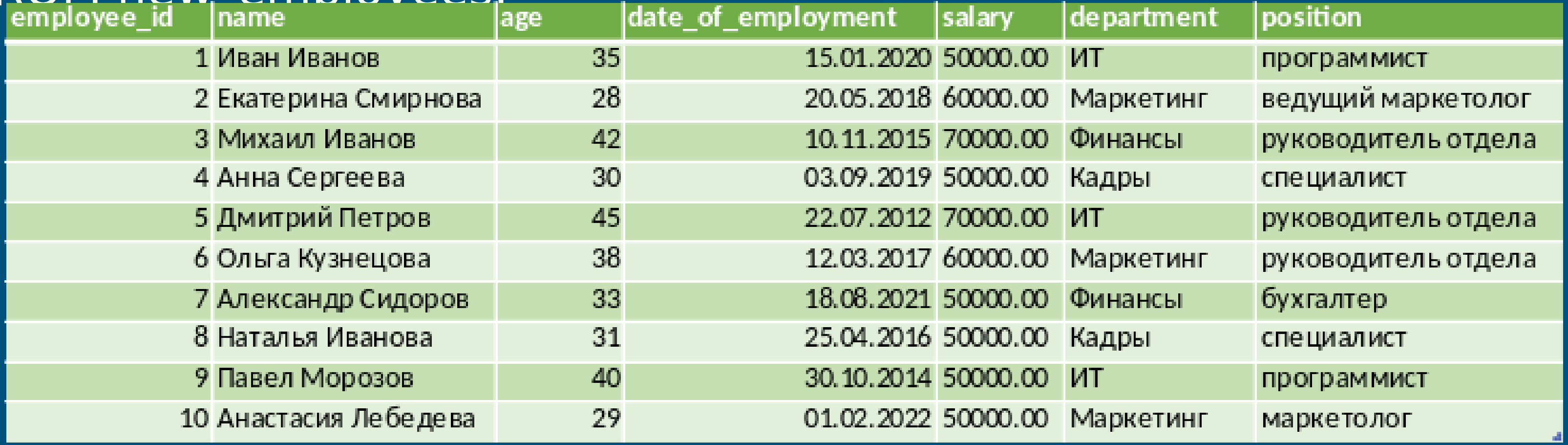
- **count() -> bigint**
  the number of records with a known value. If you need to count the number of unique values, you can use count(DISTINCT field_nm)
- **max() -> same type as input**
  the largest of all selected field values (!= NULL)
- **min() -> same type as input**
  the smallest of all selected field values (!= NULL)
- **sum() -> numeric**
  sum of all selected field values (!= NULL)
- avg() **-> numeric**
  average of all selected field values (!= NULL)

Count

```
SELECT count(*) as
total_rows
    FROM new_employees;
```



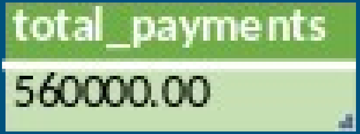
total_rows
10



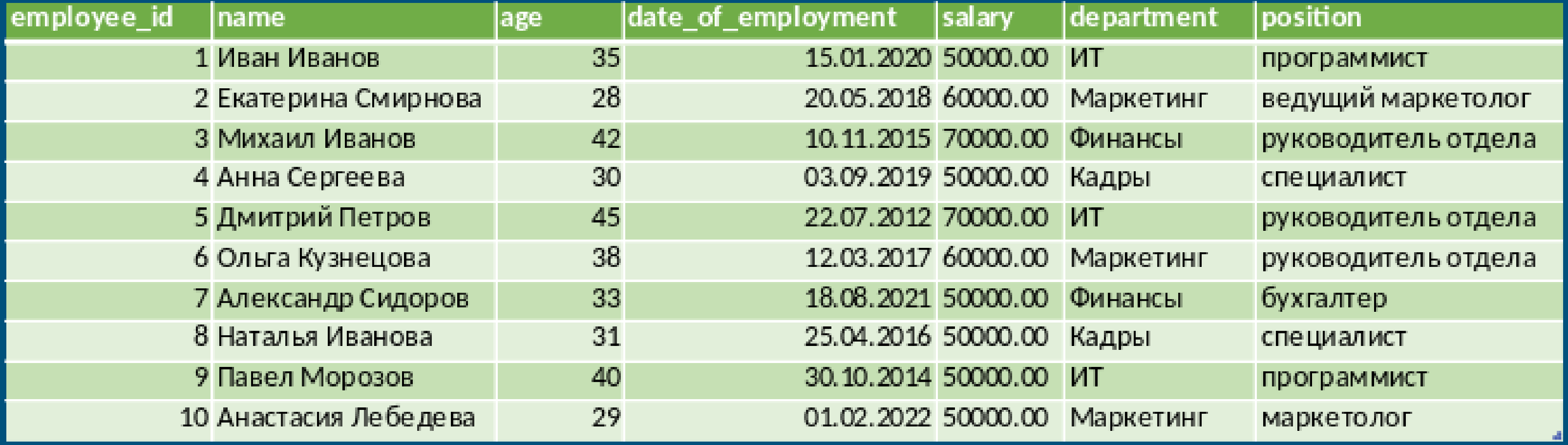
employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

Sum

```
SELECT sum(salary) as
total_payments
    FROM new_employees;
```



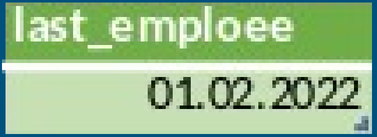
total_payments
560000.00



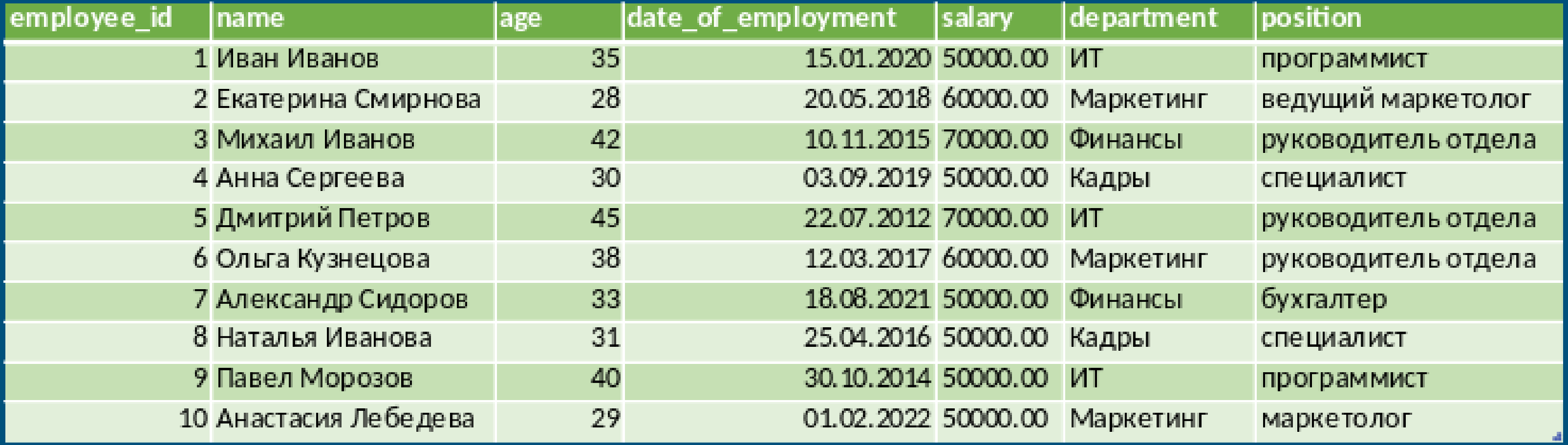
employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

Max

```
SELECT
max(date_of_employment)
AS last_emploee
    FROM new_employees;
```



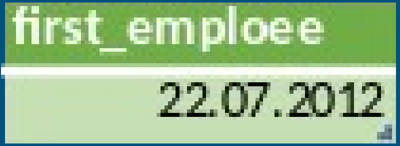
last_emploee
01.02.2022



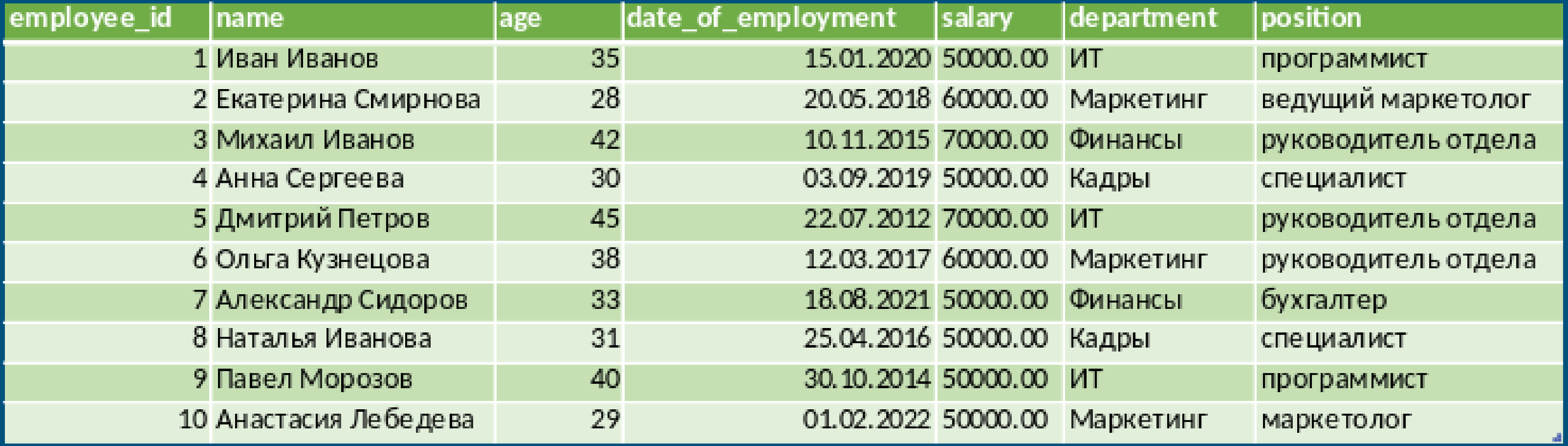
employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

Min

```
SELECT
min(date_of_employment)
AS first_emploee
    FROM new_employees;
```



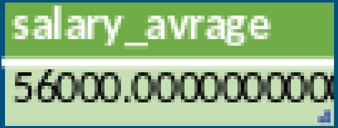
first_emploee
22.07.2012



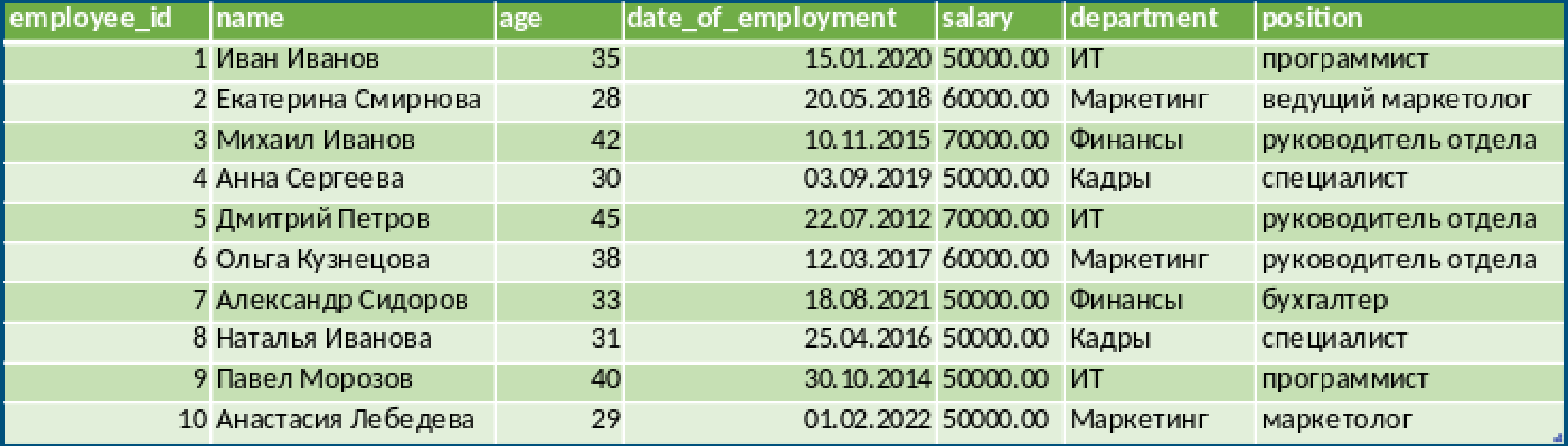
employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

Avg

```
SELECT avg(salary) as
salary_average
    FROM new_employees;
```



salary_avrage
56000.000000000



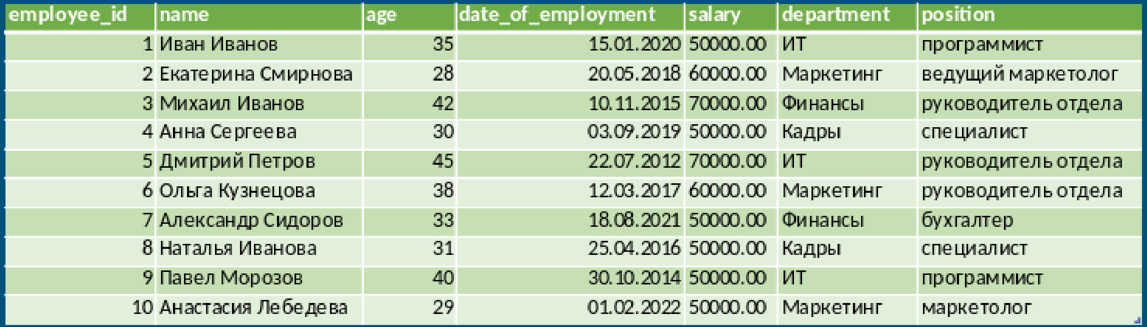
employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

SELECT: GROUP BY + aggregation

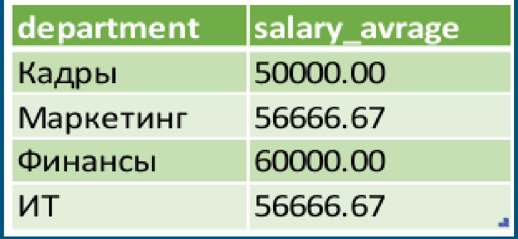
- When a GROUP BY clause or any aggregate function is present in a query, expressions in the SELECT list generally cannot access non-groupable columns because otherwise the non-groupable column would have to return more than one possible value
- Aggregate functions, when used, are evaluated over all rows that make up each group and ultimately produce a separate value for each group

SELECT: example

SELECT department,
ROUND(AVG(salary), 2)
as salary_average
FROM new_employees
GROUP BY department;



employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог



department	salary_avrage
Кадры	50000.00
Маркетинг	56666.67
Финансы	60000.00
ИТ	56666.67

SUBQUERIES

A subquery is a query contained in another SQL expression (we will call it a containing expression)

SUBQUERIES

scalar

non-scalar

SUBQUERIES

unrelated

linked

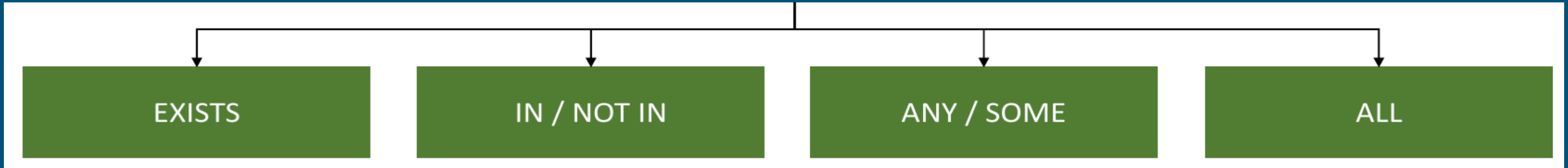
SUBQUERIES

- Unrelated, i.e. completely self-sufficient and independent of the main query
  Are executed before executing the containing expression.

- Linked, i.e. refer to the columns of the main query.
  The use of aliases is useful for writing such queries.
  For cases when the same table is used in the main query and in the subquery the use of aliases is mandatory!
  Are executed for each line containing expressions.

Expressions (predicates) of subqueries

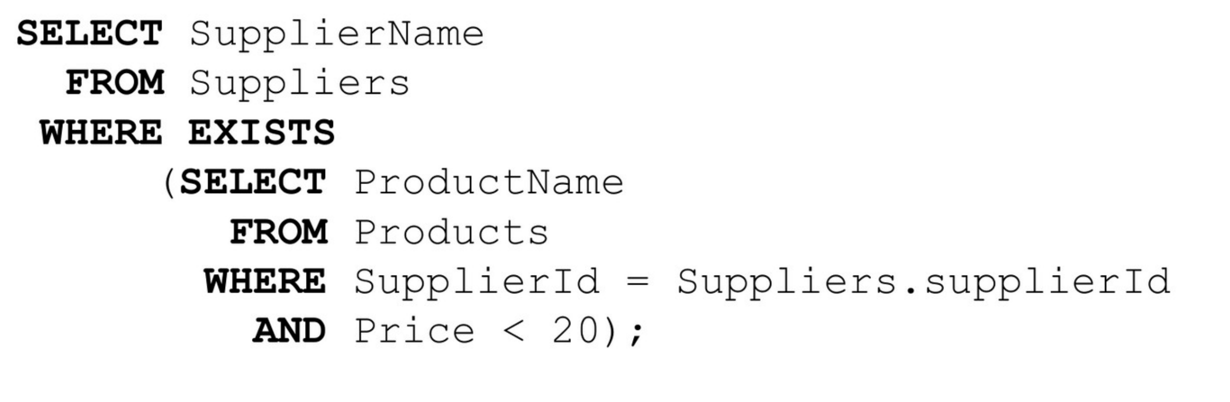
If the subquery is non-scalar (returns many values), then special predicates can be applied to it from the outer query:



SUBQUERIES: EXISTS

- **EXISTS(subquery)**
- After executing a query, the system checks whether it returns rows in the result. If it returns at least one row, the EXISTS result will be "true", and if it returns none, the result will be "false"
- The subquery may not be executed completely, but will end as soon as at least one row is returned. Therefore, “side effects” should be avoided in subqueries
- When using the EXISTS predicate, we are not interested in the return value, so in the subquery after the SELECT, as a rule, you can specify an arbitrary value

EXISTS: example



```
SELECT SupplierName
  FROM Suppliers
 WHERE EXISTS
      (SELECT ProductName
         FROM Products
        WHERE SupplierId = Suppliers.supplierId
          AND Price < 20);
```

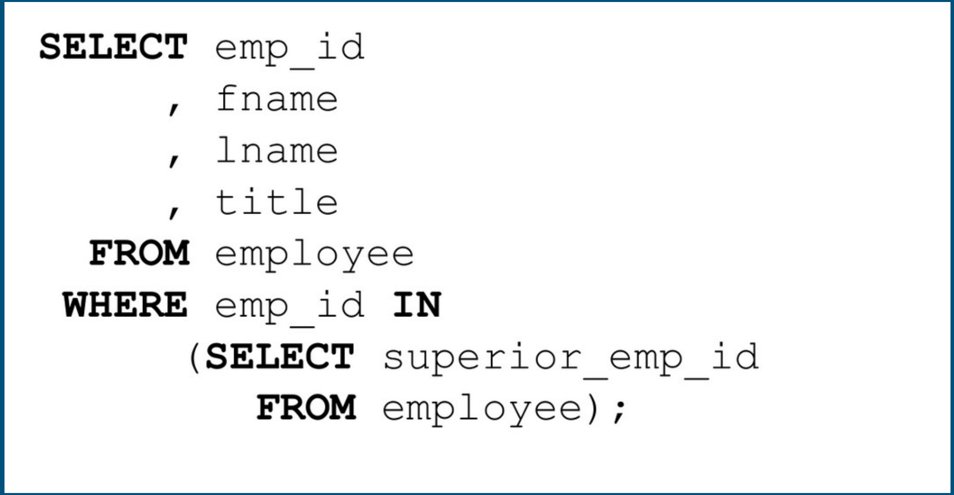
SUBQUERIES: IN / NOT IN

- **IN expression (subquery)**
- On the right side of this expression, a subquery is specified in parentheses, which must return exactly one column
- The evaluated value of the left expression is compared with the values in all rows returned by the subquery. The result of the entire IN expression will be “true” if a row with that value is found, and “false” otherwise (including when the subquery returns no rows at all).
- The entire NOT IN expression will result in "true" if only non-matching rows are found (including when the subquery returns no rows at all). If there is at least one matching line, the result will be “false”.

SUBQUERIES: IN / NOT IN

- **IN expression (subquery)**
- If the result of the expression on the left is NULL or there are no equal values on the right, and at least one of the values on the right is NULL, the IN construct returns NULL
- The request may not be completed completely!

EXAMPLE: IN / NOT IN



```
SELECT emp_id
     , fname
     , lname
     , title
  FROM employee
 WHERE emp_id IN
       (SELECT superior_emp_id
          FROM employee);
```

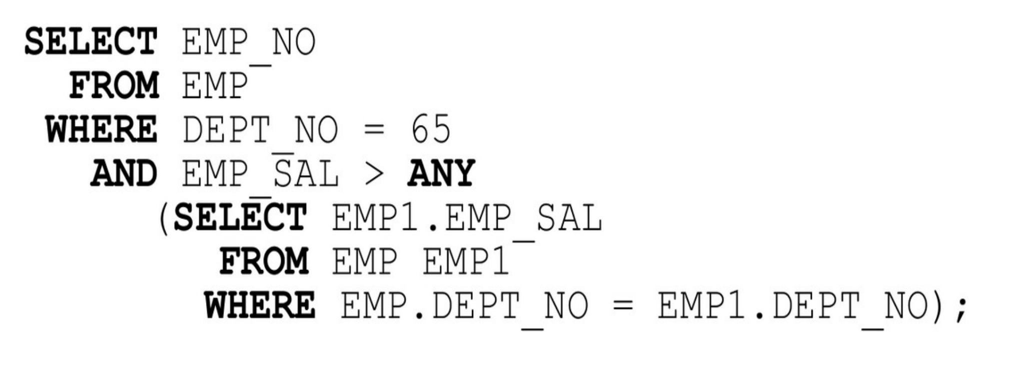
SUBQUERIES: ANY / SOME

- **expression operator ANY (subquery)**
- **expression operator SOME (subquery)**
- On the right side of the construction, a subquery is written in parentheses, which must return exactly one column. The evaluated value of the left expression is compared with the value in each row of the subquery result using the specified condition operator, which must return a Boolean value
- The result of ANY will be “true” if the condition is true for at least one row, and “false” otherwise (including when the subquery does not return any rows)

SUBQUERIES: ANY / SOME

- The SOME keyword is a synonym for ANY. The IN construct can also be written as = ANY
- If the condition is not satisfied for any of the rows, and for at least one row the conditional operator returns NULL, the ANY construct returns NULL, not false
- The request may not be completed completely!

SUBQUERIES: ANY / SOME

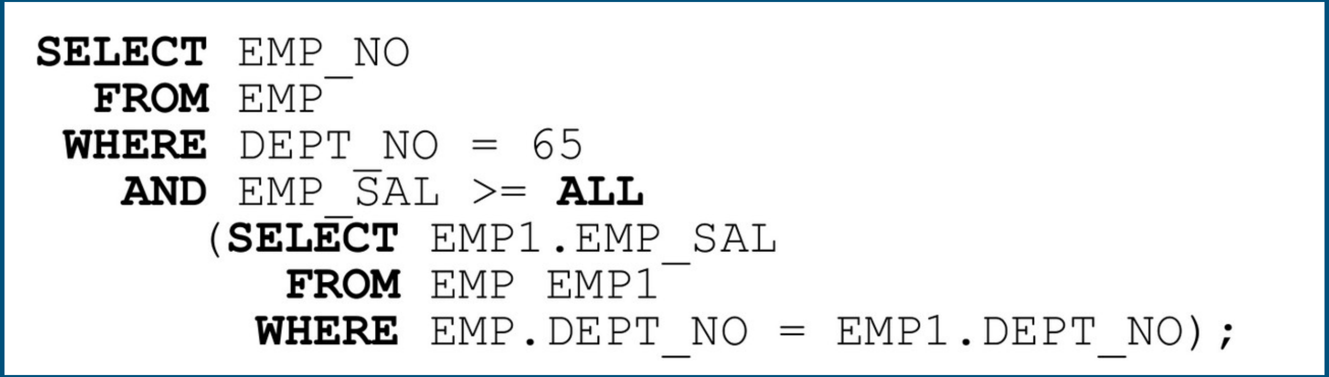


```
SELECT EMP_NO
  FROM EMP
 WHERE DEPT_NO = 65
   AND EMP_SAL > ANY
      (SELECT EMP1.EMP_SAL
         FROM EMP EMP1
        WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

SUBQUERIES: ALL

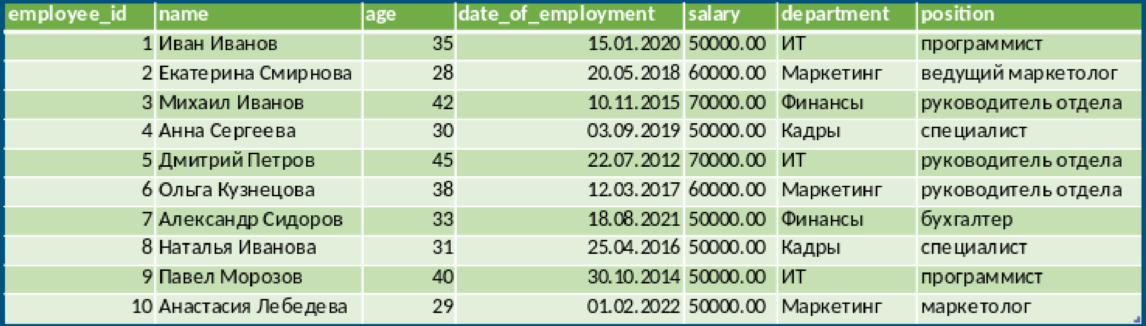
- **expression operator ALL (subquery)**
- On the right side of the construction, a subquery is written in parentheses, which must return exactly one column. The evaluated value of the left expression is compared with the value in each row of the subquery result using a specified conditional operator, which must return a Boolean value.
- The result of ALL will be “true” if the condition is true for all rows (and when the subquery returns no rows), or “false” if there are rows for which it is false. The result of the expression will be NULL if for none of the subquery rows the comparison result is true, and the minimum for one is NULL
- The NOT IN construction is equivalent to <> ALL
- The request may not be completed completely!

SUBQUERIES: ALL

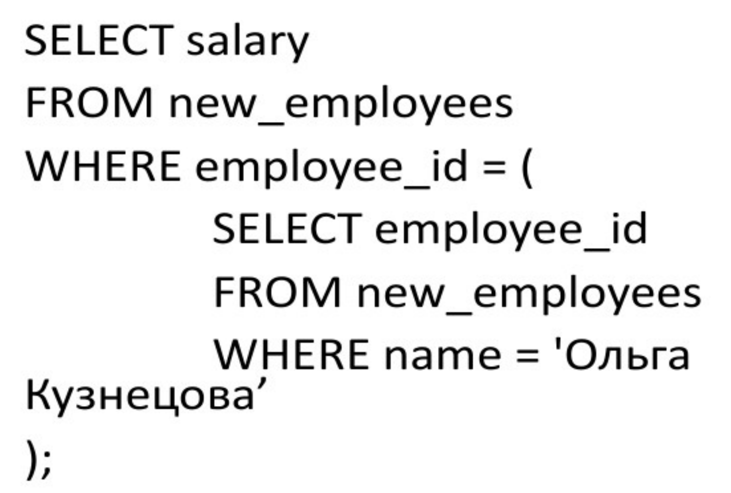


```
SELECT EMP_NO
  FROM EMP
 WHERE DEPT_NO = 65
   AND EMP_SAL >= ALL
      (SELECT EMP1.EMP_SAL
         FROM EMP EMP1
        WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

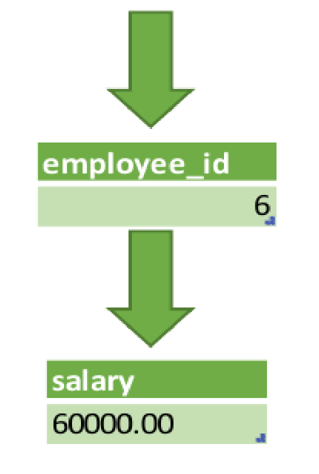
Example: scalar



employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог



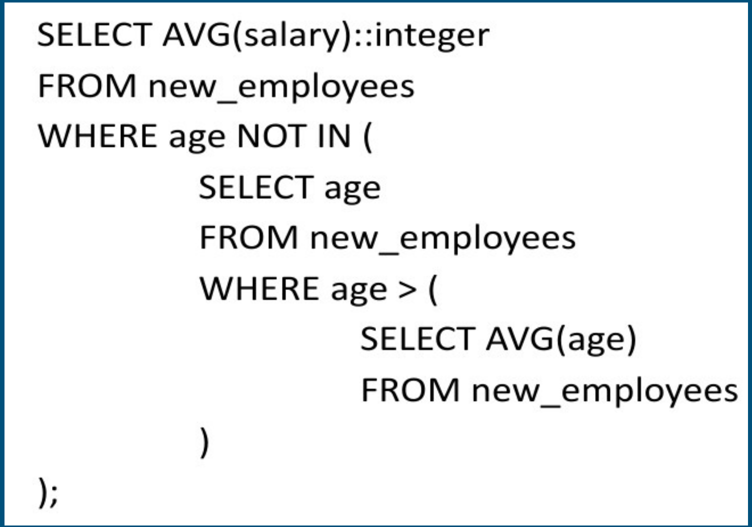
```
SELECT salary
FROM new_employees
WHERE employee_id = (
            SELECT employee_id
            FROM new_employees
            WHERE name = 'Ольга
Кузнецова’
);
```



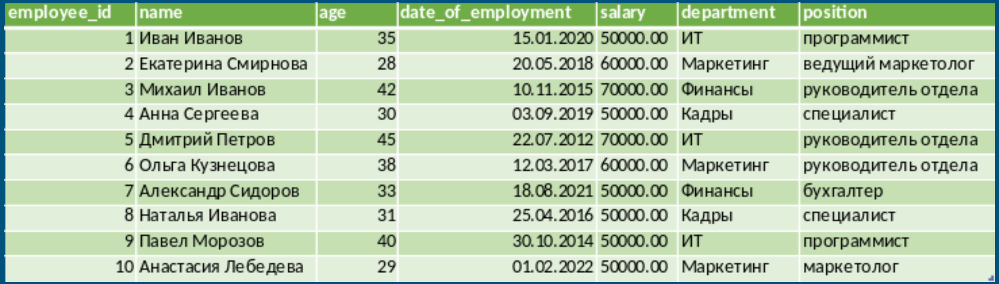
employee_id
6

salary
60000.00

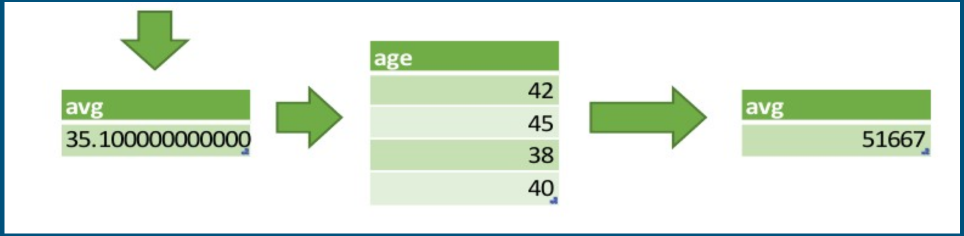
Example: non-scalar



```
SELECT AVG(salary)::integer
FROM new_employees
WHERE age NOT IN (
            SELECT age
            FROM new_employees
            WHERE age > (
                        SELECT AVG(age)
                        FROM new_employees
            )
);
```



employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог



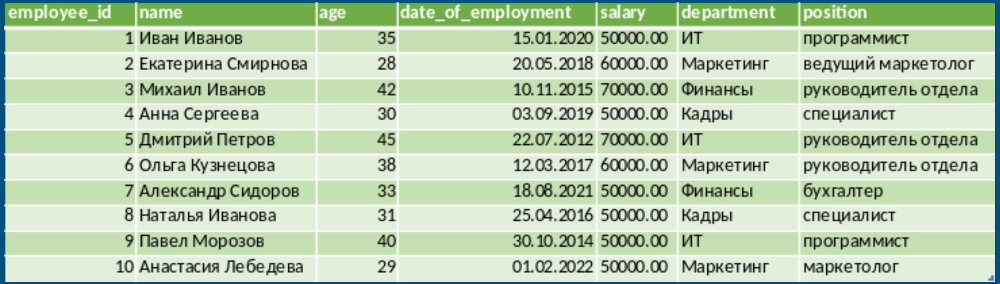
avg
35.100000000000

age
42
45
38
40

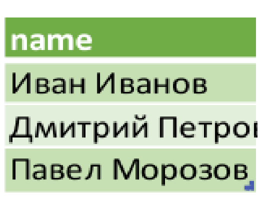
avg
51667

Example: non-scalar

```
SELECT name
FROM new_employees t1
WHERE EXISTS (
    SELECT 1
    FROM new_employees t2
    WHERE t1.department = 'ИТ'
            AND t2.department = 'ИТ’
);
```

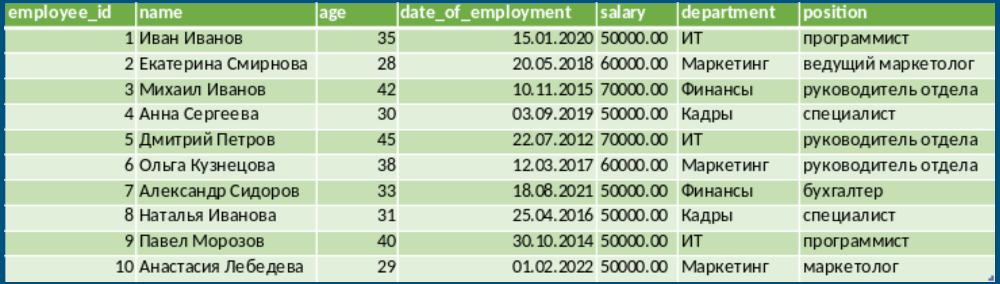


employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

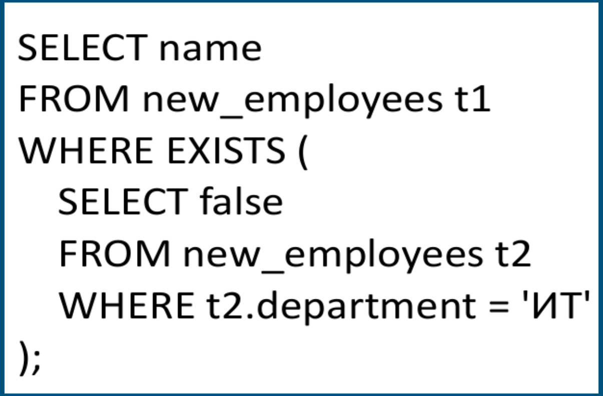


name
Иван Иванов
Дмитрий Петро
Павел Морозов

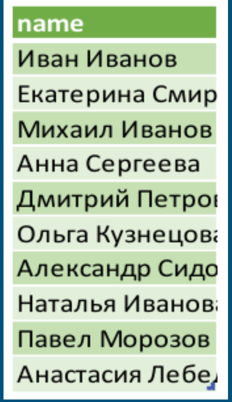
Example: non-scalar, incorrect



employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

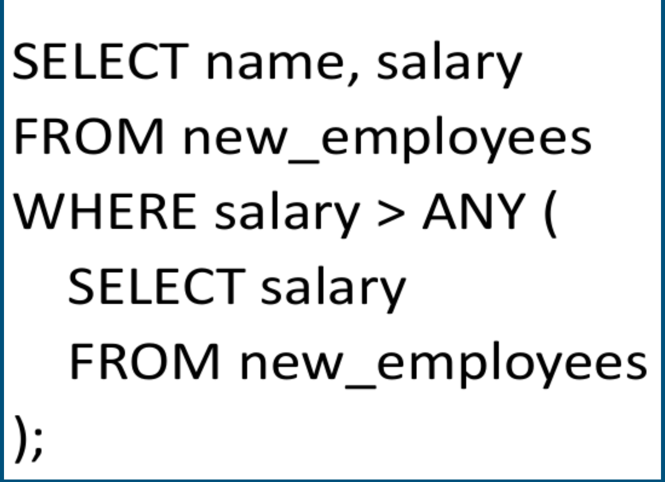


```
SELECT name
FROM new_employees t1
WHERE EXISTS (
    SELECT false
    FROM new_employees t2
    WHERE t2.department = 'ИТ'
);
```

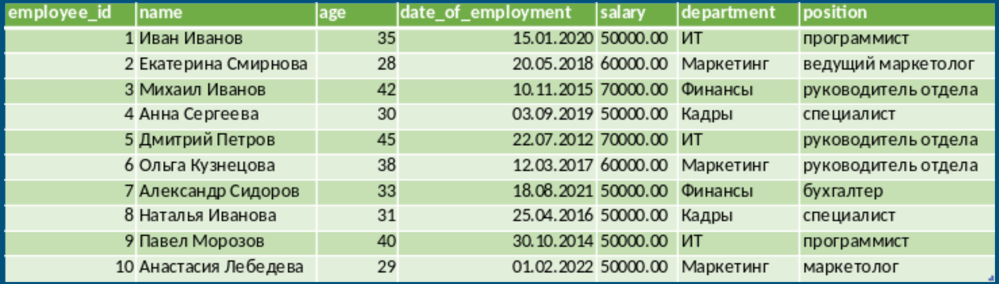


name
Иван Иванов
Екатерина Смир
Михаил Иванов
Анна Сергеева
Дмитрий Петро
Ольга Кузнецов
Александр Сидо
Наталья Иванов
Павел Морозов
Анастасия Лебе

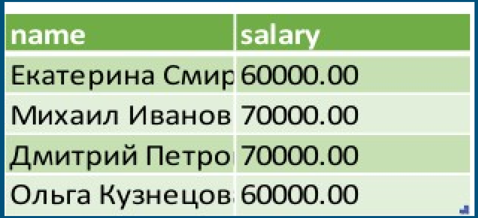
Example: non-scalar



```
SELECT name, salary
FROM new_employees
WHERE salary > ANY (
    SELECT salary
    FROM new_employees
);
```

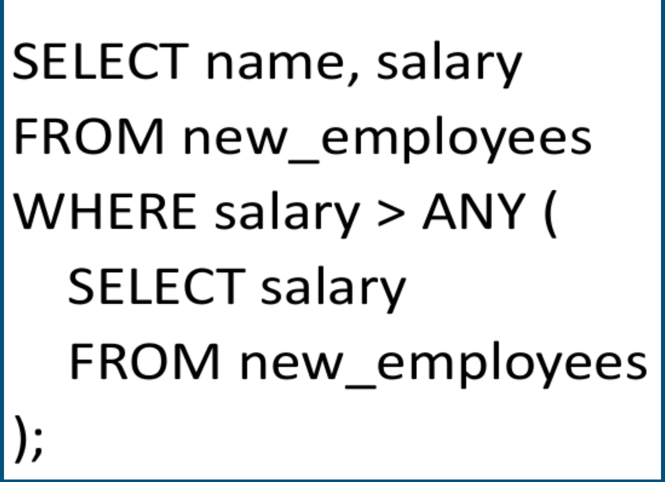


employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

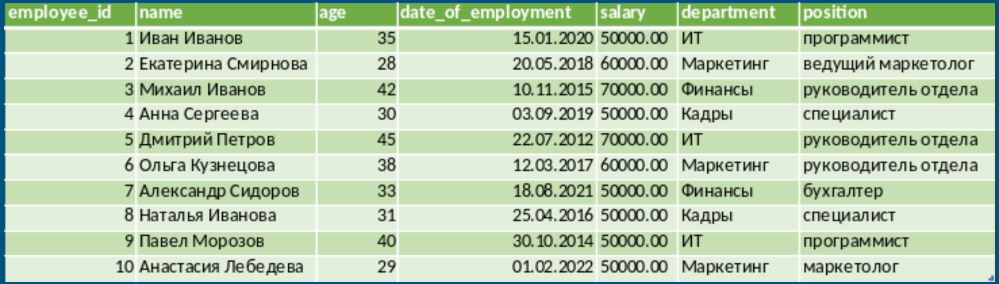


name	salary
Екатерина Смир	60000.00
Михаил Иванов	70000.00
Дмитрий Петро	70000.00
Ольга Кузнецов	60000.00

Example: non-scalar

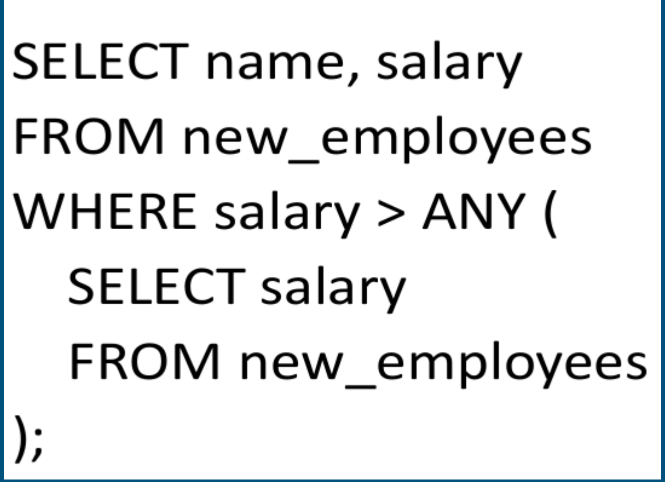


```
SELECT name, salary
FROM new_employees
WHERE salary > ANY (
    SELECT salary
    FROM new_employees
);
```

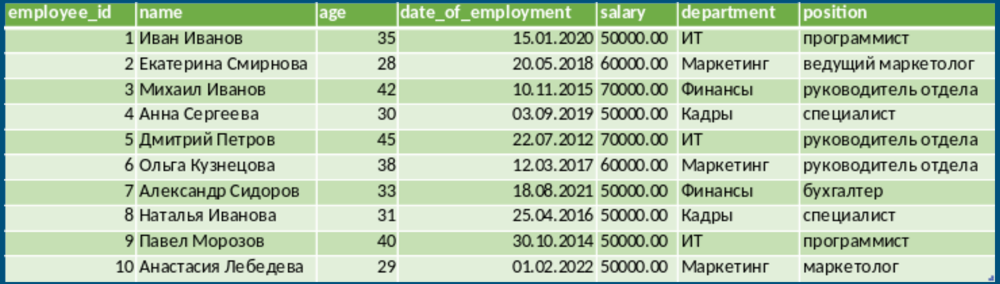


employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

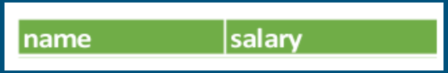
Example: non-scalar



```
SELECT name, salary
FROM new_employees
WHERE salary > ANY (
   SELECT salary
   FROM new_employees
);
```

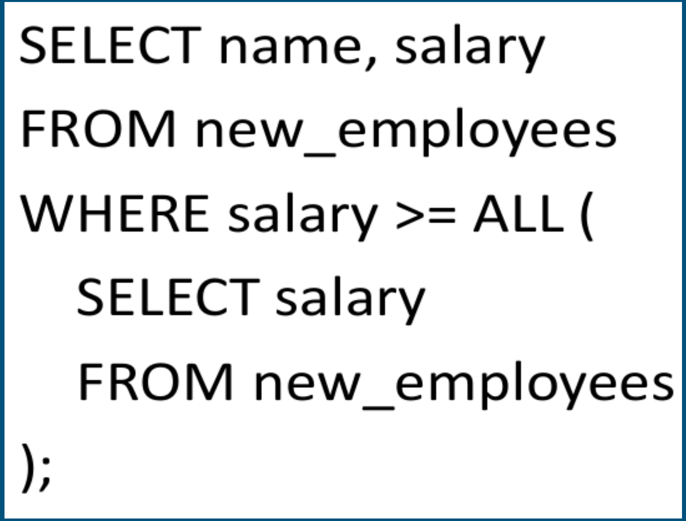


employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог

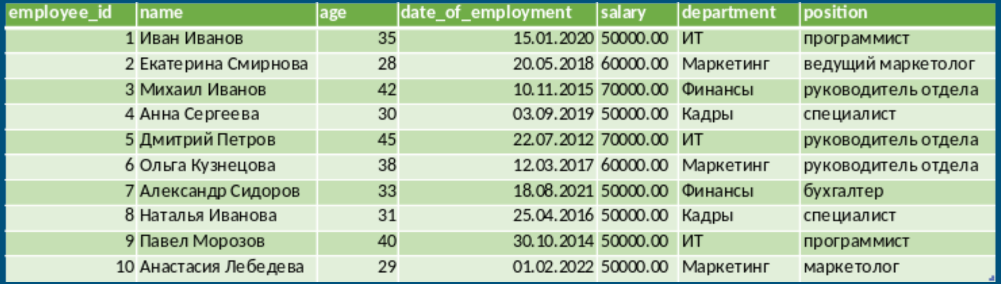


name	salary

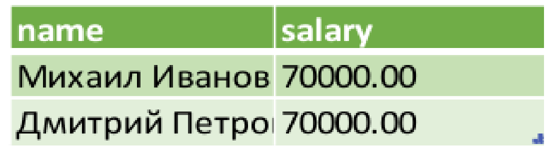
Example: non-scalar



```
SELECT name, salary
FROM new_employees
WHERE salary >= ALL (
   SELECT salary
   FROM new_employees
);
```



employee_id	name	age	date_of_employment	salary	department	position
1	Иван Иванов	35	15.01.2020	50000.00	ИТ	программист
2	Екатерина Смирнова	28	20.05.2018	60000.00	Маркетинг	ведущий маркетолог
3	Михаил Иванов	42	10.11.2015	70000.00	Финансы	руководитель отдела
4	Анна Сергеева	30	03.09.2019	50000.00	Кадры	специалист
5	Дмитрий Петров	45	22.07.2012	70000.00	ИТ	руководитель отдела
6	Ольга Кузнецова	38	12.03.2017	60000.00	Маркетинг	руководитель отдела
7	Александр Сидоров	33	18.08.2021	50000.00	Финансы	бухгалтер
8	Наталья Иванова	31	25.04.2016	50000.00	Кадры	специалист
9	Павел Морозов	40	30.10.2014	50000.00	ИТ	программист
10	Анастасия Лебедева	29	01.02.2022	50000.00	Маркетинг	маркетолог



name	salary
Михаил Иванов	70000.00
Дмитрий Петро	70000.00

DATABASE DESIGN

– the process of creating a detailed database data model, as well as the necessary

integrity constraints.

- A data model is an abstract model that:
- Organizes data elements
- Describes how they interact with each other
- Describes how they interact with objects of the outside world

DESIGN GOALS

- keeping all the necessary information
- reducing the redundancy and data duplication
- obtaining the necessary data upon the request
- ensuring database integrity

BASIC DESIGN STEPS

- Definition of the subject area and the data to be stored in the database
- Defining relationships between different data elements
- Imposing a logical structure on the data based on certain ratios
- Creating a designed database taking into account the features of the DBMS used

STAGES OF DATABASE DESIGN

- Conceptual design (infological)
- Logical design (datalogical)
- Physical design

STEP 1 DATA DEFINITION

- What subject area are we describing?
  - Full database of goods of Pyaterochka stores
  - Full customer base of Tinkoff Bank
  - MIPT educational processes
  - etc.

CONCEPTUAL DESIGN

A conceptual data model is a description of the main objects and the relationships between them.

STEP 2 DEFINING RELATIONSHIPS

We defined the subject area and highlighted the entities

Relationships need to be established

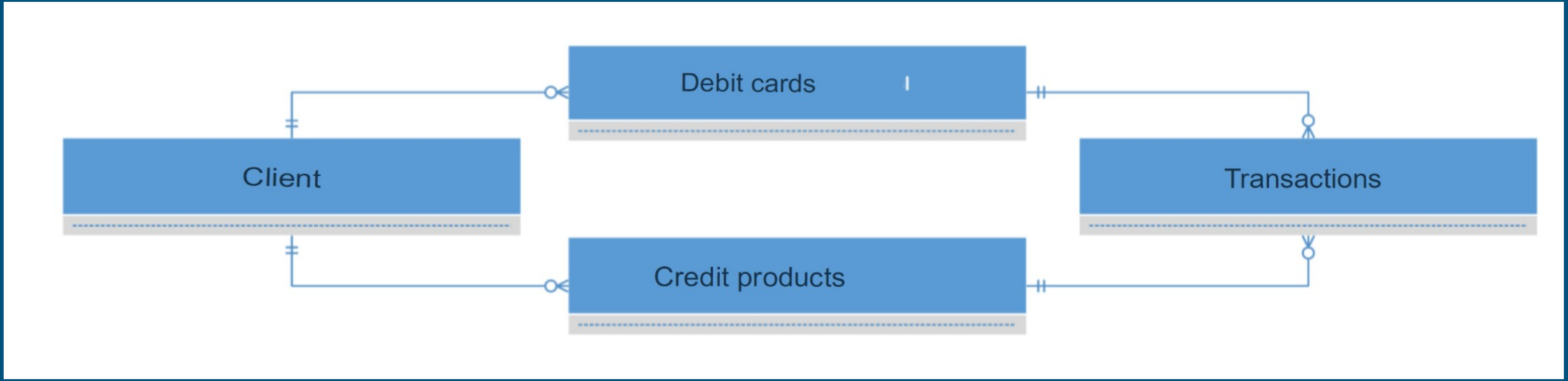
ER MODEL

ER-model (entity-relationship model) is a data model that allows you to describe conceptual and logical schemes.

Set:

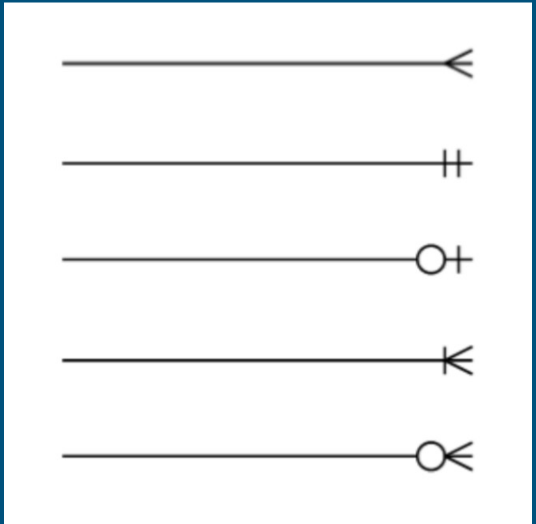
- Entities
- Connections

ENTITY NOTATION

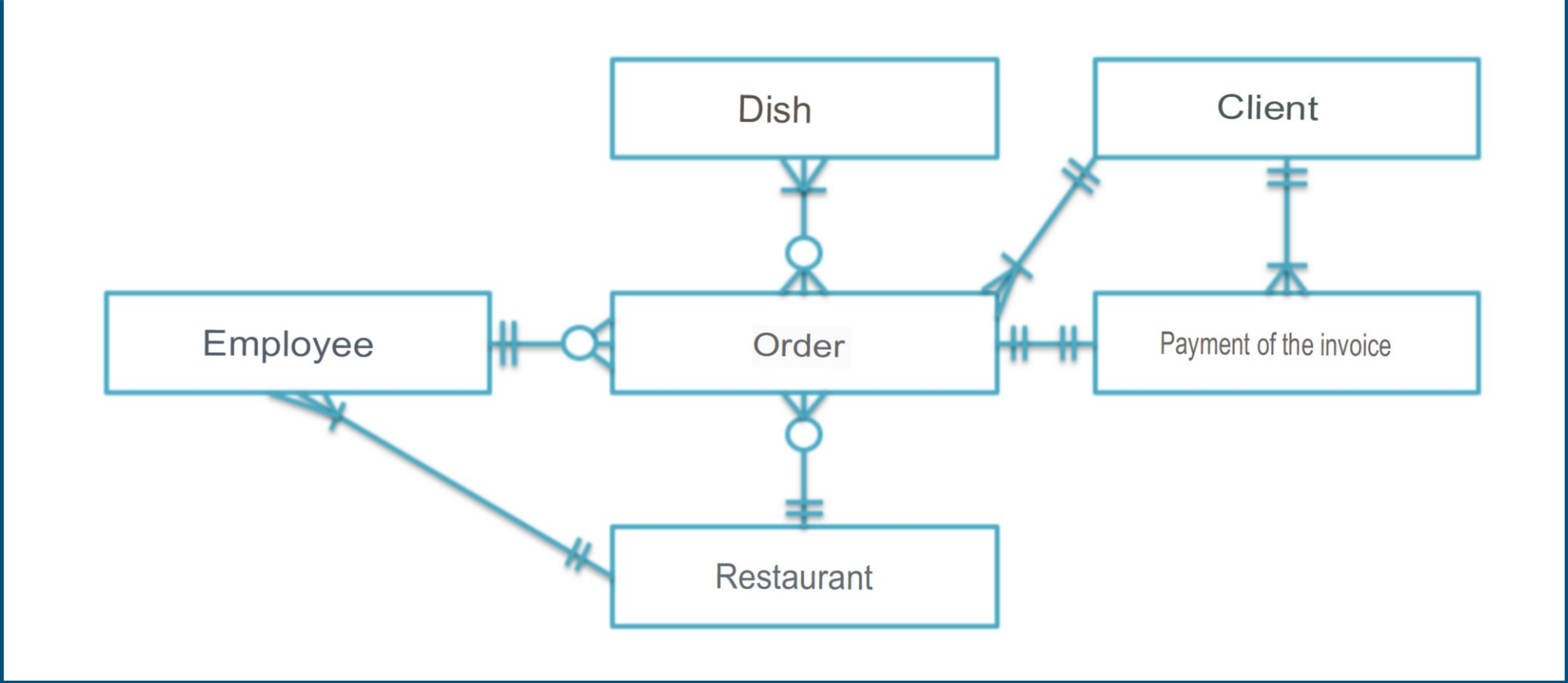


Crow’s Foot notation

- ✓ Many (strictly more than one)
- ✓ One and only one
- ✓ One or zero
- ✓ Many or one
- ✓ Many or one or zero



EXAMPLE


Dish
Client
Employee
Order
Payment of the invoice
Restaurant

STEP 3 CREATING A LOGICAL STRUCTURE

- We already have a conceptual scheme
- We want to get a higher level of detail:
- Clarification of the attribute composition
    - Clarification of the restrictions imposed on the attribute composition
    - Normalization of relations
    - In some cases, it is allowed to specify the attribute type

POTENTIAL KEY

– this is a subset of the attributes of the relationship that meets the requirements of uniqueness and minimality

- Uniqueness: there are not and cannot be two tuples of a given relation in which the values of this subset of attributes match
- Minimality: there is no smaller subset of attributes in the potential key that satisfies the uniqueness condition

POTENTIAL KEY

- From the relationship property: a potential key always exists, even if it includes all the attributes of the relationship
- It is acceptable to have multiple potential keys in relation

CLASSIFICATION BASED ON COMMUNITY

Potential key

- Simple: consists of exactly one attribute
- Composite: consists of two or more attributes

PRIMARY KEY

– this is one of the potential relationship keys selected as the Primary key (PK)

If there is only one potential key in the relation, it will be the primary key

If there are several potential keys, then:

- One of them is selected as the primary
- The other keys are called alternative keys

The absence of a value is unacceptable

PRIMARY KEY

- Theoretically, all potential keys are equally suitable for use as a primary key
- In practice, they use the potential key that:
    - Takes up less storage space
    - Will not lose its uniqueness over time

SURROGATE KEY

– this is an additional service field that is added to the already existing information fields of the table, the only purpose of which is to serve as a primary key

- The value of such a field is generated artificially
- Do not look for some deep meaning in it
- A key that is based on an already existing field is called natural
- A key that is based on a natural key by adding an additional field is called an intelligent key.

SURROGATE KEY

- Usually the surrogate key is a numeric field
- The values of the surrogate key are generated by an arithmetic progression with a step of 1
- In a number of DBMS there is a special data type that automatically generates such a sequence:
  - PostgreSQL – SERIAL
  - MySQL – AUTO_INCREMENT

ADVANTAGES OF SURROGATE KEYS

- Immutability: filled in with one value once and for all(except in extraordinary situations)
- Guaranteed uniqueness: because the value is generated by auto-increment, repetition of values is excluded
- Flexibility: since such a key does not carry any informative load, it can be freely replaced
- Easier to program: allows you not to tie on the structure of a specific database. Especially convenient for languages with statictyping
- Efficiency: more convenient when creating links to other tables

DISADVANTAGES OF SURROGATE KEYS

- Vulnerability of generators: by key numbers it is possible to find out the number of new records for a certain period of time
- Uninformativeness: manual database check becomes more complicated
- Induces the administrator to skip normalization: instead of breaking the relationship into several relationships and carefully taking into account all the connections, the temptation is great to simply create a surrogate key
- Optimization issues: the need to maintain both surrogate and natural keys (we will talk about this in Lecture 6)
- Unwitting binding of the developer to the behavior of the key generator in a specific DBMS

FOREIGN KEY

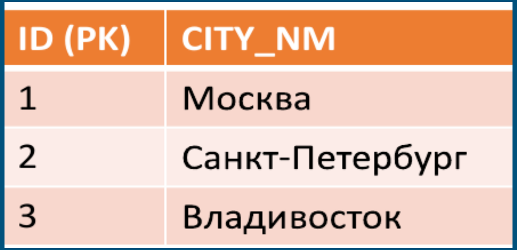
Let 1 and 2 be two variable relations, not necessarily different. The foreign key of FK in 2 is a subset of the attributes of variable 2 such that the following requirements are met:

There is a potential PK key in the relation variable 1 such that PK and FK match exactly up to the renaming of attributes

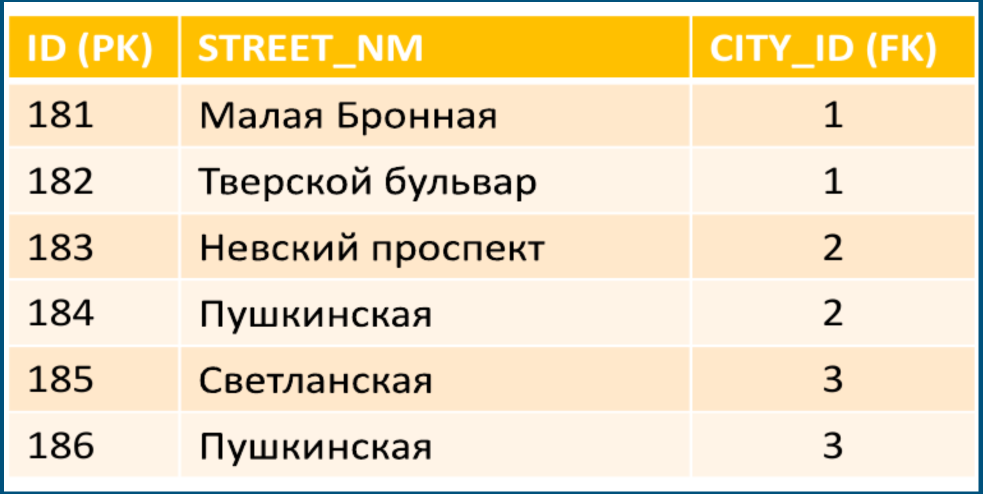
At any given time, each value of FK in the current value 2 is identical to the value of PK in some tuple in the current value 1 In other words, at any given time , the set of all values of FK in 2 is a subset of the values of PK in 1

FOREIGN KEY

- Relation 1, containing a potential key, is called the main, target, or parent
- Relation 2, containing a foreign key, is called subordinate or child



ID (PK)	CITY_NM
1	Москва
2	Санкт-Петербург
3	Владивосток



ID (PK)	STREET_NM	CITY_ID (FK)
181	Малая Бронная	1
182	Тверской бульвар	1
183	Невский проспект	2
184	Пушкинская	2
185	Светланская	3
186	Пушкинская	3

NORMAL FORM

- Normal form is a property of a relationship in a relational data model that characterizes it from the point of view of redundancy, potentially leading to logically erroneous results of sampling or data modification
- The normal form is defined as a set of requirements that the relation must satisfy
- Bringing the database to normal form – normalization
- Each following form includes the limitations of the previous ones

DATABASE NORMALIZATION

Intended:

- Minimizing logical redundancy
- Reducing potential inconsistency

Not intended for:

- Decrease/increase in DB performance
- Decrease / increase in the physical volume of the database

DATABASE NORMALIZATION

- Exclusion of certain types of redundancy
- Elimination of some anomalies* Updates
- Development of a database project that is:
  - High-quality representation of the real world
  - Intuitive
  - Easy to expand in the future
- Simplification of the procedure for applying the necessary integrity constraints

DATA ANOMALIES

The situation in the DB table is such that:

- The work with the database is significantly complicated
- There are contradictions in the database

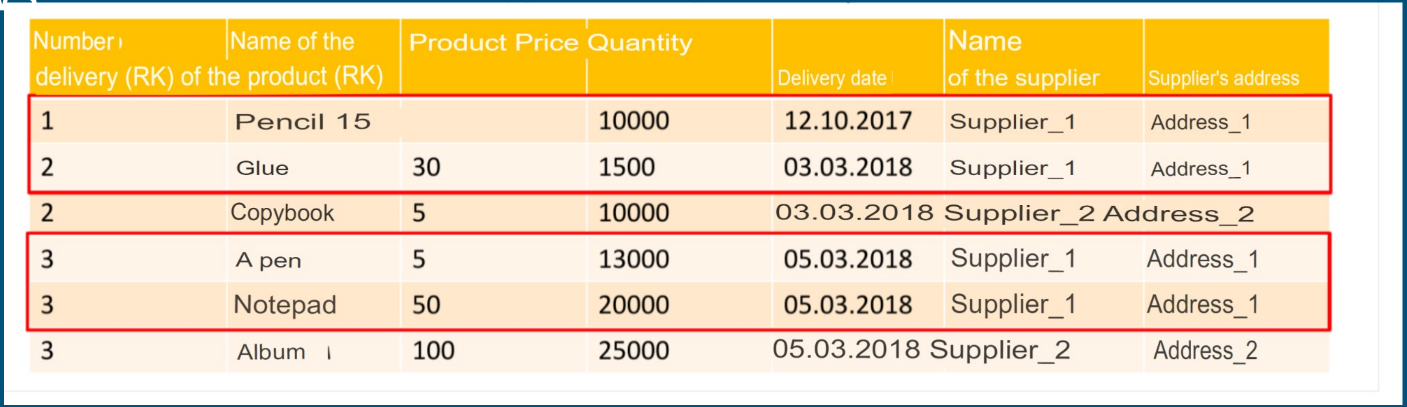
Reason:

- Excessive duplication of data in the table

MODIFICATION ANOMALIES

Changing the data of one record entails the need to change the similar data of some more records

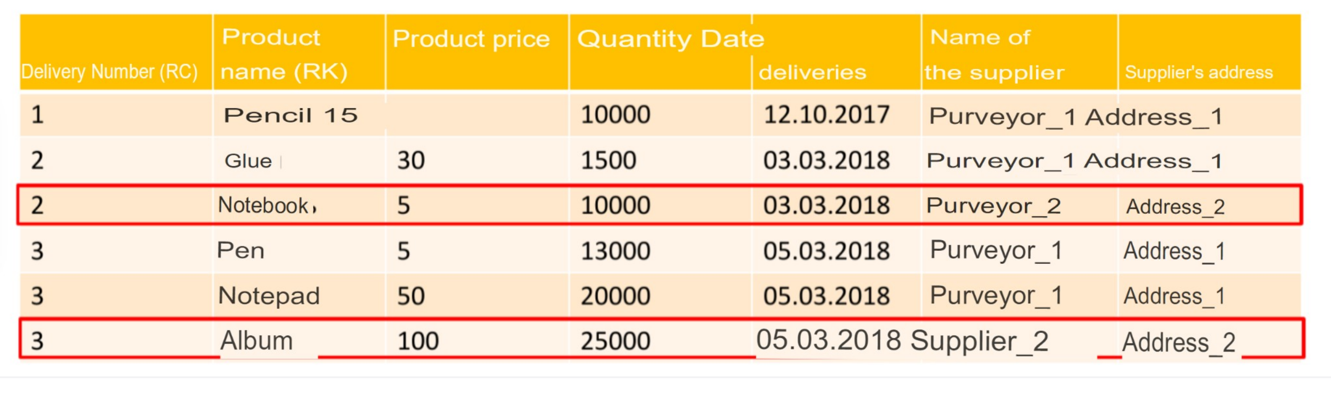
Example: to change the distributor 1’s address, we have to change all the rows



Number delivery (RK)	Name of the of the product (RK)	Product Price	Quantity	Delivery date	Name of the supplier	Supplier's address
1	Pencil 15		10000	12.10.2017	Supplier_1	Address_1
2	Glue	30	1500	03.03.2018	Supplier_1	Address_1
2	Copybook	5	10000	03.03.2018	Supplier_2	Address_2
3	A pen	5	13000	05.03.2018	Supplier_1	Address_1
3	Notepad	50	20000	05.03.2018	Supplier_1	Address_1
3	Album	100	25000	05.03.2018	Supplier_2	Address_2

ANOMALIES OF DELETION

Deleting certain records carries the loss of information that you did not want to de

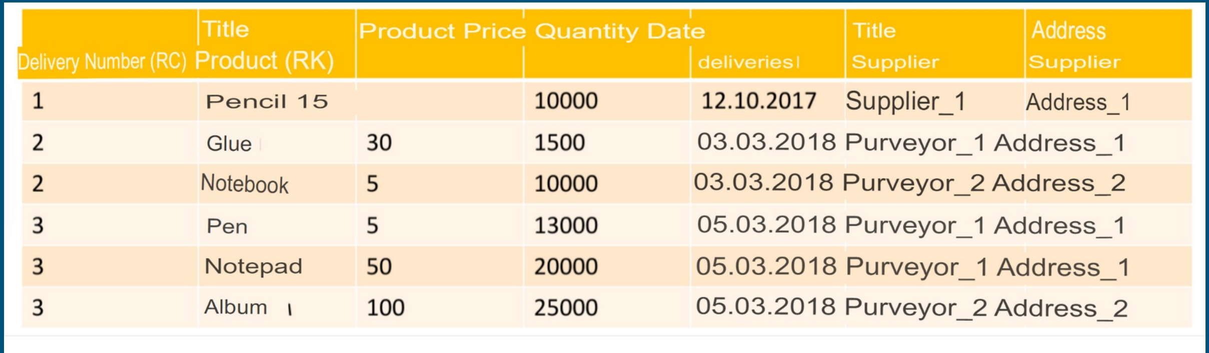


Delivery Number (RC)	Product name (RK)	Product price	Quantity	Date deliveries	Name of the supplier	Supplier's address
1	Pencil 15		10000	12.10.2017	Purveyor_1	Address_1
2	Glue	30	1500	03.03.2018	Purveyor_1	Address_1
2	Notebook	5	10000	03.03.2018	Purveyor_2	Address_2
3	Pen	5	13000	05.03.2018	Purveyor_1	Address_1
3	Notepad	50	20000	05.03.2018	Purveyor_1	Address_1
3	Album	100	25000	05.03.2018	Supplier_2	Address_2

We want to delete records of deliveries from supplier 2:
We lose all information about supplier 2, including its address

ANOMALIES OF ADDITION

We cannot add a new entry if the values of the primary keys are unknown



	Title	Product Price	Quantity Date		Title	Address
Delivery Number (RC)	Product (RK)			deliveries	Supplier	Supplier
1	Pencil 15		10000	12.10.2017	Supplier_1	Address_1
2	Glue	30	1500	03.03.2018	Purveyor_1	Address_1
2	Notebook	5	10000	03.03.2018	Purveyor_2	Address_2
3	Pen	5	13000	05.03.2018	Purveyor_1	Address_1
3	Notepad	50	20000	05.03.2018	Purveyor_1	Address_1
3	Album	100	25000	05.03.2018	Purveyor_2	Address_2

We have signed a contract with supplier 3:
We cannot add information about him to the table, because there have been no deliveries yet

DATABASE NORMALIZATION

- Database normalization is performed by decomposition of the relation
- Decomposition – decomposition of the original variable of the relation into several equivalent
- Decomposition is the reverse of a compound
- Decomposition is called lossless or correct if it is reversible

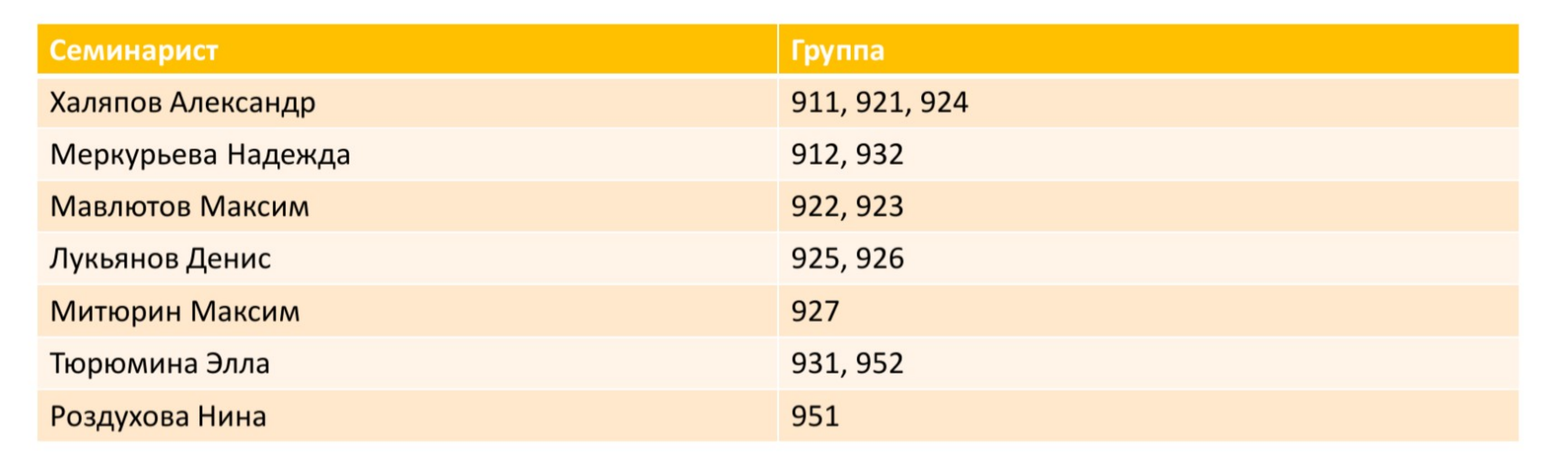
NORMAL FORMS

- **First Normal Form (1NF)**
- **Second Normal Form (2NF)**
- **Third Normal Form (3NF)**
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form / Projection-junction Normal Form (5NF/PJNF)
- Domain-key Normal Form (DKNF)
- Sixth normal Form (6NF)

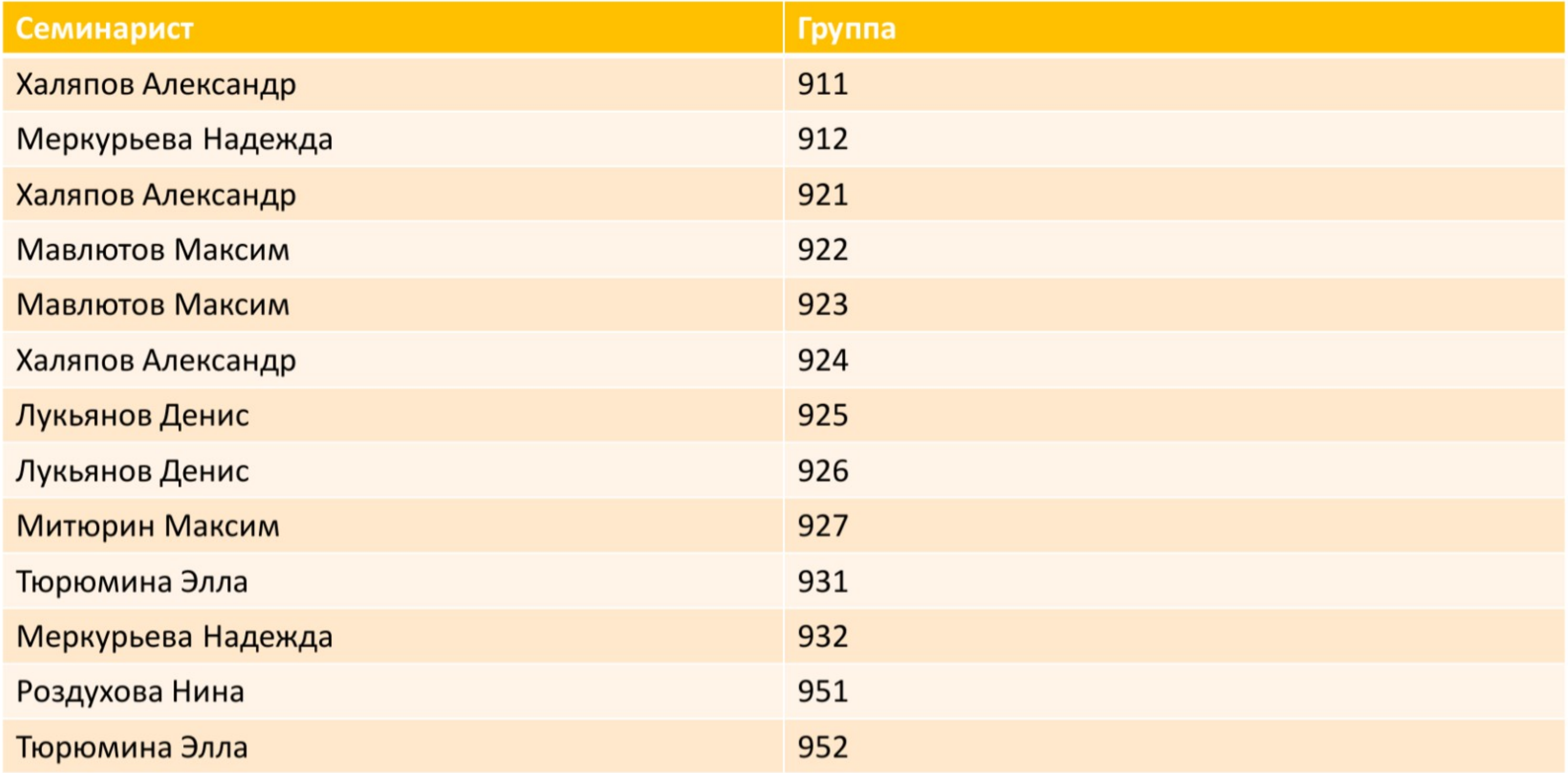
THE FIRST NORMAL FORM

- A relation variable is in the first normal form if and only if the values of all attributes of the relation are atomic.
- A relation is in 1NF if all its attributes are simple. All domains used contain only scalar values.

EXAMPLE



Семинарист	Группа
Халяпов Александр	911, 921, 924
Меркурьева Надежда	912, 932
Мавлютов Максим	922, 923
Лукьянов Денис	925, 926
Митюрин Максим	927
Тюрюмина Элла	931, 952
Роздухова Нина	951

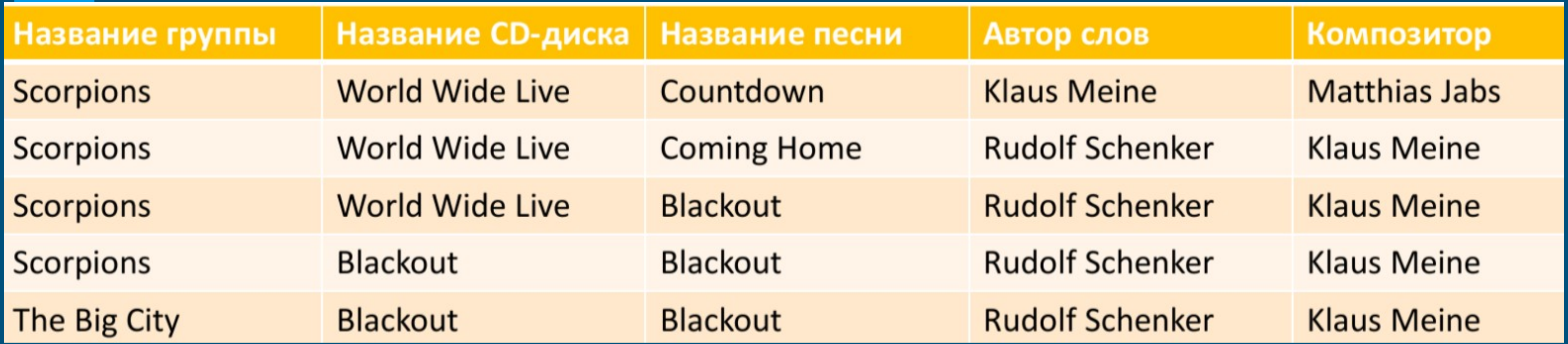


Семинарист	Группа
Халяпов Александр	911
Меркурьева Надежда	912
Халяпов Александр	921
Мавлютов Максим	922
Мавлютов Максим	923
Халяпов Александр	924
Лукьянов Денис	925
Лукьянов Денис	926
Митюрин Максим	927
Тюрюмина Элла	931
Меркурьева Надежда	932
Роздухова Нина	951
Тюрюмина Элла	952

THE SECOND NORMAL FORM

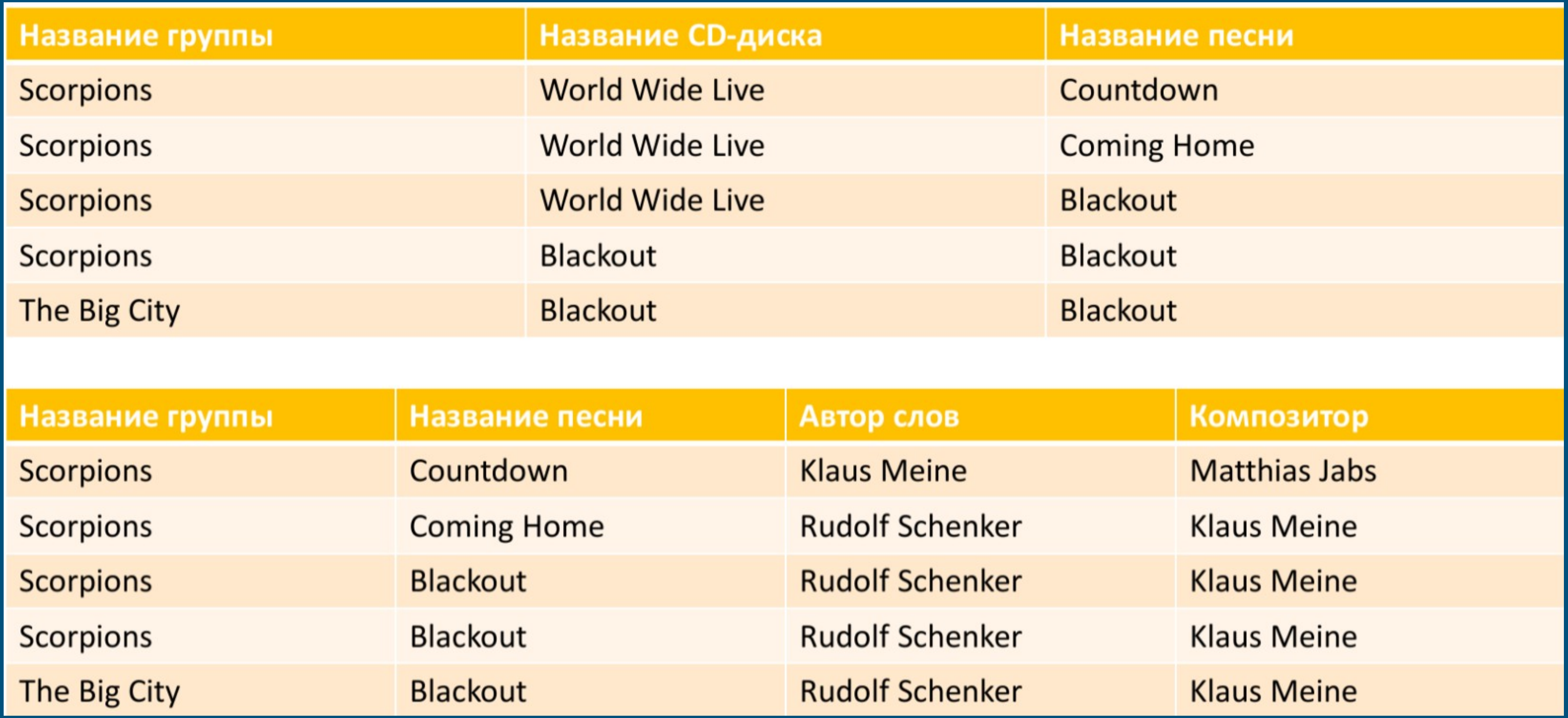
- A relation variable is in the second normal form if and only if it is in the first normal form, and each non-key attribute is minimally functionally dependent on a potential key
- The functional relationship between the sets of attributes X and Y means that for any valid set of tuples , the following is true in this respect: if two tuples coincide in the value of X, then they coincide in the value of Y
- Minimal functional dependency means that the primary key does not contain a smaller subset of attributes, relations which can also be derived from this functional dependency

Example



Название группы	Название CD-диска	Название песни	Автор слов	Композитор
Scorpions	World Wide Live	Countdown	Klaus Meine	Matthias Jabs
Scorpions	World Wide Live	Coming Home	Rudolf Schenker	Klaus Meine
Scorpions	World Wide Live	Blackout	Rudolf Schenker	Klaus Meine
Scorpions	Blackout	Blackout	Rudolf Schenker	Klaus Meine
The Big City	Blackout	Blackout	Rudolf Schenker	Klaus Meine

Author and Composer depend only on Group and Song, not on CD name



Название группы	Название CD-диска	Название песни
Scorpions	World Wide Live	Countdown
Scorpions	World Wide Live	Coming Home
Scorpions	World Wide Live	Blackout
Scorpions	Blackout	Blackout
The Big City	Blackout	Blackout

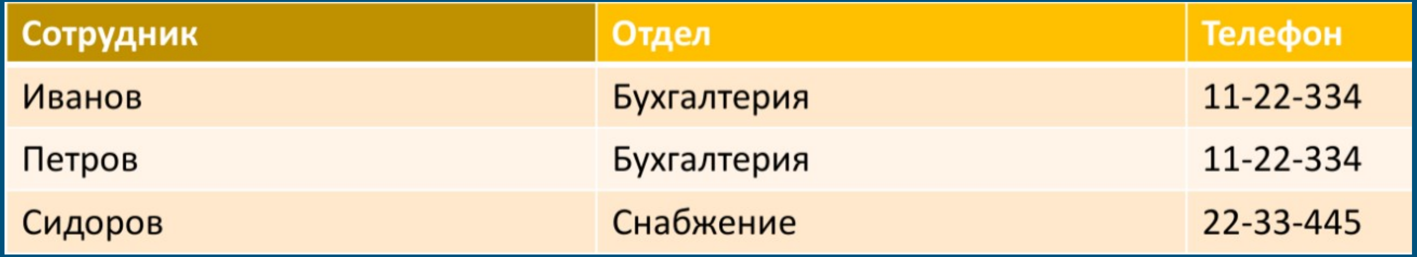
Название группы	Название песни	Автор слов	Композитор
Scorpions	Countdown	Klaus Meine	Matthias Jabs
Scorpions	Coming Home	Rudolf Schenker	Klaus Meine
Scorpions	Blackout	Rudolf Schenker	Klaus Meine
Scorpions	Blackout	Rudolf Schenker	Klaus Meine
The Big City	Blackout	Rudolf Schenker	Klaus Meine

THE THIRD NORMAL FORM

A relation variable is in the third normal form (3NF) if and only if it is in the second normal form, and each non-key attribute is non-transitively functionally dependent on the primary key

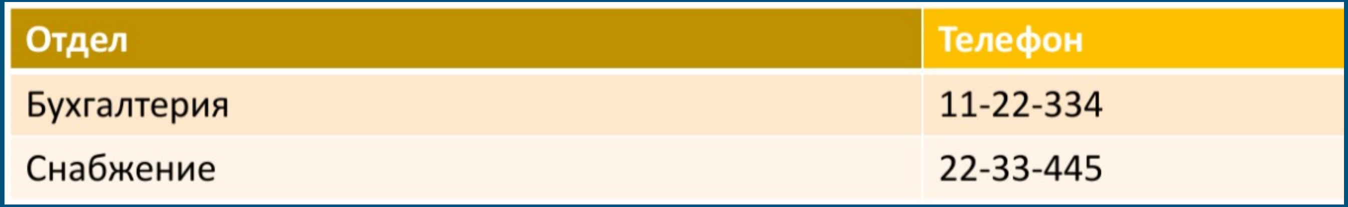
In other words, each non-key field must contain information about the key, the full key, and nothing but the key

Example

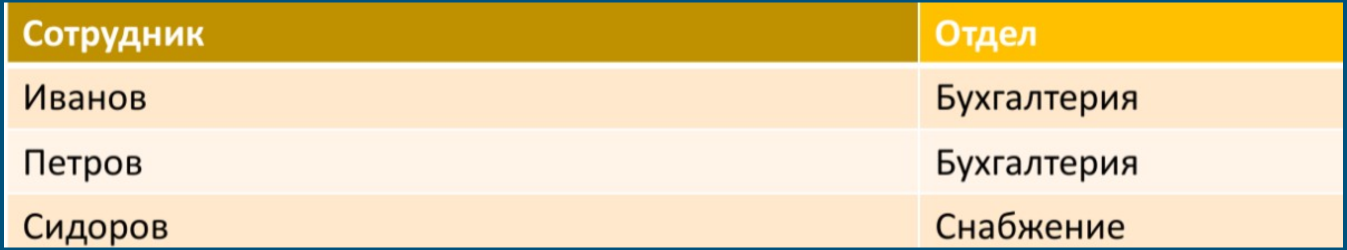


Сотрудник	Отдел	Телефон
Иванов	Бухгалтерия	11-22-334
Петров	Бухгалтерия	11-22-334
Сидоров	Снабжение	22-33-445

Example



Отдел	Телефон
Бухгалтерия	11-22-334
Снабжение	22-33-445



Сотрудник	Отдел
Иванов	Бухгалтерия
Петров	Бухгалтерия
Сидоров	Снабжение

WHY DO WE NEED THAT?

- Helps beginners create DBs that aren’t so bad
- Helps avoid typical mistakes
- Develops the habit of creating the right databases and not putting the design off for later
- Gradually develops design skills, without a strong connection to normal forms

STEP 3 CREATING A LOGICAL STRUCTURE

We already have a conceptual scheme

We want to get a higher level of detail:

Clarification of the attribute composition

Clarification of the restrictions imposed on the attribute composition

In some cases, it is allowed to specify the attribute type

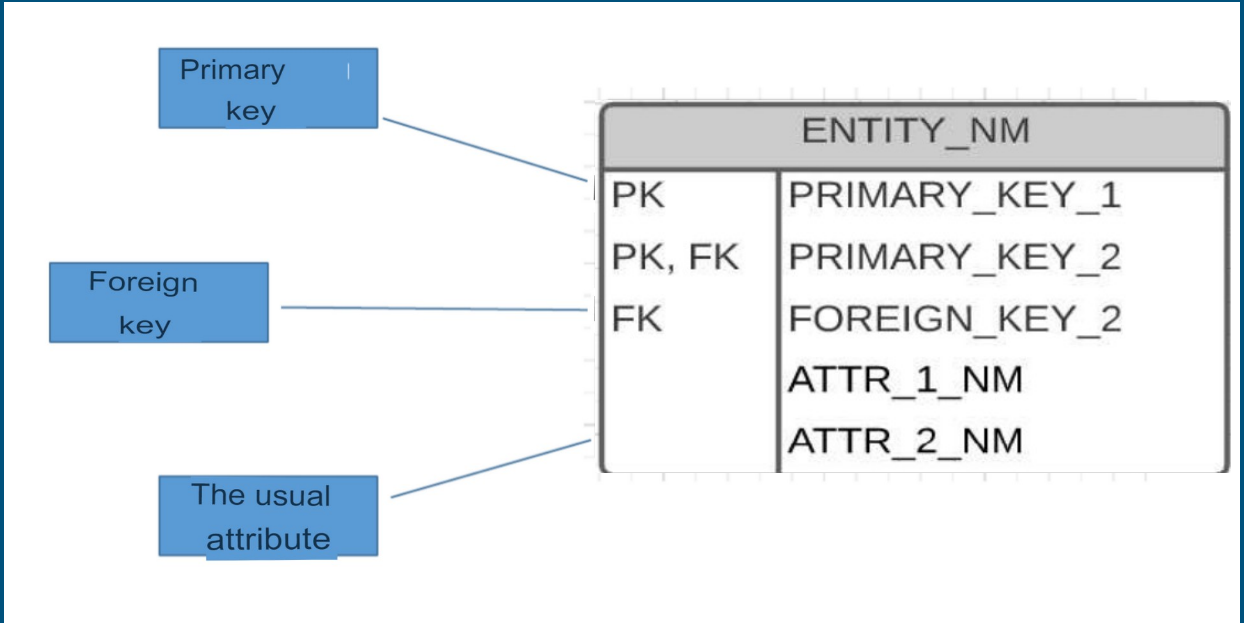
LOGICAL DATA MODEL

The logical data model is an extension of the conceptual data model by defining attributes, descriptions and constraints for entities, partially clarifies the composition of entities and the relationships between them.

LOGICAL DATA MODEL

- Allows going beyond the conceptual model
- Is a prototype of a future physical model
- Does not take into account the specifics of any particular DBMS
- ER notation is also used for illustration

ER notation “Entity”

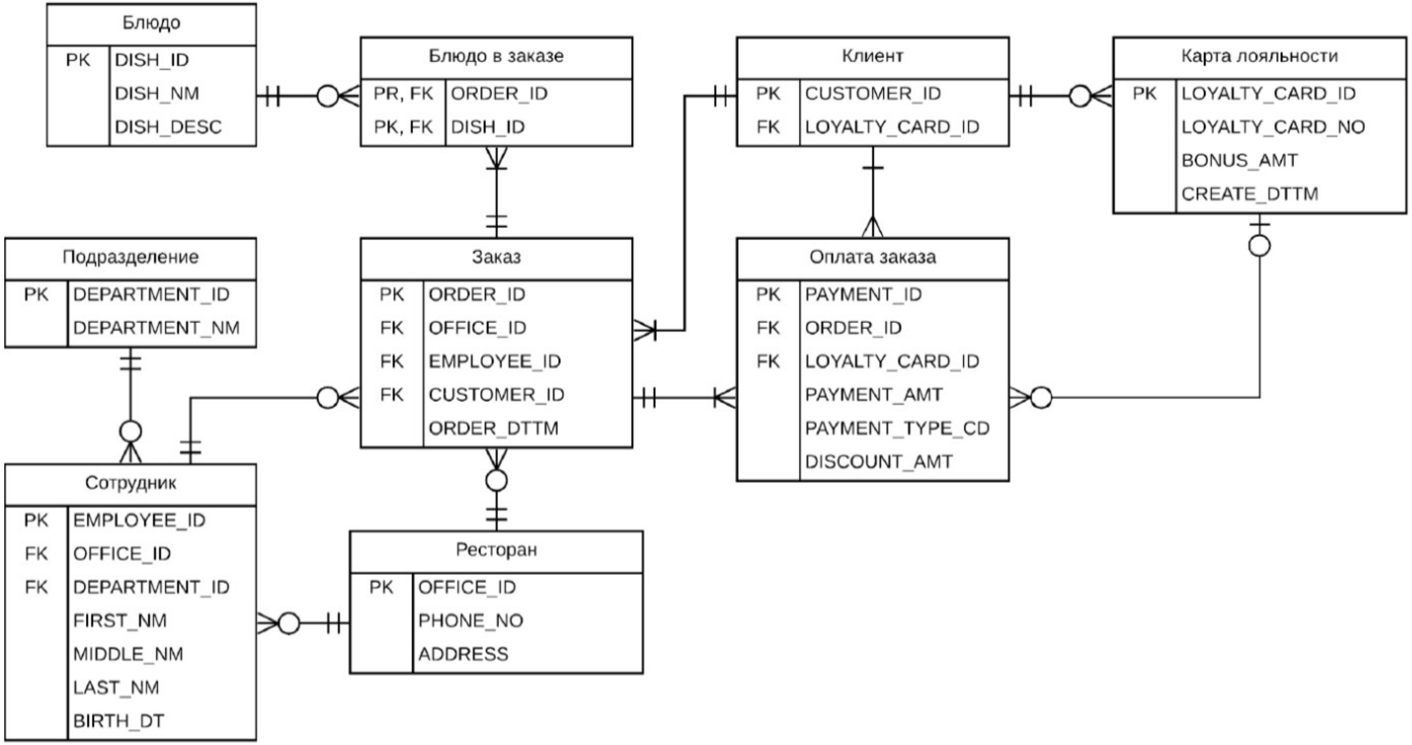


Primary key

Foreign key

The usual attribute

ENTITY_NM	
PK	PRIMARY_KEY_1
PK, FK	PRIMARY_KEY_2
FK	FOREIGN_KEY_2
	ATTR_1_NM
	ATTR_2_NM



Блюдо	
PK	DISH_ID
	DISH_NM
	DISH_DESC

Блюдо в заказе	
PR, FK	ORDER_ID
PK, FK	DISH_ID

Клиент	
PK	CUSTOMER_ID
FK	LOYALTY_CARD_ID

Карта лояльности	
PK	LOYALTY_CARD_ID
	LOYALTY_CARD_NO
	BONUS_AMT
	CREATE_DTTM

Подразделение	
PK	DEPARTMENT_ID
	DEPARTMENT_NM

Заказ	
PK	ORDER_ID
FK	OFFICE_ID
FK	EMPLOYEE_ID
FK	CUSTOMER_ID
	ORDER_DTTM

Оплата заказа	
PK	PAYMENT_ID
FK	ORDER_ID
FK	LOYALTY_CARD_ID
	PAYMENT_AMT
	PAYMENT_TYPE_CD
	DISCOUNT_AMT

Сотрудник	
PK	EMPLOYEE_ID
FK	OFFICE_ID
FK	DEPARTMENT_ID
	FIRST_NM
	MIDDLE_NM
	LAST_NM
	BIRTH_DT

Ресторан	
PK	OFFICE_ID
	PHONE_NO
	ADDRESS