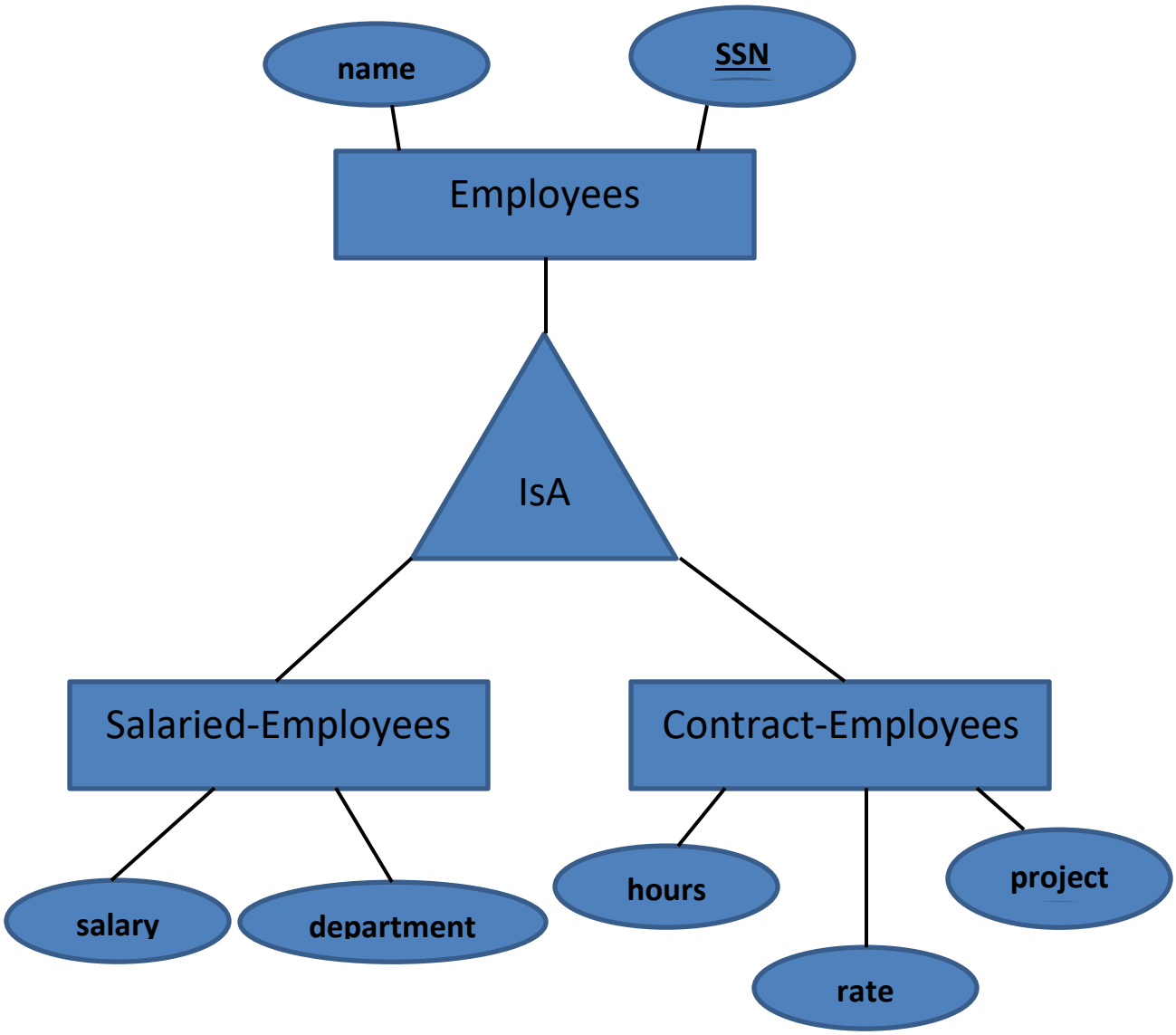
Homework #2

Ivan Martinovic

Problem 1: [30pts]

Draw the ER diagram using an ISA relationship for the following example:

Assume in a company, we have two kinds of staff, namely salaried-employees and contract-employees. Both have a unique identifier SSN and a name. For salaried-employees we keep track of their yearly salary as well as the department to which they have been assigned to. For contract-employees, we keep track of the number of hours they have a contract for, the hourly rate of pay, as well as the project number for the project in charge of this contract.



Write the DDL for **three** different possible schemas that map your ER Diagram into a relational model using one, two or three tables. For each of these three schemas, compare the redundancy, flexibility, and relative cost of querying. Note which schema you think is the best overall. Do not forget to use keys and foreign keys, as appropriate.

1) **One Relation for All** (one table)

```
CREATE TABLE Employees
        (SSN CHAR (7),
        name VARCHAR (32),
        salary REAL,
        department VARCHAR (20),
        hours INTEGER,
        rate REAL,
        project INTEGER,
        PRIMARY KEY (SSN));
```

In this approach each employee gets a record, which means that this approach would work for all types of IsA relationship (complete, partial, disjoint, overlapping). However the main drawback is that we might get lots of Null values which would likely be the case since an employee is rarely both a contract and a salaried employee, which leads to an unnecessary waste of space. Querying could get really slow since the single table has relatively many attributes, and pulling it from to main memory may be slow.

2) **Relations only for Subclasses** (two tables)

```
CREATE TABLE Salaried_employees
        (SSN CHAR (7),
        name VARCHAR (32),
        salary REAL,
        department VARCHAR(20),
        PRIMARY KEY (SSN));

CREATE TABLE Contracted_employees
        (SSN CHAR (7),
        name VARCHAR(32),
        hours INTEGER,
        rate REAL,
        project INTEGER,
        PRIMARY KEY (SSN));
```

In this case only the subclasses get their relation (table). This can be extremely useful if the IsA relationship is total and disjoint. This means that it is not very flexible i.e. if there would exist an employee who is both a contracted employee and a salaried employee (I guess it can happen in a

few rare cases) then we would keep a lot of redundant data for the same person lying around. On the other hand, if an employee is neither of the two, we would miss him completely. Since each of the 2 tables contain less attributes that the big table from the first approach, querying would be slightly faster.

3) **Relation for each Entity Set** (three tables)

```
CREATE TABLE Employees
        (SSN CHAR (7),
        name VARCHAR(32),
        PRIMARY KEY (SSN));

CREATE TABLE Salaried_employees
        (SSN CHAR (7),
        salary REAL,
        department VARCHAR(20),
        PRIMARY KEY (SSN),
        FOREIGN KEY (SSN) REFERENCES Employees (SSN));


CREATE TABLE Contracted_employees
        (SSN CHAR(7),
        hours INTEGER,
        rate REAL,
        project INTEGER,
        PRIMARY KEY (SSN),
        FOREIGN KEY (SSN) REFERENCES Employees (SSN));
```

In this case all entity sets get their own relation (table). This approach allows for flexibility since it would work for all kinds of IsA relationships (total, partial, overlapping and disjoint). The only "redundancy" is that all tables contain an SSN, however without this Primary (and also Foreign Key) we wouldn't be able to link the tables together. Compared to the previous two approaches the tables have the smallest number of attributes and would therefore have the greatest querying speeds.

In the long run I believe that the third approach is best in the long run. Its querying speeds are greatest. It has the least amount of redundancy, while also efficiently using memory space. It also provides the largest flexibility possible.

Next, consider the "none-overlap constraint among sibling classes" constraint that further restricts the type of ISA relationship being modeled. Discuss if these semantics could be enforced on any of your three designs using primary keys, foreign keys, unique and NOT NULL constraints.

Let's consider what the non-overlap condition implies for the 3 approaches

1) **Approach with 1 table** : non-overlap would imply that values of attributes (salary, department) must be not-null while (hours, rate, project) are null or vice-versa. We can see that primary keys, foreign keys, unique constraints would be of no use here. The NOT NULL constraint seems tempting however I do not yet have sufficient SQL syntax knowledge to say for certain whether the NOT NULL constraint can be used in conjunction to logical statements in order to achieve the desired exclusion (my prediction is likely no).

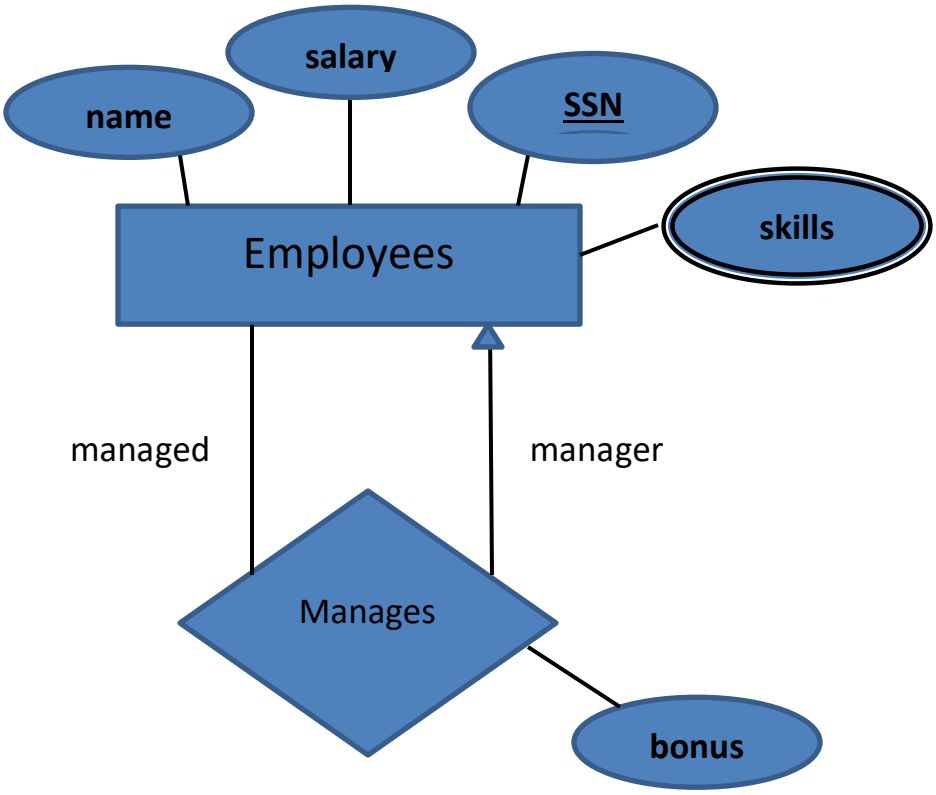
2)**Approach with 2 tables:** non-overlap would imply that no 2 entries in the Salaried-employees and Contracted-employees have the same SSN. Here it seems that unique would be the closest thing to what we want, however we would want to check whether the SSN attribute is unique across different tables, rather than within the same table. Because of my lack of SQL syntax knowledge I am unable to say whether the unique constraint would be of any help here (likely not).

3)**Approach with 3 tables:** non-overlap here would imply that if an SSN occurs in the Salaried-employees table, it is guaranteed to not appear in the Contracted-employees table. Again the case is similar to approach 2 and it seems that unique would be the closest thing to what we want, however we would want to check whether the SSN attribute is unique across different tables, rather than within the same table. Because of my lack of SQL syntax knowledge I am unable to say whether the unique constraint would be of any help here (likely not).

Problem 2: [40pts]

Assume an employee entity with attributes including Name, SSN, salary, and job-skill. Let SSN be the primary key of employee entities, while the Name attribute is not necessarily unique. Assume employee entities tend to have multiple different job skills, such as programming, networking, communication, etc. In other words, "skills" is a multi-valued attribute. In addition, an employee in your database may be the manager-of another employee in your database. Let us assume here that for each employee there would be at most one direct manager within the database, while an employee in our database may be in charge of (manager-of) up to 100 different employees. For each employee that a manager is managing successfully, she will be assigned a bonus increase in addition to her regular salary.

1. Model the above requirements using an ER diagram.



2. Translate your ER model into a relational schema using SQL DDL statements. Using the guidelines we discussed in class, include reducing the number of relations whenever appropriate.

```
CREATE TABLE Employees
        (SSN CHAR (7),
        managerSSN CHAR (7),
        name VARCHAR (64),
        salary REAL,
        bonus REAL,
        PRIMARY KEY (SSN),
        FOREIGN KEY (managerSSN) REFERENCES Employees(SSN));

CREATE TABLE Skills
        (SSN CHAR (7),
        skill VARCHAR (64),
        PRIMARY KEY (SSN, skill),
        FOREIGN KEY (SSN) REFERENCES Employees (SSN));
```

3. Enter your DDL statements into ORACLE to verify that they are correct. Develop some data set for your database application, and load your data into your schema. Show us the log (using spool off/on) of the above script of you working with ORACLE DBMS, i.e., of creating your schema and loading data into it.

```
SQL> spool
currently spooling to hw2.txt
SQL> CREATE TABLE Employees
  2        (SSN CHAR (7),
  3        managerSSN CHAR (7),
  4        name VARCHAR (64),
  5        salary REAL,
  6        bonus REAL,
  7        PRIMARY KEY (SSN),
  8        FOREIGN KEY (managerSSN) REFERENCES Employees(SSN));

Table created.

SQL> CREATE TABLE Skills
  2        (SSN CHAR (7),
  3        skill VARCHAR (64),
  4  PRIMARY KEY (SSN, skill),
  5        FOREIGN KEY (SSN) REFERENCES Employees (SSN));

Table created.

SQL> INSERT INTO Employees (SSN, name, salary, bonus) VALUES ('0000001', 'Jerry', 100, 0);

1 row created.

SQL> INSERT INTO Employees (SSN, managerSSN, name, salary, bonus) VALUES ('0000002',
'0000001', 'Jerry', 100, 0);

1 row created.
```

```
SQL> INSERT INTO Employees (SSN, managerSSN, name, salary, bonus) VALUES ('0000003', '0000001', 'Tom', 100, 50);

1 row created.

SQL> INSERT INTO Employees (SSN, managerSSN, name, salary, bonus) VALUES ('0000004', '0000002', 'John', 100, 150);

1 row created.

SQL> INSERT INTO Employees (SSN, managerSSN, name, salary, bonus) VALUES ('0000005', '0000002', 'Chris', 200, 150);

1 row created.

SQL> INSERT INTO Skills (SSN, skill) VALUES ('0000001', 'CEO');

1 row created.

SQL> INSERT INTO Skills (SSN, skill) VALUES ('0000001', 'General Manager');

1 row created.

SQL> INSERT INTO Skills (SSN, skill) VALUES ('0000002', 'Division Manager');

1 row created.

SQL> INSERT INTO Skills (SSN, skill) VALUES ('0000003', 'Division Manager');

1 row created.

SQL> INSERT INTO Skills (SSN, skill) VALUES ('0000004', 'Programmer');

1 row created.

SQL> INSERT INTO Skills (SSN, skill) VALUES ('0000005', 'Designer');

1 row created.

SQL> spool off
```

State additional assumptions that you have made, if any.

An employee's name is no longer than 64 characters. Each skill an employee possesses can be described in 64 characters. The Social Security Number is 7 characters long. The salary and bonus can be decimal, hence they are REALs. The bonus field is the bonus that the manager with Social Security Number managerSSN earns for managing Employee with Social Security Number SSN.

4. Provide a brief discussion justifying your chosen mapping strategy for the different cases of mapping that you encounter.

3 Mapping Strategies were used:
1) Entity sets become Relations – the Employees Entity Set became the Relation Entities with all corresponding attributes (except skills since it is a multi-valued attribute)

2) Multi-valued Attributes become Entity Sets – the skills attribute became a relation. It contains the SSN of the employee they describe, and the skill description as primary keys. SSN is also a foreign key referencing the SSN attribute in the Employees Entity Set, since the skills attribute belongs to the Employees Entity Set.

3) Many-to-One relationships without Keys – The Manages relationship is reflexive, however more importantly it is also a many-to-one relationship. Hence as any many-to-one relationship, the rule is to take the primary key of the "one's" side over to the "many's" side. In our case, because Manages is reflexive, this key would be the Manager's Social Security Number, denote as manager SSN. managerSSN must also be a foreign key since it references a different tuple in the same Employees table.

5. Provide a listing of any information, including constraints and cardinalities that could not be represented in the relational model using the basic SQL constraints like primary keys, foreign keys, unique and not null.

The only thing that the basic SQL constraints cannot capture is the cardinality constraint that an Employee who is also a manager, cannot manage more than 100 people.

Problem 3 (University Database) [30 Points]

Create a Relational Model (the CREATE TABLE statements) for the following database:

- We have students; each student has a unique Id, name, address, gender, and overall GPA (has default value of 0).
- Each student must have one major (mandatory attribute) and optionally one minor.
- The model must check that the gender takes values either "Male" or "Female"
- We have courses, each course has a unique CourseId, title, and number of credits
- Students will register in courses in certain semesters. We need to keep track of the grade (number, NOT letter) that a student has received in a given course. The model should allow a student to take the same course in different semesters.

Your CREATE TABLE statements should clearly indicate the following: 1) The attribute names and their data types (choose appropriate types), 2) The primary keys in each table, 3) The NULL (or NOT NULL) and the DEFUALT constraints, 4) The CHECK domain constrains, and 5) The foreign key constraints.

Assumptions: it wouldn't make sense to have a Student or Course with a name/title as NULL, so a NOT NULL constraint is used.

```
CREATE TABLE Students
        (Id INTEGER,
        name VARCHAR(32) NOT NULL,
        address VARCHAR (64),
        gender VARCHAR (6),
        gpa REAL DEFAULT 0,
        major VARCHAR (32) NOT NULL,
        minor VARCHAR (32),
        PRIMARY KEY(Id),
        CHECK(gender in ('Male', 'Female')) );

CREATE TABLE Courses
        (CourseId INTEGER,
        title VARCHAR (32) NOT NULL,
        credits REAL,
        PRIMARY KEY (CourseId));

CREATE TABLE Registrations
        (studentId INTEGER,
        courseID INTEGER,
        grade INTEGER,
        term VARCHAR(10),
        PRIMARY KEY (studentId, CourseId, term),
        FOREIGN KEY (studentId) REFERENCES Students (Id),
        FOREIGN KEY (courseID) REFERENCES Courses (courseID));
```

Deliverables:

Students should submit a .pdf file containing their appropriately numbered ER diagrams, SQL statements, and other responses to questions. The file MUST be a .pdf file (other file types need to be converted to .pdf for submission).

Submission:

Submit via Canvas. Late submissions will not be accepted.

Submission notes:

1) Include your name on the sheet
2) Be sure that your submission is in .pdf format.
3) Handwritten solutions or pictures of handwritten solutions will not be accepted.