TÉCNICO LISBOA

Sistemas de Informação e Bases de Dados 2020/2021

Class 08: Translating E-A to SQL (cont)

Prof. Paulo Carreira

Class Outline

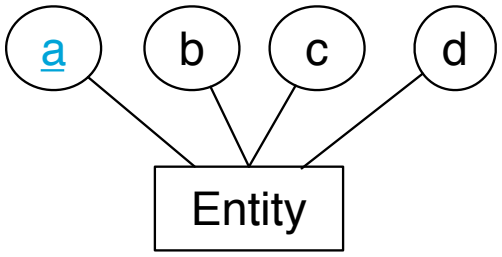
- Translating Column and Domain Constraints
- Translating Specialisation/Generalisation
- Translating Weak Entities
- Translating Aggregations

Translating Column and Domain Constraints

Column Constraints

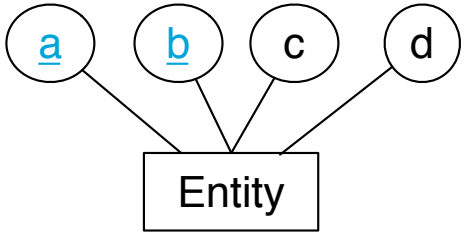
PRIMARY KEY Constraints

One Attribute




```
CREATE TABLE <table_name>(
        a INTEGER,
        b VARCHAR(80),
        c NUMERIC(12,4),
        d DATE,
        PRIMARY KEY(a)
);
```

Multiple atributes



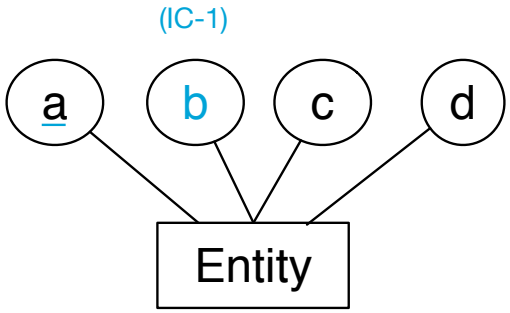

```
CREATE TABLE <table_name>(
        a INTEGER,
        b VARCHAR(80),
        c NUMERIC(12,4),
        d DATE,
        PRIMARY KEY(a, b)
);
```

A PRIMARY KEY constraint is specified as

PRIMARY KEY(*col*$_1$, …, *col*$_n$)

Uniqueness Constraints

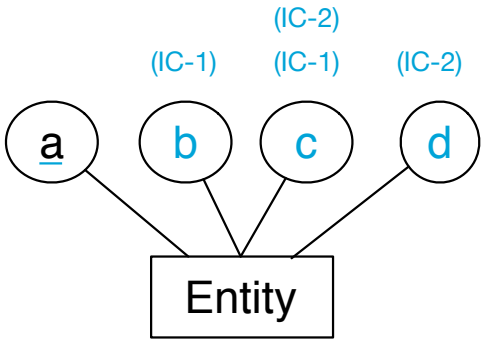
One “unique" attribute



Integrity Constraints:
(IC-1) b is unique

```
CREATE TABLE table_name(
      a INTEGER,
      b VARCHAR(80),
      c NUMERIC(12,4),
      d DATE,
      PRIMARY KEY(a),
      UNIQUE(b)
);
```

Combination of "unique"



Integrity Constraints:
(IC-1) (b,c) is unique
(IC-2) (c,d) is unique

```
CREATE TABLE table_name(
      a INTEGER,
      b VARCHAR(80),
      c NUMERIC(12,4),
      d DATE,
      PRIMARY KEY(a),
      UNIQUE(b, c),
      UNIQUE(c, d)
);
```

A uniqueness constraint is specified as:

UNIQUE(*col*$_1$, …, *col*$_n$)

Domain Constraints

Domain Constraint Checking

The **CHECK** clause can be used to specify the verification of the VALUES of any field of a record every time the record is inserted ou updated:

CHECK(*condition*)

```
CREATE TABLE products (
    product_no INTEGER,
    name VARCHAR(80),
    price NUMERIC,
    discounted_price NUMERIC,
    CHECK (price > 0),
    CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Domain Constraints

A Domain constraint guarantees that the VALUES of a column (field VALUES) are within the intended domain

```
CREATE TABLE employee(
    ssn NUMERIC(11),
    name VARCHAR(80) NOT NULL,
    birthdate DATE NOT NULL,
    gender CHAR(1),
    PRIMARY KEY(ssn),
    CHECK (length(name) > 3),
    CHECK (birthdate > '1920-01-01'),
    CHECK (gender in ('M', 'F'))
);
```

Domain Constraint validation using a Technical Table

Whenever the domain is too large, the valid VALUES can be validated against a technical table

```
CREATE TABLE employee
   (ssn NUMERIC(11),
    name VARCHAR(80) NOT NULL,
    birthdate DATE NOT NULL,
    birth_country CHAR(80),
    PRIMARY KEY(ssn),
    CHECK(birth_country
       IN (SELECT name FROM country))
);
```

Can be modelled as FOREIGN KEY

Record/Line Constraints

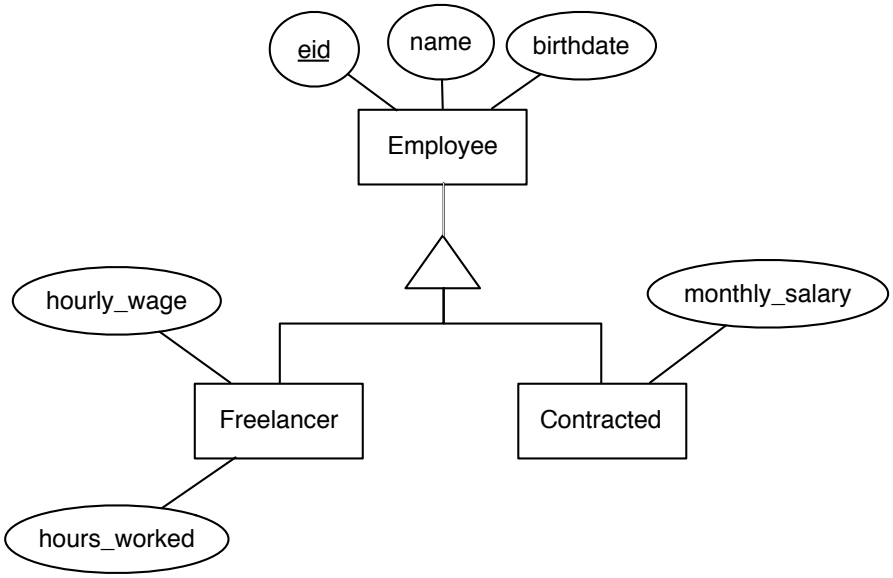
A record (row or line) constraint is one that guarantees that the data of the record (row or line) is correct coherent

```
CREATE TABLE employee(
    eid NUMERIC(9),
    name VARCHAR(80) NOT NULL,
    birthdate DATE NOT NULL,
    graduation DATE NOT NULL,
    PRIMARY KEY(eid),
    CHECK (LENGTH(name) > 3),
    CHECK (birthdate > '1920-01-01'),
    CHECK (extract(year FROM age(birthdate)) > 18),
    CHECK graduation > birthdate
);
```

Translating Generalisations

Simple Generalisation

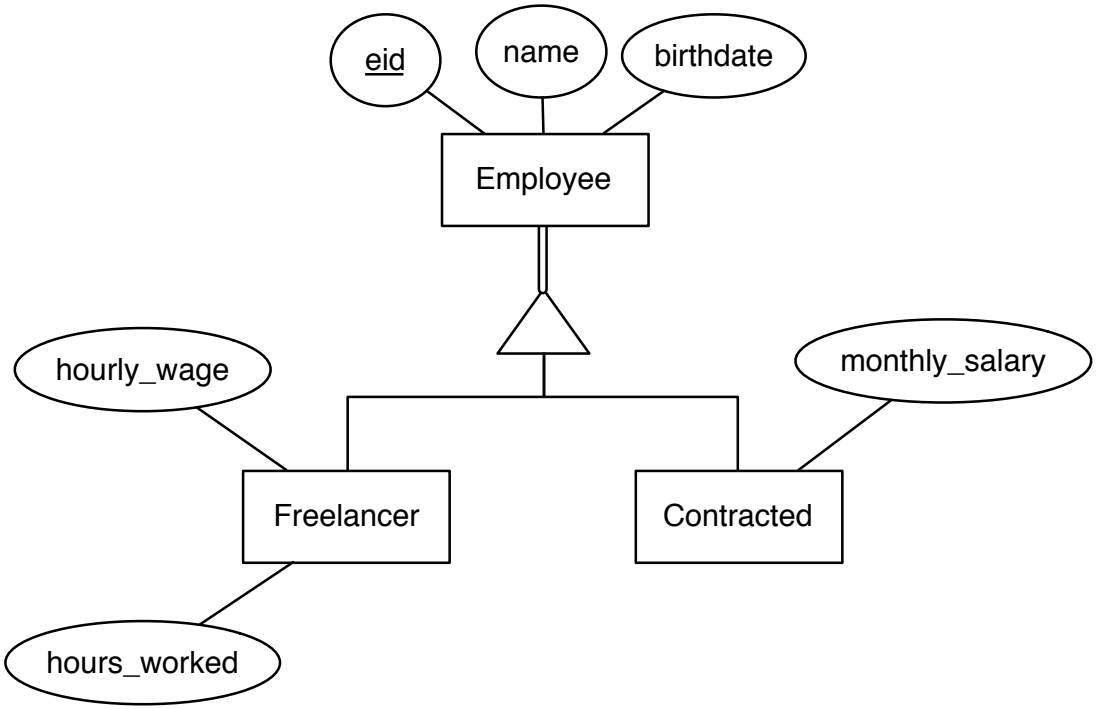
```
CREATE TABLE employee (
      eid INTEGER,
      name VARCHAR(80) NOT NULL,
      bdate DATE NOT NULL,
      PRIMARY KEY(empid)
);
```




```
CREATE TABLE freelancer (
      eid INTEGER,
      hourly_wage money,
      hours_worked INTEGER,
      PRIMARY KEY(eid),
      FOREIGN KEY(eid) REFERENCES
         employee(eid)
);
```

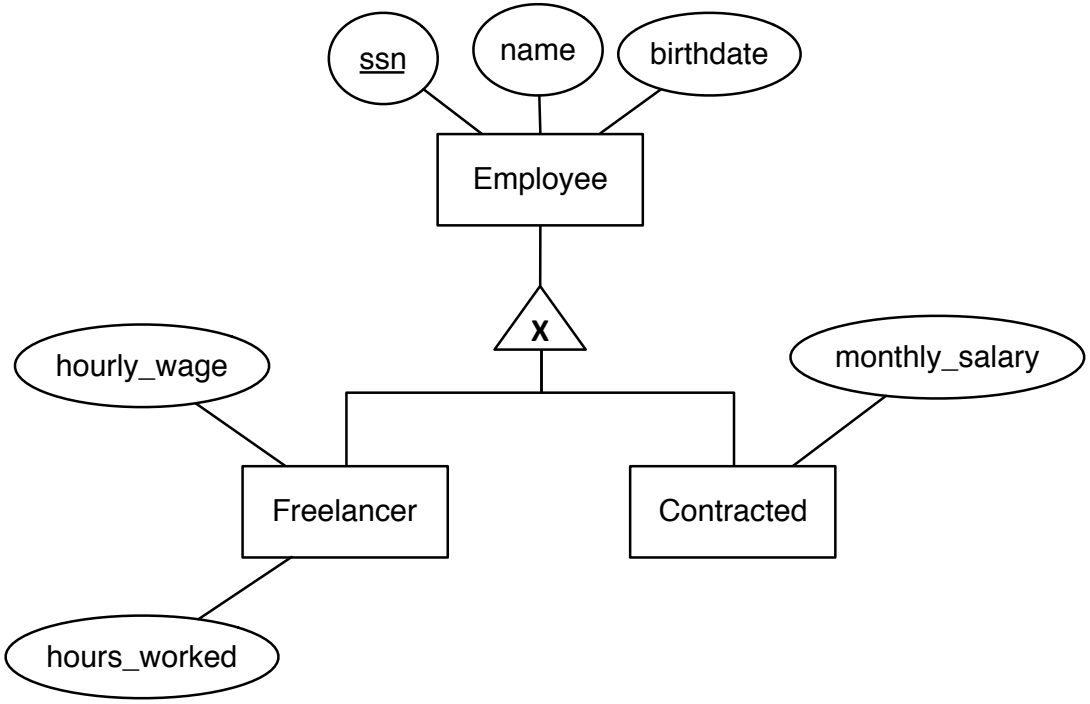
```
CREATE TABLE contracted (
      eid INTEGER,
      monthly_salary money,
      PRIMARY KEY(eid),
      FOREIGN KEY (eid) REFERENCES
         employee(eid)
);
```

Mandatory Specialisation



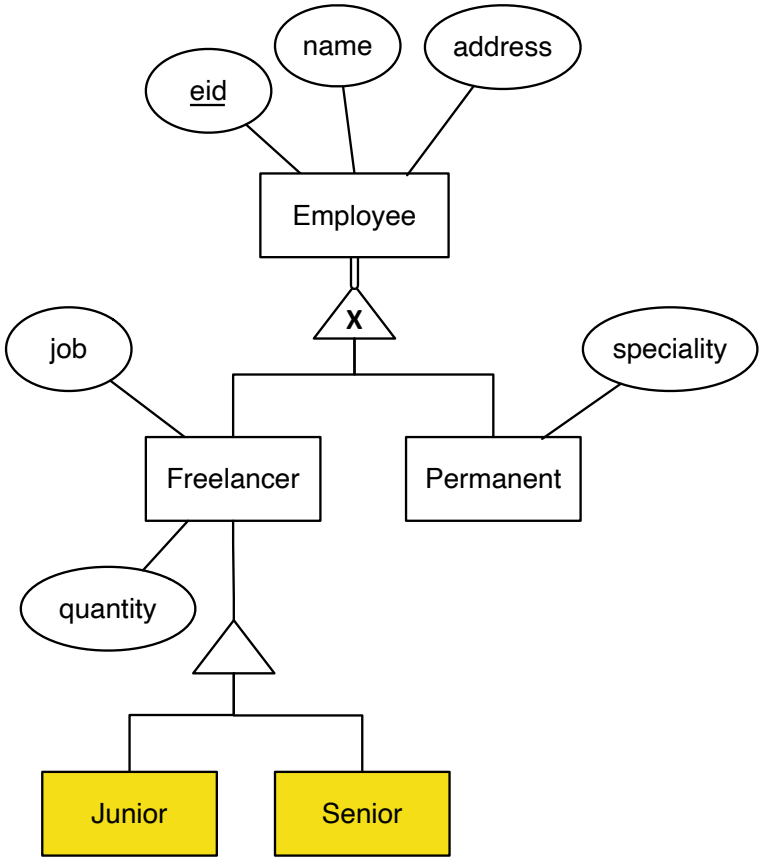
Every eid of Employee must exist either in Freelancer or in Contracted

Disjoint Specialisation



The *eid* of an Employee cannot exist in Freelancer and Contracted at the same time

Nested Specialisation




```
CREATE TABLE employee (
      eid INTEGER,
      name VARCHAR(80) NOT NULL,
      bdate DATE NOT NULL,
      PRIMARY KEY(empid)
);
CREATE TABLE freelancer(
      eid INTEGER,
      hourly_wage money,
      hours_worked INTEGER,
      PRIMARY KEY(eid),
      FOREIGN KEY(eid) REFERENCES
          employee(eid)
);
```

```
CREATE TABLE contrated(
      eid INTEGER,
      monthly_salary money,
    PRIMARY KEY (eid),
```

```
create table junior (
      eid INTEGER,
      PRIMARY KEY (eid),
      FOREIGN KEY (eid) REFERENCES
         freelancer(eid)
);
```

```
CREATE TABLE senior (
      eid INTEGER,
      PRIMARY KEY (eid),
      FOREIGN KEY (eid) REFERENCES
         freelancer(eid)
);
```

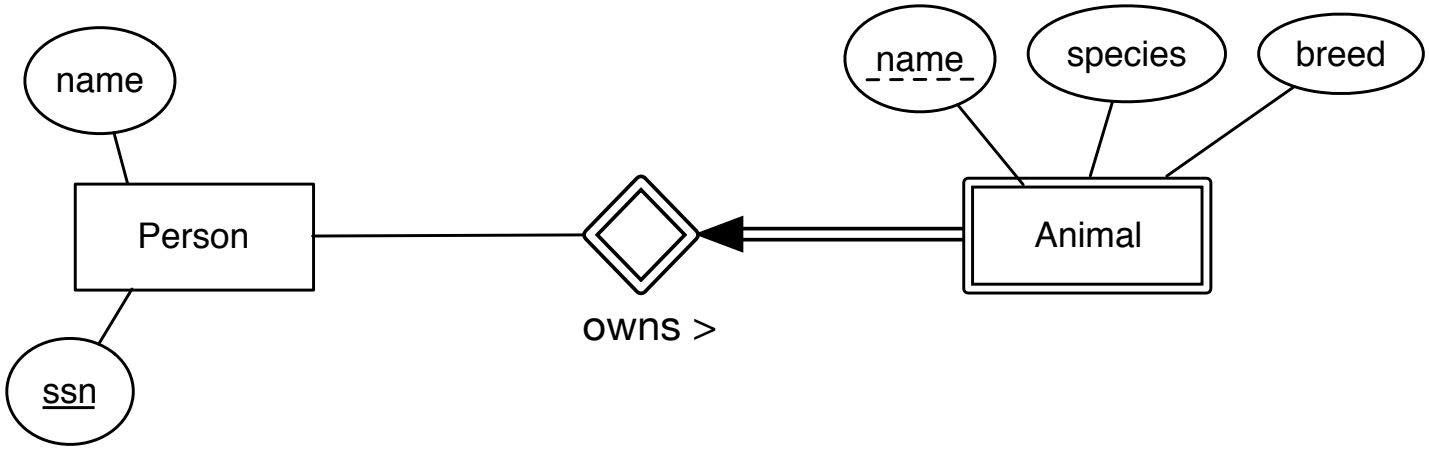
Mapping Generalisations/ Specialisations

1. Map the super-entity in a table
2. Map sub-entities into distinct tables where:
   - The key of each table corresponding to a sub-entity is the key of the super-entity (enforced with the corresponding FK constraint)
3. Disjoint or Mandatory specialisation constraints will mapped through ICs over the super-entity (using advanced database programming or application code)

⚠There is no simple DBMS implementation for **Disjoint** and **Mandatory** constraints. This will be explained later in the course.

Translating Weak Entities

Weak Entities




```
CREATE TABLE person(
      name VARCHAR(80),
      ssn NUMERIC(9)
      PRIMARY KEY(ssn)
);
```

```
CREATE TABLE animal(
    ssn NUMERIC(9)
    name VARCHAR(80),
    species VARCHAR(20),
    breed VARCHAR(20),
    PRIMARY KEY(ssn, name),
    FOREIGN KEY(ssn) REFERENCES person(ssn)
);
```

The translation is similar to the case of M:1 Mandatory Participation

Weak Entities

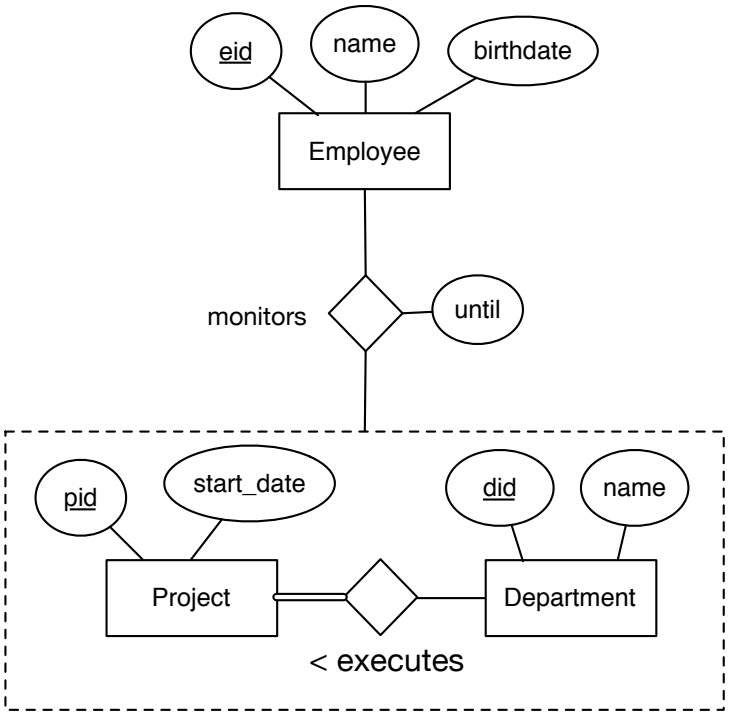
The **Weak Entities** originate a table that has a key composed by:

1. The association key that corresponds to the strong entity
2. The specified partial key
3. Any attributes of the weak entity (if they exist)

The association is not converted into a table

Translating Aggregations

Aggregation




```
CREATE TABLE monitors(
     eid INTEGER,
     pid INTEGER,
     did INTEGER,
     until date,
     PRIMARY KEY (eid, pid, did),
     FOREIGN KEY (pid, did) REFERENCES executes(pid, did)
     FOREIGN KEY (eid) REFERENCES employee(eid)
);
```

```
CREATE TABLE executes(
      pid INTEGER,
      did INTEGER,
      PRIMARY KEY (pid, did)
      FOREIGN KEY (pid) REFERENCES project(pid)
      FOREIGN KEY (did) REFERENCES department(did)
);
```

Aggregation

An Aggregation is mapped as an association where:

1. The interior of the aggregation is mapped to a table
2. The association with the aggregation is mapped into a table