DA 512H: Database Management Systems

Domain Constraints

Debanga Raj Neog
Mehta Family School of Data Science and Artificial Intelligence (MFSDS&AI)
IIT Guwahati

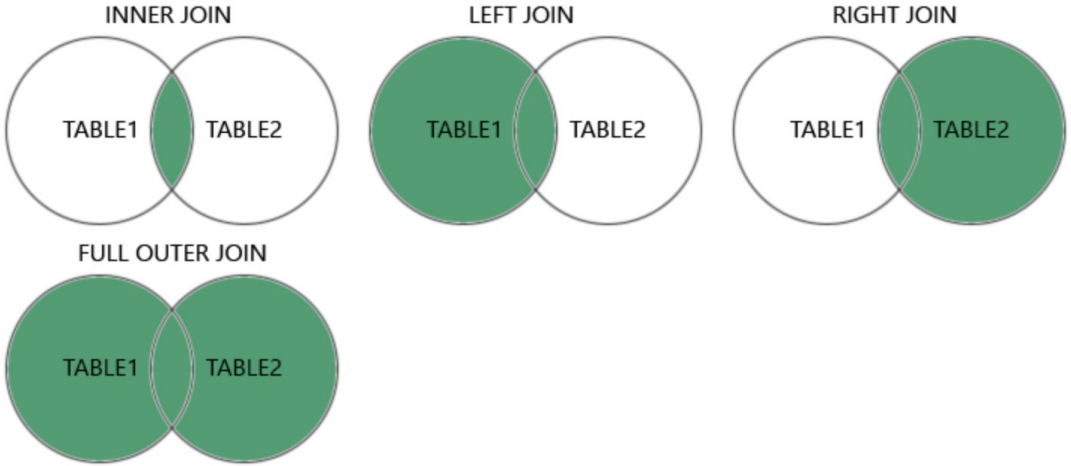
Slides courtesy:
Prof. Ashok Singh Sairam, IITG

JOINS

- **(INNER) JOIN:** Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN:** Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN:** Returns all records when there is a match in either left or right table

JOINS

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```



Domain Constraints

- Domain constraints ensure valid values for attributes
  - They test values inserted in the database, and test queries to ensure that the comparisons make sense
- The data type of attributes are string, integers, Boolean, binary, etc.

Students(name: **string**, rollno: **integer**, branch: **string**, CPI: **real**)

Entity Integrity constraints

- Entity integrity constraint ensures that an attribute (primary key) value cannot be NULL
- The primary key cannot be NULL since it will be used to identify a tuple
  - Other fields may be NULL

```
CREATE TABLE student (
        name varchar(28),
        rollno INT(7) not null,
        branch varchar(20),
        cpi decimal(4,2)
        );
```

Key Constraints

- Keys are one or more attributes used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys (candidate keys), but out of which one key will be the primary key
- A **primary key** contain a unique value in the relational table
- A **foreign key** is an attribute which matches a primary key in a different relation

Referential Integrity

- Referential Integrity
  - Referential integrity refers to the accuracy and consistency of data within a relationship
  - Achieved by having a **foreign** key
- A referential integrity must be specified between two relations

Enforcing Integrity Constraints

- **Integrity Constraints** - condition that must be true for *any* instance of the database;
  - ICs are **specified** when schema is **defined**.
  - ICs are **checked** when relations are **modified**.

Enforcing Integrity Constraints

- **Integrity Constraints** - condition that must be true for *any* instance of the database; e.g., **domain constraints**
  - ICs are **specified** when schema is **defined**.
  - ICs are **checked** when relations are **modified**.
- A **legal** instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Key constraints

- A set of fields is a **candidate key** for a relation if:
  1. No two distinct tuples can have same values in all key fields
  2. This is not true for any subset of the key

Key constraints

- A set of fields is a **candidate key** for a relation if :

  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.

  Example “rollno” number is a candidate key

sname	rollno	branch	cpi
John	12345	MnC	9.1
John	34567	MnC	9.1
Jones	45678	CSE	10

Key constraints

- A set of fields is a **candidate key** for a relation if :

  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.

  Example “room_no, Hostel” number is a candidate key

sname	room_no	Hostel	branch
John	G-101	Siang	MnC
John	G-101	Barak	MnC
John	G-102	Siang	CSE

Key constraints

- A set of fields is a **candidate key** for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.

- If part 2 false? A **superkey**.
  {*room_no, hostel, branch*} is an example of a superkey

sname	room_no	Hostel	branch
John	G-101	Siang	MnC
John	G-101	Barak	MnC
John	G-102	Siang	CSE

Primary key

- If there's more than one candidate key for a relation, one of the keys is chosen (by the Database Administrator) to be the **primary key**

Primary key

- If there's more than one candidate key for a relation, one of the keys is chosen (by the Database Administrator) to be the **primary key**
  - For example, both rollno and mobile_no are candidate keys
  - The DBA can choose one of them (say rollno) as the primary key

sname	rollno	mobile_No	branch	cpi
John	12345	1234567890	MnC	9.1
John	34567	2345678901	MnC	9.1
Jones	45678	3456789012	CSE	10

Primary keys - continued

- We can use any key to refer to a tuple, not necessarily the primary key
- However, the database expects the tuples will be frequently referred using the primary key and accordingly optimizes
  - The DBMS may create an index with the primary key as the search key to make search

Specifying key constraints in SQL

- We can name a constraint by preceding it with CONSTRAINT
  - constraint <*constraint-name*>
- If the constraint is violated, the constraint name helps in identifying the error.

```
CREATE TABLE student (
        name varchar(28),
        rollno INT(7),
        branch varchar(20),
        cpi decimal,
        primary key(rollno)
        );
```

```
CREATE TABLE student (
        name varchar(28),
        rollno INT(7),
        branch varchar(20),
        cpi decimal,
      constraint StudentKey primary key(rollno)
          );
```

Primary key: practical considerations

- An integer primary key is more efficient during join operations
- Consider you have an entity user with two attributes – name and email id
  - Although email id can be used as a key it may result in performance issues
  - Use auto_increment value as key. A unique no. is automatically generated when a new tuple is inserted into the table

```
CREATE TABLE Users (
        user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
        name VARCHAR(128),
        email VARCHAR(128),
        PRIMARY KEY(user_id),
        INDEX(email)
);
```

Foreign Key

- **Foreign key** - A **Foreign Key** is a column or a combination of columns whose values match a **Primary Key** in a different table. Like a **logical pointer**.
- Suppose we want to ensure the following condition “Only bonafide students can enroll in a course”

  student(sname: string, rollno: integer, subject: string, cpi: decimal)

  enrolled(cid: string, grade: string, rollno: integer)
  - Note that the field rollno is a *foreign key* and it refers to the *primary key* in the relation student

Foreign keys in SQL

Only students listed in the Students relation should be allowed to enroll for courses

```
create TABLE enrolled(
    cid varchar(5),
    rollnumber integer(7),
    grade char(2),
    PRIMARY key(cid, rollnumber),
    FOREIGN KEY (rollnumber) REFERENCES student(rollno)
    );
```

Example: Foreign keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

Enrolled

cid	rollnumber	Grade
MA518	**12345**	AB

Students

sname	rollno	branch	CPI
John	12345	MnC	9.1
James	23456	CSE	8.7
Tom	34567	MnC	9.0

Example: Foreign keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

Enrolled

cid	rollnumber	Grade
MA518	**12345**	AB
CS348	**12345**	BB

Students

sname	rollno	branch	CPI
John	12345	MnC	9.1
James	23456	CSE	8.7
Tom	34567	MnC	9.0

Example: Foreign keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

Enrolled

cid	rollnumber	Grade
MA518	**12345**	AB
CS348	**12345**	BB
MA518	**34567**	AB

Students

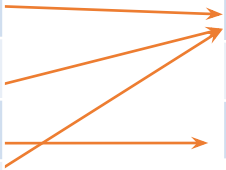
sname	rollno	branch	CPI
John	12345	MnC	9.1
James	23456	CSE	8.7
Tom	34567	MnC	9.0

Example: Foreign keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses

Enrolled

cid	rollnumber	Grade
MA518	**12345**	AB
CS348	**12345**	BB
MA518	**34567**	AB
CS343	**12345**	BB



Students

sname	rollno	branch	CPI
John	12345	MnC	9.1
James	23456	CSE	8.7
Tom	34567	MnC	9.0

What is the key of the Enrolled table?

Enforcing Referential Integrity: Insertion

Consider Students and Enrolled; *rollnumber* in Enrolled is a foreign key that references Students.

Options:

What should be done if we try to insert a tuple in Enrolled with non-existent student *rollnumber*? (Reject it!)

On trying to insert a non-existent roll number the following error will occur

INSERT into enrolled(cid, rollnumber, grade) **VALUES**
("MA518", 99999, "AB");

```
- Cannot add or update a child row: a foreign key constraint fails (`marks`.`enr
```

Enforcing Referential Integrity: Deletion/Updation

- What should be done if a Students tuple is deleted?
  - Disallow deletion of a Students rows that is referred to
  - Also delete all Enrolled rows that refer to it (**cascade delete**)
  - Set *rollnumber* in Enrolled rows that refer to it to a **default value/Null** value, denoting **unknown** or **inapplicable**
- Similar if primary key of Students rows is updated

Referential Integrity in SQL

- Default is NO ACTION (*delete/update is rejected*)
- CASCADE (also delete all rows that refer to deleted row)
- SET NULL / SET DEFAULT (sets foreign key value of referencing row)

```
CREATE TABLE enrolled(
        cid varchar(5),
        rollnumber integer(7),
        grade char(2),
        primary key (cid, rollno),
        foreign key (rollnumber) references student(rollno)
        on delete cascade
    on update no action
    );
```

Querying relational data

Relational database query (query, for short) is a question about the data, and the answer consists of a new relation containing the result.

SELECT * **FROM** Students S
WHERE S.cpi < 8

- S is as a variable that takes on the value of each tuple in Student
- * means retain all the fields of the selected tuples in the result

sname	rollno	branch	cpi
Tom	23456	MnC	6.4
Tim	45678	MnC	7.4

Querying Multiple Relations

- Given the following instances of Enrolled and Student:

Enrolled

cid	rollnumber	grade
MA518	23456	AA
MA518	34567	AA
MA518	45678	AB

sname	rollno	branch	cpi
Tom	23456	MnC	6.4
John	34567	MnC	8.0
Tim	45678	MnC	7.4

Student

- We run the following query:

```
SELECT S.sname, S.rollno, E.cid, E.grade
    FROM student S, Enrolled E
    WHERE S.rollno=E.rollnumber AND E.grade="AA";
```

- We get

sname	rollno	cid	grade
Tom	23456	MA518	AA
John	34567	MA518	AA

Summary

- Primary key constraint: enforced during table creation by using the keyword **primary key (<attribute_name>)**
- Foreign key: Attribute(s) that match a primary key in a different relation
- Enforcing referential integrity during insertion and deletion

FOREIGN KEY (<attribute_name>) **REFERENCES**
<Table_name>(<primary_key>)

DA 512H: Database Management Systems

ER to Relations

Debanga Raj Neog
Mehta Family School of Data Science and Artificial Intelligence (MFSDS&AI)
IIT Guwahati

Slides courtesy:
Prof. Ashok Singh Sairam, IITG

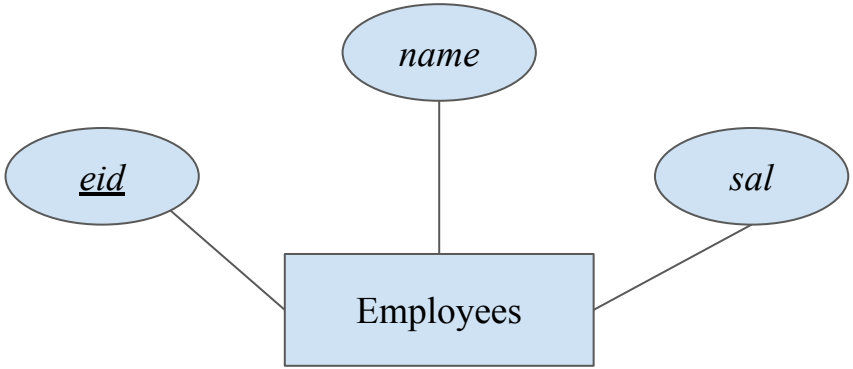
ER to Relational

- The ER model is useful for initial database design representation
- Our goal is to translate ER diagram into a collection of tables with associated constraints
- Results is a relational database schema that closely approximates the ER design

Entity Sets to Tables

- An entity set is mapped to a relation in a straightforward way
- Each attribute of the entity set becomes an attribute of the table
- Note that we know both the domain of each attribute and the (primary) key of an entity set

Entity Sets to Tables



Entity Sets to Tables

eid	name	sal
123-22-3666	Rahul	50000
231-31-5368	Abhishek	75000
131-24-3650	Aman	65000

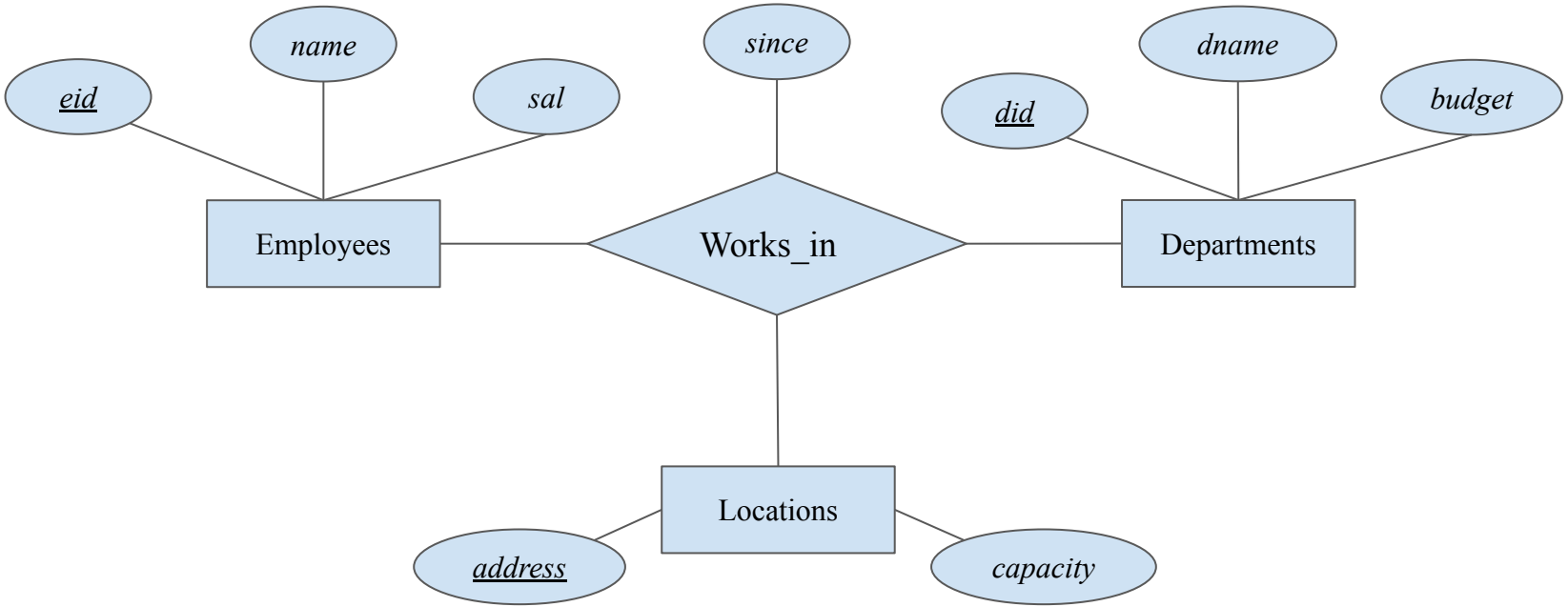
Entity Sets to Tables

```
CREATE TABLE Employees ( eid CHAR(11),
                         name CHAR(30) ,
                         sal INTEGER,
                         PRIMARY KEY (eid) );
```

Relationship Sets (without Constraints) to Tables

- A relationship set, like an entity set, is mapped to a relation (or table) in the relational model.
- To represent a relationship, we must be able to identify each participating entity and give values to the descriptive attributes of the relationship.
- Thus, the attributes of the relation include:
  - The primary key attributes of each participating entity set, as foreign key fields
  - The descriptive attributes of the relationship set

Relationship Sets (without Constraints) to Tables



Relationship Sets (without Constraints) to Tables

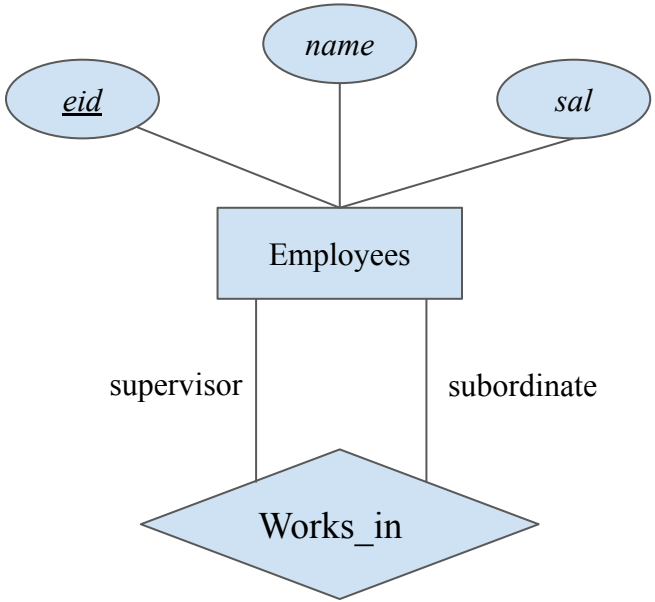
```
CREATE TABLE Departments ( budget INTEGER,
                           dname CHAR(30) ,
                           did INTEGER,
                           PRIMARY KEY (did) );

CREATE TABLE Locations ( address CHAR(30),
                         capacity INTEGER,
                         PRIMARY KEY (address) );
```

Relationship Sets (without Constraints) to Tables

```
CREATE TABLE Works_in ( eid CHAR(11),
        did INTEGER,
        address CHAR(20),
        since DATE,
        PRIMARY KEY (eid, did, address),
        FOREIGN KEY (eid) REFERENCES Employees (eid),
        FOREIGN KEY (address) REFERENCES Locations (address),
        FOREIGN KEY (did) REFERENCES Departments(did));
```

- The address, did and eid fields cannot take on null values, because these fields are part of the primary key for Works_in, a NOT NULL constraint is implicit for each of these fields.

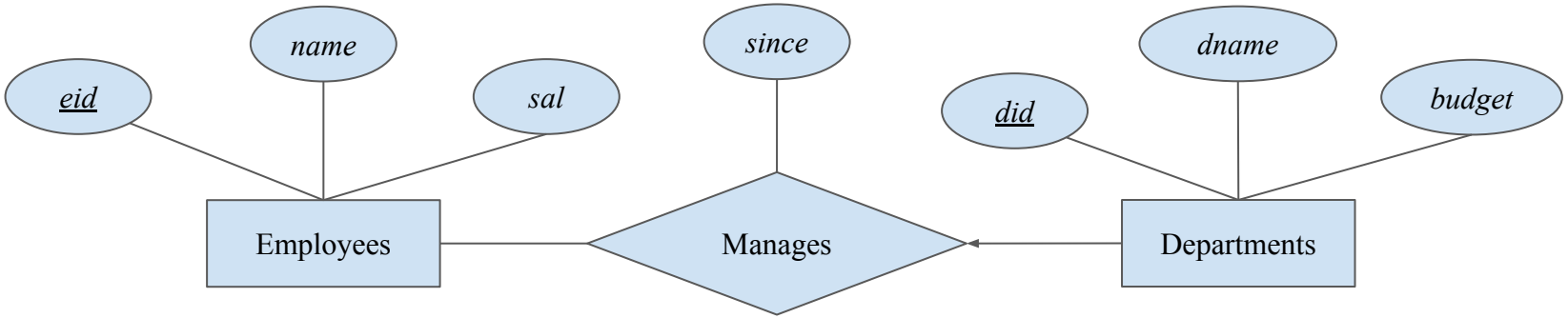
Relationship Sets (without Constraints) to Tables



Relationship Sets (without Constraints) to Tables

```
CREATE TABLE Reports_to (
        supervisor_eid CHAR (11) ,
        subordinate_eid CHAR(11) ,
        PRIMARY KEY (supervisor_eid, subordinate_eid),
        FOREIGN KEY (supervisor_eid) REFERENCES Employees (eid),
        FOREIGN KEY (subordinate_eid) REFERENCES Employees(eid) );
```

Translating Relationship Sets with Key Constraints



- Each department has at most one manager
  - Is (eid, did) a primary key?

Translating Relationship Sets with Key Constraints

- Each department has at most one manager
- No two tuples can have the same **did** value but differ on the **eid** value.
- A consequence of this observation is that did is itself a key for Manages; indeed, the set (**eid, did**) is not a key (because it is not minimal).

Translating Relationship Sets with Key Constraints

```
CREATE TABLE Manages (eid CHAR(11),
        did INTEGER,
        since DATE,
        PRIMARY KEY (did),
        FOREIGN KEY (eid) REFERENCES Employees(eid),
        FOREIGN KEY (did) REFERENCES Departments(did));
```

Translating Relationship Sets with Key Constraints

Approach 2:

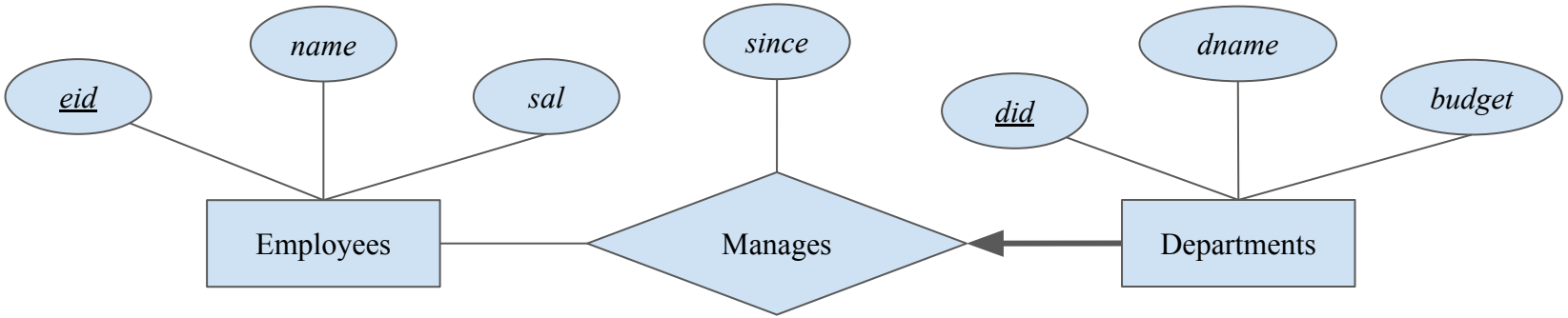
- The idea is to include the information about the relationship set in the table corresponding to the entity set with the key, taking advantage of the key constraint.
- This approach eliminates the need for a separate Manages relation, and queries asking for a department's manager can be answered without combining information from two relations.
- Drawback to this approach is that space could be wasted if several departments have no managers (recall at most one manager constraint). In this case the added fields would have to be filled with null values

Translating Relationship Sets with Key Constraints

```
CREATE TABLE Manager ( did INTEGER,
                       dname CHAR(20),
                       budget INTEGER,
                       eid CHAR(11),
                       since DATE,
                       PRIMARY KEY (did),
                       FOREIGN KEY (eid) REFERENCES Employees(eid));
```

- Note eid can take on null values (as it is not a primary key)

Participation Constraints



- Total participation: Each department must participate in Manages and there must be one manager

Participation Constraints

```
CREATE TABLE Dept_Mgr ( did INTEGER,
                        dname CHAR(20),
                        budget INTEGER,
                        eid CHAR(11) NOT NULL,
                        since DATE,
                        PRIMARY KEY (did),
                        FOREIGN KEY (eid) REFERENCES Employees(eid)
                              ON DELETE NO ACTION);
```

- eid CHAR(11) NOT NULL: It is not possible to have a department with no manager
- ON DELETE NO ACTION: An employee cannot be deleted (or fired) if he is also a manager. The new manager has to be hired (thus old employee will have no pointer to Dept_Mgr) and then only can be deleted.

Participation Constraints

- It captures the participation constraint that every department must have a manager: Because eid cannot take on null values, each tuple of Dept_Mgr identifies a tuple in Employees (who is the manager).
- The NO ACTION specification, which is the default and need not be explicitly specified, ensures that an Employees tuple cannot be deleted while it is pointed to by a Dept_Mgr tuple. If we wish to delete such an Employees tuple, we must first change the Dept_Mgr tuple to have a new employee as manager.
- We could have specified CASCADE instead of NO ACTION, but deleting all information about a department just because its manager has been fired seems a bit extreme!

Participation Constraints