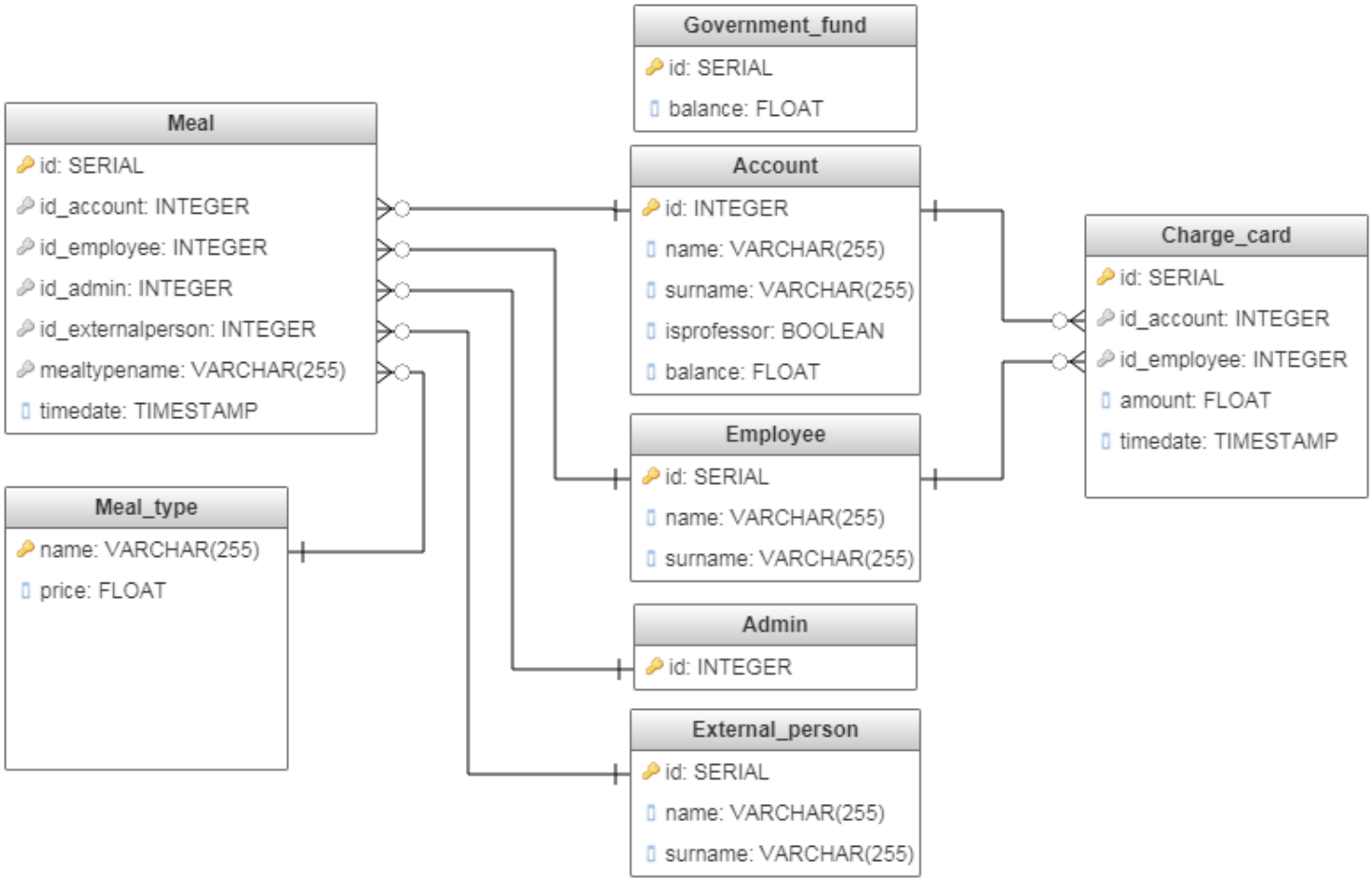
MILESTONE 2

Relational schema:



Relations:
Meal N : 1 Account
Meal N : 1 Employee
Meal N : 1 Admin
Meal N : 1 External_person
Meal N : 1 Meal_type
Account N : M Employee (relation: Charge_card)

There can be denoted an interesting fact inside the table Account: there is a boolean attribute called isprofessor.
Isprofessor is used in order to avoid to implement two different tables for professors and students (which are both ISA of Account), because it wuold be superflous and the problem can be archieved in a simplier way.

P.S. The script files are not commented, because it would be annoying to read from a text file; there are the same commands of the scripts, but with the relative comments.

Create file content (create.sql):

```
/* SAVEV DAVID, ID 14057, dsavev@unibz.it */

CREATE TABLE Government_fund
(id serial PRIMARY KEY,
balance float DEFAULT 0 CHECK (balance >= 0.0));
/*Serial data type on id is automatically created and incremented each time a new tuple is inserted – it is a fast solution in case you don't need a "real" identificator number.
On balance, a default empty balance is created (untill the government does not provide money), and when updating it is not allowed to set it negative (CHECK constraint). */

CREATE TABLE Account
(id int PRIMARY KEY CHECK (id >= 100),
surname varchar(255) NOT NULL,
name varchar(255) NOT NULL,
isprofessor boolean NOT NULL,
balance float DEFAULT 0 CHECK (balance >= 0.0));
/*id in this case contains the CHECK constraint that allows only id's >= 100, since in this university the unique id's of students and professors are always greater or equal 100.
Isprofessor is a boolean attribute, and its scope is to differentiate a student from a professor (TRUE if it is a professor, FALSE if it is a student).
Obviously, each student / professor must be inserted with a name and a surname (NOT NULL constraint).
Also in this case, balance is by default empty (untill the student / professor does not charge the card), and when updating it is not allowed to set it negative (CHECK constraint) - This means that the student / professor cannot buy a menu if in his balance has not sufficient money. */

CREATE TABLE Employee
(id serial PRIMARY KEY,
name varchar(255) NOT NULL,
surname varchar(255) NOT NULL);
/* Also in this case a serial data type was used on the id which is automatically created and incremented each time a new employee is assumed.
Obviously, we need a name and a surname of each of the employees in order to know who exactly is (NOT NULL constraint). */

CREATE TABLE Charge_card
(id serial PRIMARY KEY,
id_account int,
id_employee int,
FOREIGN KEY(id_account) REFERENCES Account(id) on delete set null on update cascade,
FOREIGN KEY(id_employee) REFERENCES Employee(id) on delete set null on update cascade,
timedate timestamp DEFAULT CURRENT_TIMESTAMP,
amount float NOT NULL CHECK (amount > 0.0));
/* Charge_card table is used in order to keep track of the charged cards and the relative amount of cash inserted.
This table is between a many to many relationship (Account, Employee) but doesn't contain a combined key.
I inserted a serial id as primary key, because each student can charge his card as many times he wants with the same employee (with a combined key this wuoldn't work).
When creating the two foreign keys, it was used the referential integrity constraint on update cascade in order to change id_account or id_employee if it is changed in the referenced table (example: a student has lost his card and receives a new one with a new id → it wuold be nice if his cronology wuold be connected with the new id).
The constraint on delete set null, because if an account or employee is deleted, its transactions should not (because if by deleting an account we would miss his transactions, we cannot calculate the incomes of the day correctly!).
Timedate is a variable used in order to save the actual date and time when a new tuple is inserted (by using the function CURRENT_TIMESTAMP).
Amount cannot obviously be null and must be > 0 (can you immagine to charge your card with a negative sum ?) */

CREATE TABLE Admin
(id int PRIMARY KEY CHECK(id < 100));
/* In this case, an admin will use an id < 100 (because >= 100 are used by students and professors). */

CREATE TABLE Meal_type
(name varchar(255) PRIMARY KEY,
price float DEFAULT 0 CHECK (price >= 0.0));
/* The name of the meal type contains the following strings („Full menu", „Light menu", „Extra light menu").
The price of a meal type cannot obviously be negative ! */

CREATE TABLE External_person
(id serial PRIMARY KEY,
surname varchar(255) NOT NULL,
name varchar(255) NOT NULL);
/* Also in this case a serial data type was used on the id which is automatically created and incremented each time a new external person is registered.
Obviously, we need a name and a surname of each of the external person for security issues (NOT NULL constraint).*/
```

```
CREATE TABLE Meal
(id serial PRIMARY KEY,
id_employee int,
id_admin int,
id_account int,
id_externalperson int,
mealtypename varchar(255) NOT NULL,
FOREIGN KEY (id_employee) REFERENCES Employee(id) on delete set null on update cascade,
FOREIGN KEY (id_admin) REFERENCES Admin(id) on delete set null on update cascade,
FOREIGN KEY (id_account) REFERENCES Account(id) on delete set null on update cascade,
FOREIGN KEY (id_externalperson) REFERENCES External_person(id) on delete set null on update cascade,
FOREIGN KEY (mealtypename) REFERENCES Meal_type(name) on delete no action on update cascade,
timedate timestamp DEFAULT CURRENT_TIMESTAMP);
/* Meal is the most complex table in the DB.
Meal is always in relations M : 1, and then will contain as foreign keys the references to the other tables with which is in relation.
The id is obviously a serial primary key and is auto incremented each time a new meal is sold.
Id_admin, id_account, id_externalperson can be null, because only one of them has bought the specific menu.
The used referential integrity constraints are: on delete set null (because, as explained before if by deleting an account we would miss his transactions, we cannot calculate the total amount of menus sold).
On update cascade was used for all the keys, in order to change automatically in this table the values of the attributes changed in the referenced one.
Mealtypename uses on delete no action, because by using on delete set null we would loose an important information to derive the price.
Also here, a timedate attribute is very important, in order to say at which time in which date a menu was sold.*/
```

Load file content (load.sql):

```
/* SAVEV DAVID, ID 14057, dsavev@unibz.it */

INSERT INTO employee(name, surname) values ('John', 'Brown');
INSERT INTO employee(name, surname) values ('Ivan', 'McConnor');
INSERT INTO employee(name, surname) values ('Marco', 'Rossi');
INSERT INTO account values (1010, 'Savev', 'David', false, 35);
INSERT INTO account values (5093, 'Piero', 'Pierucci', true, 12);
INSERT INTO account values (100240, 'Leonardo', 'Antonucci', true, 29);
INSERT INTO government_fund(balance) values (105000);
INSERT INTO meal_type values ('Full menu', 8.64);
INSERT INTO meal_type values ('Light menu', 6.90);
INSERT INTO meal_type values ('Extra light menu', 5.20);
INSERT INTO external_person(name, surname) values('Pietro', 'Pietrini');
INSERT INTO external_person(name, surname) values('Leo', 'Guappi');
INSERT INTO admin values(1);
INSERT INTO meal(id_employee, id_account, mealtypename) values(2, 5093, 'Extra light menu');
/* Inserts inside meal that account with id nr. 5093 has bought an Extra light menu from employee with id 2.*/
UPDATE account set balance = balance - (select price from meal_type where name = 'Extra light menu') WHERE id = 5093;
/* Updates the balance of account id nr. 5093 by substracting the price of an Extra light menu.*/
INSERT INTO meal(id_employee, id_externalperson, mealtypename) values(1, (Select id from external_person WHERE surname = 'Guappi' AND name = 'Leo'), 'Light menu');
/* Put inside meal that external person with the following name has bought a Light menu from employee with id 1.*/
INSERT INTO meal(id_employee, id_externalperson, mealtypename) values(2, (Select id from external_person WHERE surname = 'Pietrini' AND name = 'Pietro'), 'Light menu');
INSERT INTO charge_card(id_account, id_employee, amount) values(1010, 3, 40);
/* Add's to charge_card that employee with id 3 has charged on the card of account with id 1010, 40 euros.*/
UPDATE account set balance = balance + 40 WHERE id = 1010;
/* Add's to the balance of account id nr. 1010, 40 euros.*/
```

Clean file content (clean.sql):

```
/* SAVEV DAVID, ID 14057, dsavev@unibz.it */

DROP TABLE charge_card;
DROP TABLE meal;
DROP TABLE meal_type;
DROP TABLE account;
DROP TABLE admin;
DROP TABLE employee;
DROP TABLE external_person;
DROP TABLE government_fund;
```