Advanced Management of Data
Exercise 1 Topic 3:
Object-Relational Database Systems

Object-Relational Database (ORD) or Object-Relational Database Management System (ORDBMS)

- similar to relational database but with object-oriented database model
- since the early 1990s and partly part of the SQL standard via structured user-defined types or just structures types since SQL:1999
- try to provide the best of both worlds between the two extremes relational database and object-oriented database
- characteristic properties of ORDBMS are
  - complex data
  - type inheritance
  - object behaviour

Object-Relational Database Management System (ORDBMS) PostgreSQL

- PostgreSQL is a famous example for such an ORDBMS
- it is free and open-source and we already used it in the last exercises
- we've even used some of the features in previous exercises like arrays
- we will still use PostgreSQL in the next exercises and explore some more features
- unfortunately, not all parts of the standard are implemented, but we will see…

Row Types

- until now, we mostly used the relational model with the First Normal Form (1NF)
- at this point we are going to violate it, by dropping the rule, that all attributes have to contain only atomic values, so we are using Non First Normal Form (NF^2)
- Task: create a new table `persons` with the attributes `name`, `address`, `email`, `telephone` and `dateofbirth`
  - the `name` consists of `forename` and `surname`
  - the `address` consists of `street`, `city` and `state`
  - the `street` consists of `streetname` and `housenumber`
  - the `city` consists of `cityname` and `postalcode`
  - use `date`-type for `dateofbirth` and some string-type for all other attributes or sub-attributes (there might be basic conventions on several attributes, but they differ from country to country and even house-numbers can contain letters)

Structured Types

```
CREATE TYPE nametype AS (forename VARCHAR, surname VARCHAR);

CREATE TYPE streettype AS (streetname VARCHAR, housenumber VARCHAR);

CREATE TYPE citytype AS (cityname VARCHAR, postalcode VARCHAR);

CREATE TYPE addresstype AS (street streettype, city citytype, state VARCHAR);

CREATE TABLE persons (name nametype, address addresstype, email VARCHAR, telephone
VARCHAR, dateofbirth DATE);
```

- in PostgreSQL you can't declare tables with anonymous composite types as presented in the lecture, so you have to create structured user-defined types at first
- Task: create an own type for `email` and `telephone`, too and change the table definition to use the new types

Distinct Types

```
CREATE DOMAIN emailtype AS VARCHAR;

CREATE DOMAIN telephonetype AS VARCHAR;

ALTER TABLE persons
  ALTER email TYPE emailtype,
  ALTER telephone TYPE telephonetype;
```

- the syntax for creating distinct types is a little bit different than creating structured types, but basically `DOMAIN`s are types, too (with the option to define constraints for this special alias type)
- Task: let's use this possibility and change the definition of `emailtype` to check for one "@"-sign, which is not at the beginning or the end

Distinct Types
Constraints

```
ALTER DOMAIN emailtype
  ADD CHECK (value ~ '^[^@]+@[^@]+$');
```

- a basic regular expression, that states
  - `^` at the beginning
  - `[^@]+` there must be at least one character, that is not an “@”
  - `@` followed by an “@”
  - `[^@]+` and again at least one character, that is not an “@”
  - `$` until the end
- Task: insert the data of the following people into the table `persons`
  - Max Mustermann, Musterstraße 42, 12345 Musterstadt, Germany, +49 1234 56789, Max@Mustermann.de, 1965-04-03
  - John Doe, 21 Main Street, Anytown 12345-6789, United States of America, +1-800-555-1234, John@Doe.com, 1976-05-04

Row Types
Inserted

```
INSERT INTO persons VALUES
  (ROW('Max', 'Mustermann'),
   ROW(ROW('Musterstraße', '42'), ROW('Musterstadt', '12345'), 'Germany'),
   'Max@Mustermann.de',
   '+49 1234 56789',
   '1965-04-03'),
  (('John', 'Doe'),
   (('Main Street', '21'), ('Anytown', '12345-6789'), 'United States of America'),
   'John@Doe.com',
   '+1-800-555-1234',
   '1976-05-04');
```

- the key word ROW is optional when there is more than one expression in the list, as shown with the second dataset
- Task: Max Mustermann has moved, so his address has to be changed to Mustergasse 42a, 01234 Musterdorf

Row Types Updated

```
UPDATE persons
  SET address = (('Mustergasse', '42a'), ('Musterdorf', '01234'),
                 'Germany')
  WHERE name = ROW('Max', 'Mustermann')::nametype;
```

- you have to give the DBMS a type-hint at the `WHERE`-clause, because it doesn't know how to compare an anonymous record-type with your `nametype`
- Task: Max and John have wives, which share their husbands address and telephone-number, but have their own email-addresses, that follow the same pattern as their husbands, so add them to the table, too, but try not to retype the addresses and telephone-numbers in your SQL-query
  - Erika Mustermann, 1966-02-01
  - Jane Doe, 1977-08-09

Row Types
Copied

```
INSERT INTO persons VALUES
  (ROW('Erika', 'Mustermann'),
   (SELECT address FROM persons WHERE (name).forename = 'Max'),
   'Erika@Mustermann.de',
   (SELECT telephone FROM persons WHERE (name).forename = 'Max'),
   '1966-02-01'),
  (('Jane', 'Doe'),
   (SELECT address FROM persons WHERE (name).surname = 'Doe'),
   'Jane@Doe.com',
   (SELECT telephone FROM persons WHERE (name).surname = 'Doe'),
   '1977-08-09');
```