TÉCNICO LISBOA

Sistemas de Informação e Bases de Dados 2020/2021

Class 07: Translating E-A to SQL

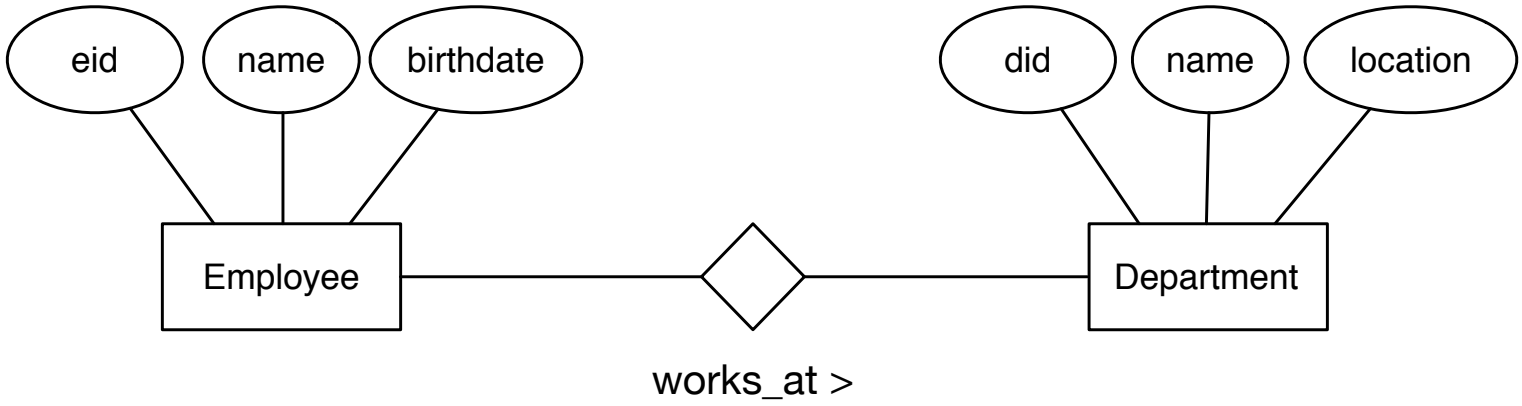
Prof. Paulo Carreira

Class Outline

- Motivation
- Revisiting Referential Integrity
- Translating Entities and Attributes
- SQL Data Types
- Translating Associations

Motivation

Translation Example



Employee

eid	name	birthdate
1	Alice	10/10/1995
2	Bob	03/02/1996
3	Caroline	04/04/1997
4	Daniel	03/04/1998
5	Eduard	10/03/1994

works_at

eid	did
1	1
2	2
3	3
4	3
5	3

Department

did	name	location
1	Finance	Buraca
2	Marketing	Damaia
3	Sales	Chelas

⚠ **works_at** encodes the association of Employees to Departments
All VALUES must be coherent

Relations and Attributes

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

```
CREATE TABLE department(
      did INTEGER,
      name  VARCHAR(20) NOT NULL,
      location VARCHAR(20) NOT NULL,
      PRIMARY KEY (did)
);
```

```
INSERT INTO employee VALUES(1, 'Alice', '1995-10-10');
INSERT INTO employee VALUES(2, 'Bob', '1996-03-02');
```

```
INSERT INTO department VALUES(1, 'Finance', 'Buraca');
INSERT INTO department VALUES(2, 'Marketing', 'Damaia');
```

Relations and Attributes

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

```
CREATE TABLE department (
      did INTEGER,
      name  VARCHAR(20) NOT NULL,
      location VARCHAR(20) NOT NULL,
      PRIMARY KEY(did)
);
```

```
CREATE TABLE works_at(
      eid INTEGER,
      did INTEGER,
      PRIMARY KEY (eid, did)
      FOREIGN KEY(eid) REFERENCES employee(eid),
      FOREIGN KEY(did) REFERENCES department(did)
);
```

Prevents inputing invalid *eid* or *did* values

```
INSERT INTO works_at VALUES(1, 1);
INSERT INTO works_at VALUES(2, 1);
INSERT INTO works_at VALUES(2, 99);
```

Referential Integrity

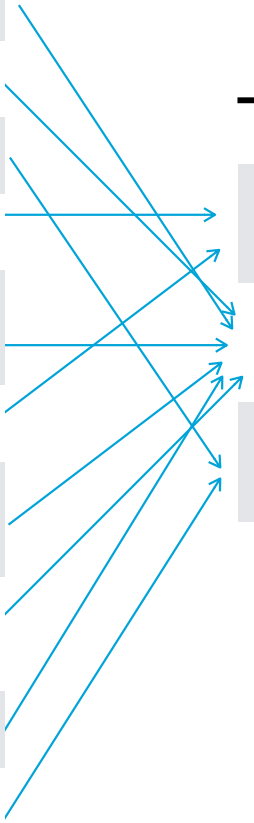
Table Relationships

A foreign key connects two tables

Foreign Key

Employee

ID	Name	Tax id	T-Shirt	DID
001	João Guilherme Silva da Cunha	12345678	M	EN
002	Tomás Pinto dos Santos	91234567	M	EN
003	David Miguel Redwanz Duque	89012345	L	MK
004	Pedro Daniel Diz Pinela	67890123	M	HR
005	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL	EN
006	Marta Isabel de Almeida Cardoso	34562732	S	HR
007	Filipe Emanuel Lourenço Ramalho Fernandes	82533235	L	EN
008	Gabriel Filipe Queirós Mesquita Delgado Freire	23134539	M	EN
009	João Gomes Vultos Freitas	22231233	L	EN
010	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L	MK

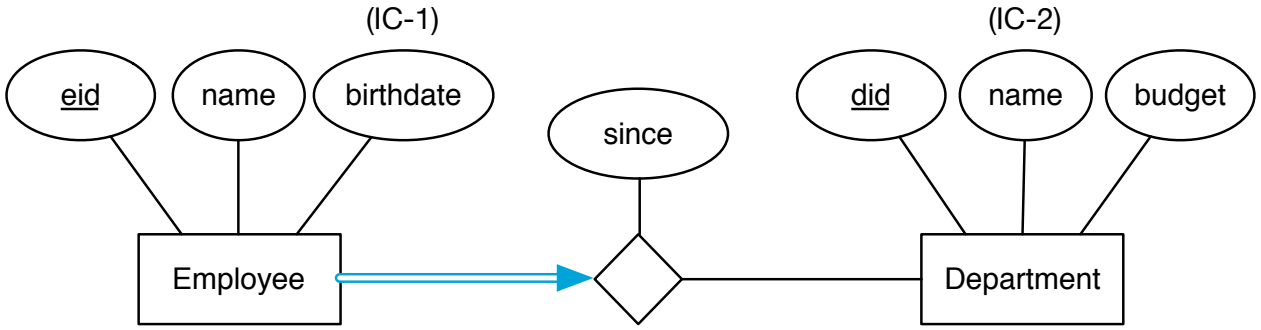


Department

DID	Name	Budget
HR	Human Resources	50 000
EN	Software Engineering	1 200 000
MK	Marketing	150 000

Primary Key

How to translate this case?



Integrity Constraints:
(IC-1) an employee must be at least 18 years old
(IC-2) Department names must be unique

⚠ All VALUES in Employee.dep must exist as PRIMARY KEY the Department.did

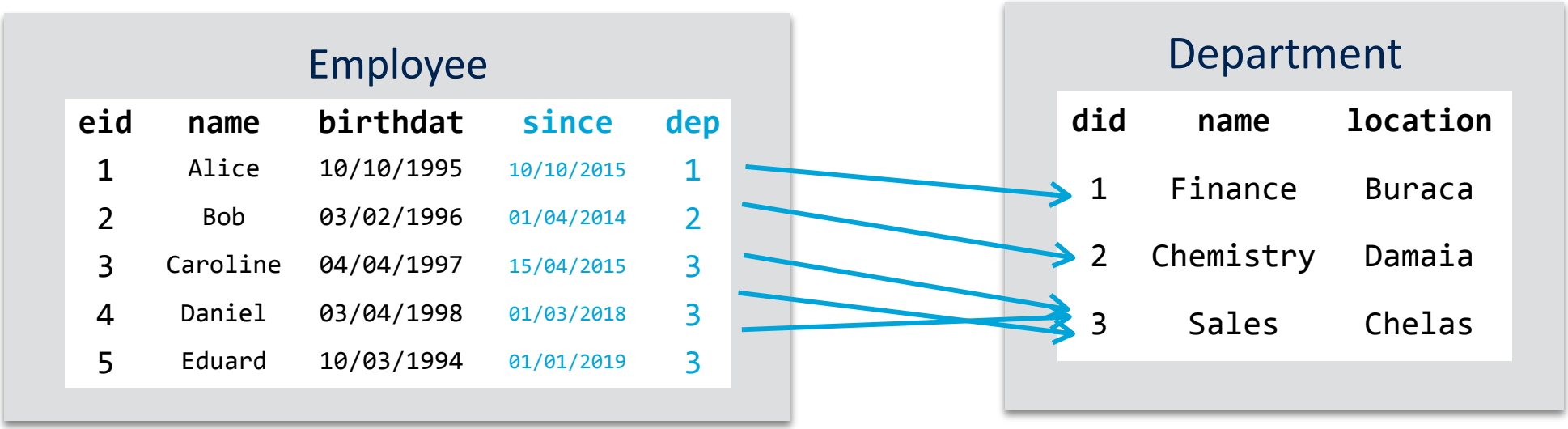
Employee

eid	name	birthdat	since	dep
1	Alice	10/10/1995	10/10/2015	1
2	Bob	03/02/1996	01/04/2014	2
3	Caroline	04/04/1997	15/04/2015	3
4	Daniel	03/04/1998	01/03/2018	3
5	Eduard	10/03/1994	01/01/2019	3

Department

did	name	location
1	Finance	Buraca
2	Chemistry	Damaia
3	Sales	Chelas

Operations that violate Referential Integrity



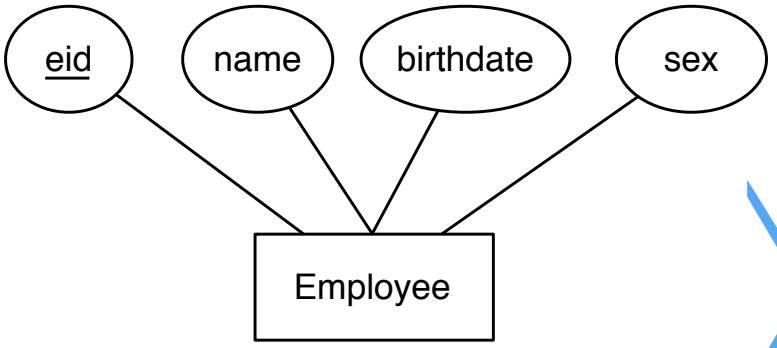
- **Removing** lines from *Department*: We cannot remove *departments* to which *employees* are still associated
- **Updating** VALUES on *Department*: We cannot change VALUES on d*epartment* that imply changing VALUES on *employee*
- **Inserting** lines on *Employee*: We cannot add *employees* on *departments* that do not exist

Referential Integrity Constraints (or Foreign Keys)

- The most common constraint involving two tables is the Referential Integrity constraint
- Data in one table must be always coherent with the data of another table. A table is usually related to other tables.

Translating Entities and Attributes

Entities




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

1. **Entities** result in a table with corresponding attributes
2. PRIMARY KEY is the same
3. Constraints have also to be translated (will see how later on…)

Data Types and Values

Datatype Families

Text Types

Varchar	Char	Text

```
'John Smith', 'R2D2', 'Red', ''
```

Numeric Types

Character strings

Integer	Fixed Point	Floating Point

```
-1, 25, +6.34, 0.5, 25e-03
```

Date & Time

Date	Time	Timestamp

```
'2029-01-01', '08-JAN-2029 10:35:02'
```

Text Types

VARCHAR(*n*)	CHAR(*n*)	TEXT
Variable length character string max size *n*	Character string with fixed size *n*	Variable length text (multi-line) field (typically limited to 65535 characters)
n < 4000	*n < 4000*	*length typically limited to 65535 characters*

Integer Numeric Types

INTEGER	SMALLINT	BIGINT
An integer with up to 9 digits	An integer with up to 4 digits	An integer with up to 18 digits
4 bytes	2 bytes	8 bytes
-2^{31} -2 147 483 648 to $+2^{31}-1$ 2 147 483 647	-2^{15} -32 768 to $+2^{15}-1$ 32 767	-2^{63} -9 223 372 036 854 775 808 to $+2^{63}-1$ 9 223 372 036 854 775 807

Fixed Point Number Types

NUMERIC(*p*, *s*)

A numeric value with arbitrary exact precision

≈(3+*p*/4) bytes

Up to 131072 digits before the decimal point

Up to 16383 digits after the decimal point

- Precision ***p***: total number of digits, must be positive (***p*** > 0)
- Scale ***s***: number of digits to right of the decimal point, can be zero but must always be smaller than the precision (0 < ***s*** < ***p***)

Whenever ***s*** is zero, we can write **NUMERIC(*p*)**

Floating Point Numeric Type

REAL	DOUBLE
variable-precision, inexact 6 digits	variable-precision, inexact 15 digits
4 bytes	8 bytes
1E-37 to 1E+37	1E-307 to 1E+308

Fixed- vs. Floating Point

	FIXED POINT	FLOATING POINT
Precision	Fixed precision	Variable-precision
Storage	Stores numbers exactly	Stores numbers inexactly (as approximations)
Retrieval	Retrieved valued never show any discrepancies	Retrieved values may show discrepancies from values stored
Speed	Slower calculations (*)	Faster calculations
Money Amounts	Can be safely used for monetary values	Should never be used for monetary values

(*) In practice, this difference is often neglectable

Behaviour of Fixed and Float

Behaviour of Fixed Point

```
CREATE TABLE teste(
    x NUMERIC(1,0)    – same as NUMERIC(1)
);
```

```
INSERT INTO teste VALUES (0);
```

```
INSERT INTO teste VALUES (-1);
```

```
INSERT INTO teste VALUES (9);
```

```
INSERT INTO teste VALUES (9.1);
```
⚠ rounded

```
INSERT INTO teste VALUES (11);
```
✕ error

⚠ Inserting in a NUMERIC with a precision or scale larger than specified will result in an error or in a warning; the value is often truncated.

Behaviour of Fixed Point

```
CREATE TABLE teste(
    x NUMERIC(4,2)
);
```

```
INSERT INTO teste VALUES (0.0);
```

```
INSERT INTO teste VALUES (-1.2);
```

```
INSERT INTO teste VALUES (12.34);
```

```
INSERT INTO teste VALUES (12.345);
```

⚠ Rounds to 12.35

Behaviour of Float

```
CREATE TABLE test(
 x float,
 y float
);

INSERT INTO test VALUES(1.2, 1.2);

SELECT x+y FROM test;
```

```
2.4000000953674316
```

```
SELECT (1.0/3.0)*3.0;
```

Date Types

DATE	TIME	TIMESTAMP
Stores dates with a resolution of 1 day	Stores a time with a resolution of 1 microsecond	Stores instant timestamps with resolution of 1 microsecond
4 bytes	8 bytes	8 bytes
4713 BC to 5874897 AD	00:00:00 to 24:00:00	4713 BC to 294276 AD

Intervals

- **interval** (16 bytes) – enables capturing the difference between dates, times, or timestamps

⚠ Dates and times cannot be added. However, we can add intervals to dates and intervals to hours.

Indicative Field Sizes

Person/Organisation

Field	Database Type	Max Size	Min Size	Validation
PERSON/ORGANIZATION DETAILS				
Person Full Name	VARCHAR	80		
Company Name	VARCHAR	200		
Street Address	VARCHAR	255		
City	VARCHAR	30		
Postal Code	VARCHAR	12	2	
Phone Number	VARCHAR	15	3	ITU E.16
Phone Extension	VARCHAR	11		ITU E.16
Language	CHAR	3		ISO 639
Country Name	VARCHAR	70		ISO 3166-1
Latitude	NUMERIC	9,6		
Longitude	NUMERIC	8,6		

Finance

Field	Database Type	Max Size	Min Size	Validation
FINANCE				
VAT ID	VARCHAR	20	1	
IBAN	VARCHAR	30		
Credit Card Number	NUMERIC	16		
Money	NUMERIC	16,4		

Electronic Commerce

Field	Database Type	Max Size	Min Size	Validation
ELECTRONIC				
E-mail Address	VARCHAR	254	6	IETF RFC 3696 Checking email addresses
Domain Name	VARCHAR	253	4	
URL	VARCHAR	2083	11	
IP address (incl V6)	VARCHAR	45	11	
GUID	char	36		

Social Networks

Field	Database Type	Max Size	Min Size
SOCIAL NETWORK			
Facebook max name length	VARCHAR	50	
Youtube channel	VARCHAR	20	
Twitter max name length	VARCHAR	15	

NULL VALUES

NULL is a special value that means, simultaneously, unfilled / unknown / not applicable

Suppose that Daniel did not specify his birthdate, we could write the INSERT statement:

```
INSERT INTO employee VALUES(11, 'Daniel', null);
```

and then query the database for his record:

```
SELECT * FROM employee
WHERE eid=11;
```

```
eid  | name      | bdate
-----+-----------+-------
11   | Daniel    |
```

⚠ The use of NULL is ambiguous and should be avoided.

NOT NULL Constraint

- In SQL all columns (not part of the key) by default may have null VALUES.
- To prevent columns from taking null VALUES, we must add the **NOT NULL** constraint in front of the data type:

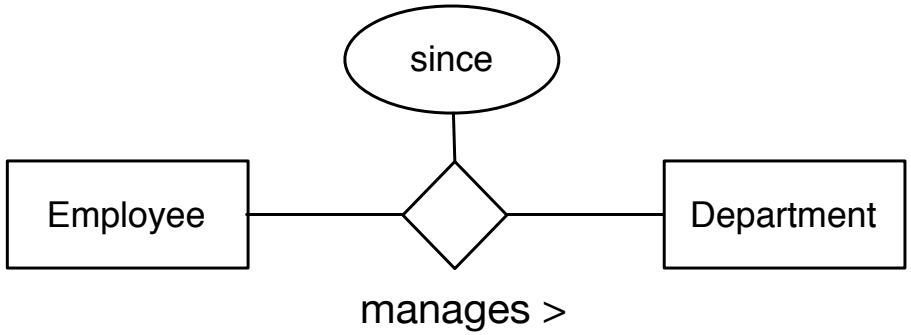
<field> <type> **NOT NULL**

```
CREATE TABLE employee(
    ssn NUMERIC(11),
    name VARCHAR(80) NOT NULL,
    birthdate DATE NOT NULL,
    PRIMARY KEY(ssn)
);
```

- Most columns (most often all columns) should be NOT NULL
- Any columns that participate in the PRIMARY KEY are already NOT NULL (because they null is never a valid value on a key)

Translating Associations

M:N Associations

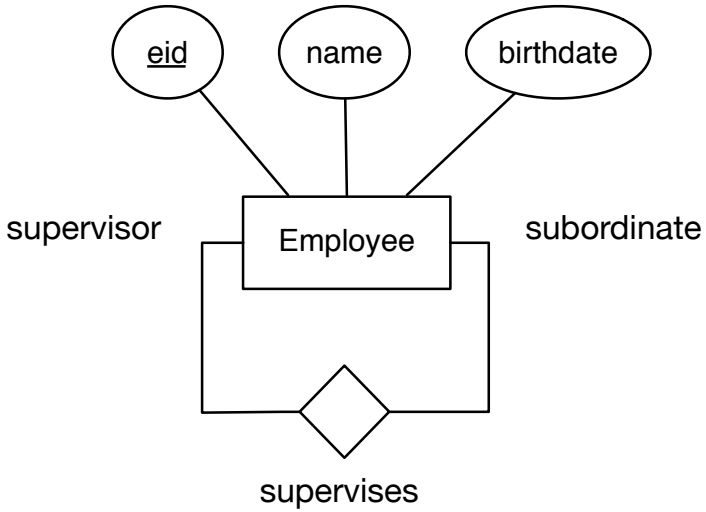



```
CREATE TABLE manages (
      eid INTEGER,
      did INTEGER,
      since DATE,
      PRIMARY KEY(eid, did),
      FOREIGN KEY (eid) REFERENCES employee(eid),
      FOREIGN KEY (did) REFERENCES department(did)
);
```

Captures any valid combination of ⟨eid, did⟩

M:N Auto Association

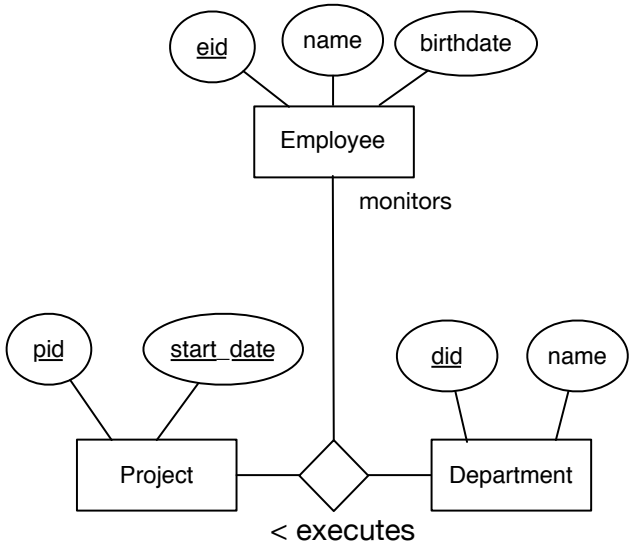
Special case of the self-association




```
CREATE TABLE supervises (
      sup_eid INTEGER,
      sub_eid INTEGER,
      PRIMARY KEY(sup_eid, sub_eid)
      FOREIGN KEY(sup_eid) REFERENCES
            Employee(eid),
      FOREIGN KEY(sub_eid) REFERENCES
            Employee(eid)
);
```

- Field names cannot repeat
- The fields sup_eid and sub_eid capture any valid combination of ⟨eid, eid⟩

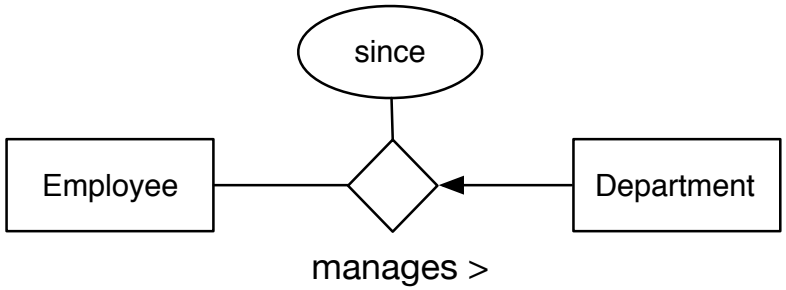
Ternary Associations




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

Captures any valid combination of ⟨eid, pid, did⟩

One-to-many Associations

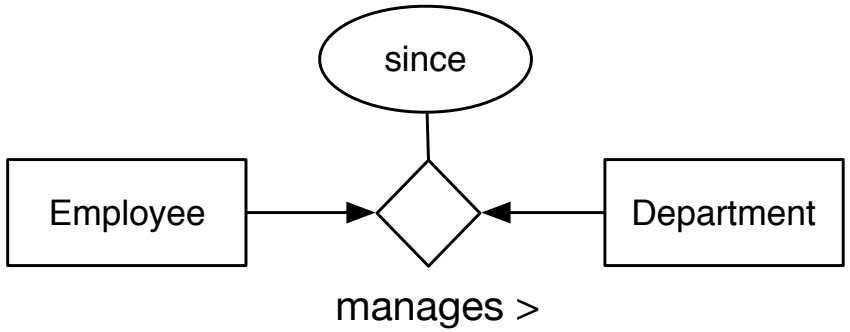


Guarantees that VALUES of ***did*** *can never repeat!*

```
CREATE TABLE manages (
      eid INTEGER,
      did INTEGER,
      since DATE,
      PRIMARY KEY(did),
      FOREIGN KEY(eid) REFERENCES employee(eid),
      FOREIGN KEY(did) REFERENCES department(did)
);
```

- Once a department is associated to an employee, it cannot be associated again (to another employee)
- We encode this by guaranteeing that each did appears only once in the table that represents the association (i.e., that did it is associated only once)

One-to-one Associations



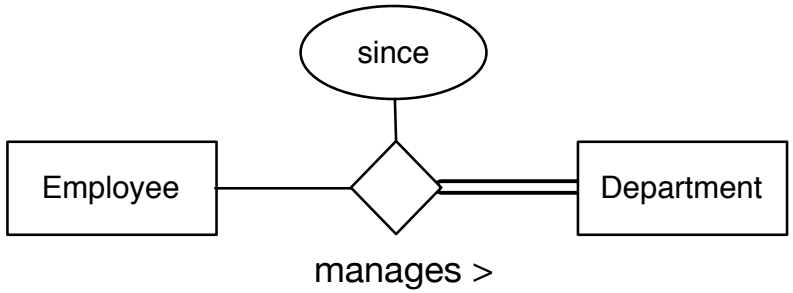
Neither *did* nor *did* repeat

Therefore: Once a '*department*' (or an '*employee*') exists in the table '*manages*', no other can exist

```
CREATE TABLE Manages (
      eid INTEGER,
      did INTEGER,
      since DATE,
     PRIMARY KEY(eid)
      UNIQUE (did),
      NOT NULL(did),
      FOREIGN KEY(eid) REFERENCES employee(eid),
      FOREIGN KEY(did) REFERENCES department(did)
);
```

- Both a Department and a Employee can only be associated once
- We encode this by guaranteeing that both eid and did appear only once

M:N Mandatory Participation

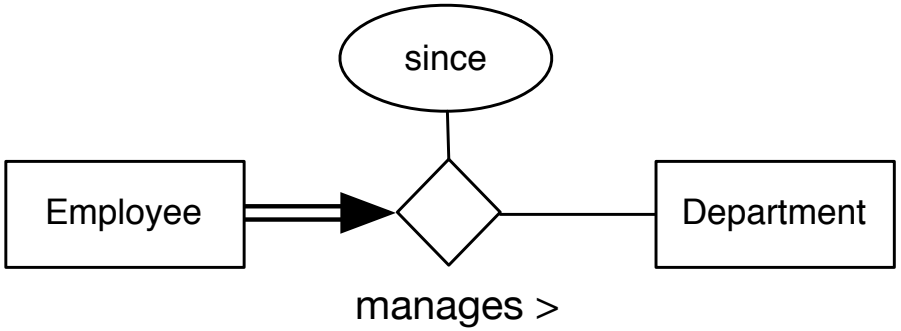


Every *department (did)* must participates in the '*manages*' association

```
CREATE TABLE manages (
      eid INTEGER,
      did INTEGER,
      since date,
      PRIMARY KEY(eid, did),
      FOREIGN KEY(eid) REFERENCES employee(eid),
      FOREIGN KEY(did) REFERENCES department(did)
);
```

⚠There is no simple DBMS implementation for this constraint. Must often ensured by the application code.

Many-to-one Mandatory Participation




```
CREATE TABLE employee (
      eid INTEGER,
      name  VARCHAR(80) NOT NULL,
      bdate DATE NOT NULL,
      since DATE NOT NULL,
      did INTEGER NOT NULL,
      PRIMARY KEY(eid)
      FOREIGN KEY(did) REFERENCES department(did)
);
```

- Instead of creating a table for the 'manages' association, extend the table employee with the reference (a foreign key) to the department.
- Every record on employee must be connected to (in this case managing some) department