Lecture 4

Relational Model

Dr. Vandana Kushwaha

Department of Computer Science
Institute of Science, BHU, Varanasi

Introduction

- The **Relational Data Model** was first introduced by **Ted Codd** of **IBM Research** in **1970** in a classic paper (Codd 1970).
- **Relational model** attracted immediate attention due to its **simplicity** and **mathematical foundation.**
- The **model** uses the concept of a ***mathematical relation***—which looks somewhat like a **table** of values—as its **basic building block**.
- The **first commercial implementations** of the **Relational model** became available in the **early 1980s**, such as the **SQL/DS system by IBM** and the **Oracle DBMS.**
- Current **popular relational DBMSs** (RDBMSs) include **DB2** and **Informix Dynamic Server** (from IBM), **Oracle RDBMS**(from Oracle), **Sybase** DBMS (from Sybase) and **SQL Server** and **Access** (from Microsoft).
- In addition, several **open source systems**, such as **MySQL** and **PostgreSQL,** are available.

Relational Model Concepts

- The **Relational model** represents the **database** as a **collection of *relations.***
- Informally, each **relation** resembles a **table** of values.
- Each **row** in the **table represents** a **collection** of **related data values**.
- A **row** represents a fact that typically corresponds to a **real-world entity** or **relationship.**
- The **table name** and **column names** are used to help to interpret the meaning of the **values** in each **row.**
- For **example**, one **table** is named **STUDENT** because **each row** represents facts about a **particular student entity.**
- The **column names**—*Name, Student_number, Class, and Major*—specify how to interpret the **data values** in **each row**, based on the column each value is in.
- All **values** in a **column** are of the **same data type.**

Relational Model Concepts

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

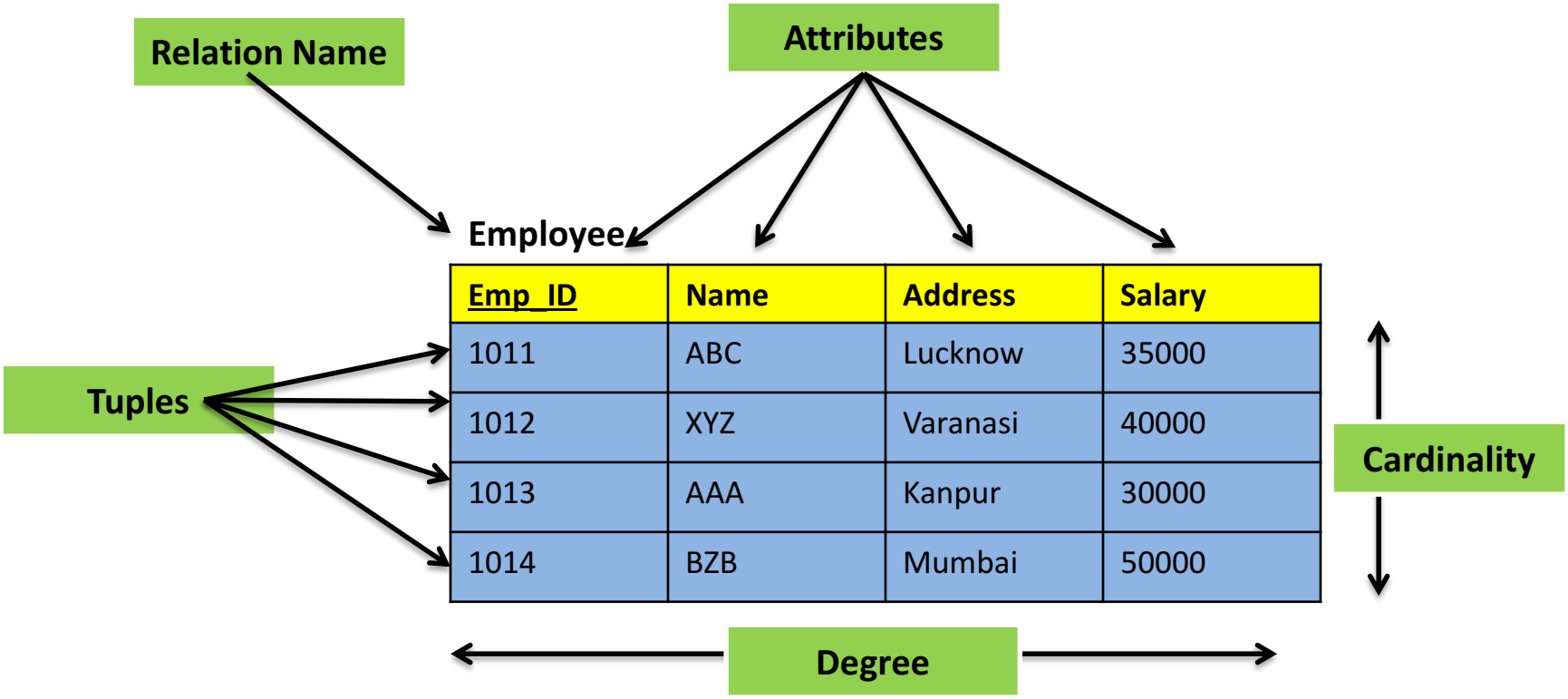
Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

Relation Schema

- A **relation schema *R***, denoted by ***R*(*A*1, *A*2, ...,A*n*)**, is made up of a **relation name *R*** and a **list of attributes**, ***A*1, *A*2, ..., *An*.**
- For each **attribute Ai**, there is a **set of permitted values**, called the **domain D** of that attribute in the relation schema ***R***.
- ***D*** is called the **domain** of ***Ai*** and is denoted by **dom**(**A**i*).*
- A **relation schema** is used to ***describe* a relation**; ***R*** is called the **name** of this **relation.**
- The **Degree** of a **relation** is the **number of attributes *n*** of its relation schema.
- The **Cardinality** of a **relation** is the **number of tuples(records/rows)** in that relation.
- **Example:** A **relation** of **degree seven**, which stores information about university students, would contain **seven attributes** describing each student as follows:

STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)

A Relation


Relation Name
Attributes
Employee
Emp_ID
Name
Address
Salary
1011
ABC
Lucknow
35000
1012
XYZ
Varanasi
40000
1013
AAA
Kanpur
30000
1014
BZB
Mumbai
50000
Tuples
Cardinality
Degree

Properties of a Relation

1. **No duplicate tuples**
   - A **relation** can not contain two or more tuples which have the same values for all the **attributes.**
   - Thus in any **relation every row(tuple)** should be **unique**.
2. **Tuples are unordered**
   - The **order of rows(tuples)** in a **relation** is **immaterial.**
3. **Attribute values are atomic**
   - **Each tuple** contains **exactly one value** for **each attribute**.

Relational Model Constraints

- There are various types of **Constraints** that can be specified on a relational database schema:

1. **Domain Constraints**
2. **Key Constraints**
3. **Integrity Constraints**
    a. **Entity Integrity Constraints**
    b. **Referential Integrity Constraints**

1. Domain Constraints

- **Domain constraints** specify that within **each tuple**, the **value of each attribute A** in a **Relation** must be taken from the **same domain**.
- A **domain definition** usually consists of the following **components:**
  - ***Data type***
  - ***Size or length***
  - ***Allowable values or allowable range***
- **Example:**
- ***Names:*** The set of character strings that represent names of persons.
- ***Employee_ages*****:** Possible ages of employees in a company; each must be an integer value between 15 and 80.
- ***Academic_department_names:*** The set of academic department names in a university, such as Computer Science, Economics, and Physics.
- ***Mobile_numbers:*** The set of ten-digit numbers.

2. Key Constraints

- In the formal **Relational model**, a ***relation*** is defined as a ***set of distinct tuples***.
- This means that **no two tuples** can have the **same combination of values** for ***all*** **their attributes.**
- The **subsets of attributes** of a **relation schema *R*** with the property that **no two tuples** in any **relation state *r* of *R*** should have the **same combination of values** for **these attributes.**
- Suppose that we denote **one** such **subset of attributes** by **SK;** then for **any two *distinct* tuples *t1*** and ***t*2** in a **relation state *r* of *R***, we have the **constraint** that:

$$t_1[\mathrm{SK}] \neq t_2[\mathrm{SK}]$$

- Any **such set of attributes SK** is called a **superkey** of the **relation schema *R*.**
- A **superkey SK** specifies a ***uniqueness constraint*** that no two distinct tuples in any **state *r* of *R*** can have the same value for **SK.**
- **Every relation** has **at least one default superkey—the set of all its attributes**.

2. Key Constraints

- A **key *K*** of a **relation schema *R*** is a **superkey** of ***R*** with the additional **property** that **removing any attribute *A* from *K*** leaves a set of attributes ***K*** that is **not a superkey of *R* any more.**
- Hence, a **key** satisfies **two properties:**

1. **Two distinct tuples** in any state of the relation cannot have identical values for (all) the attributes in the key.
   - This first property also applies to a **superkey.**
2. It is a ***minimal superkey***—that is, a **superkey** from which we cannot remove any attributes and still have the uniqueness constraint in **condition 1** hold.
   - This property is not required by a **superkey.**

- A **key** is also a **superkey** but not **vice versa.**

2. Key Constraints

- Consider the **STUDENT relation** :
  - ***STUDENT(****Name, Ssn, Home_phone, Address, Age, Salary****)***
- The **attribute set {Ssn}** is a **key** of **STUDENT** because no two student tuples can have the same value for Ssn.
- Any set of attributes that includes Ssn—for example, **{Ssn, Name, Age}—**is a **superkey.**
- However, the **superkey {Ssn, Name, Age}** is **not a key** of **STUDENT** because removing Name or Age or both from the set still leaves us with a **superkey.**
- In general, any **superkey** formed from a **single attribute is also a key**.
- A **key with multiple attributes** must require ***all*** **its attributes** together to have the **uniqueness property.**

2. Key Constraints

- In general, a **relation schema** may have **more than one key**.
- In this case, each of the **keys** is called a **candidate key**.
- For example, the **CAR relation** has **two candidate keys**: License_number and Engine_serial_number.
  - ***CAR(License_number, Engine_serial_number, Make, Model ,Year)***
- It is common to **designate one of the candidate keys** as the **primary key** of the **relation.**
- This is the **candidate key** whose values are used to ***identify*** **tuples** in the relation.
- We use the convention that the attributes that form the **primary key** of a **relation schema** are **underlined.**

Constraints on NULL Values

- Another **constraint on attributes** specifies whether **NULL values** are **permitted** or are **not permitted.**
- For **example**, if every **STUDENT tuple** must have a **valid**, **non-NULL** value for the **Name attribute.**
- Then **Name of STUDENT** is **constrained** to be **NOT NULL.**

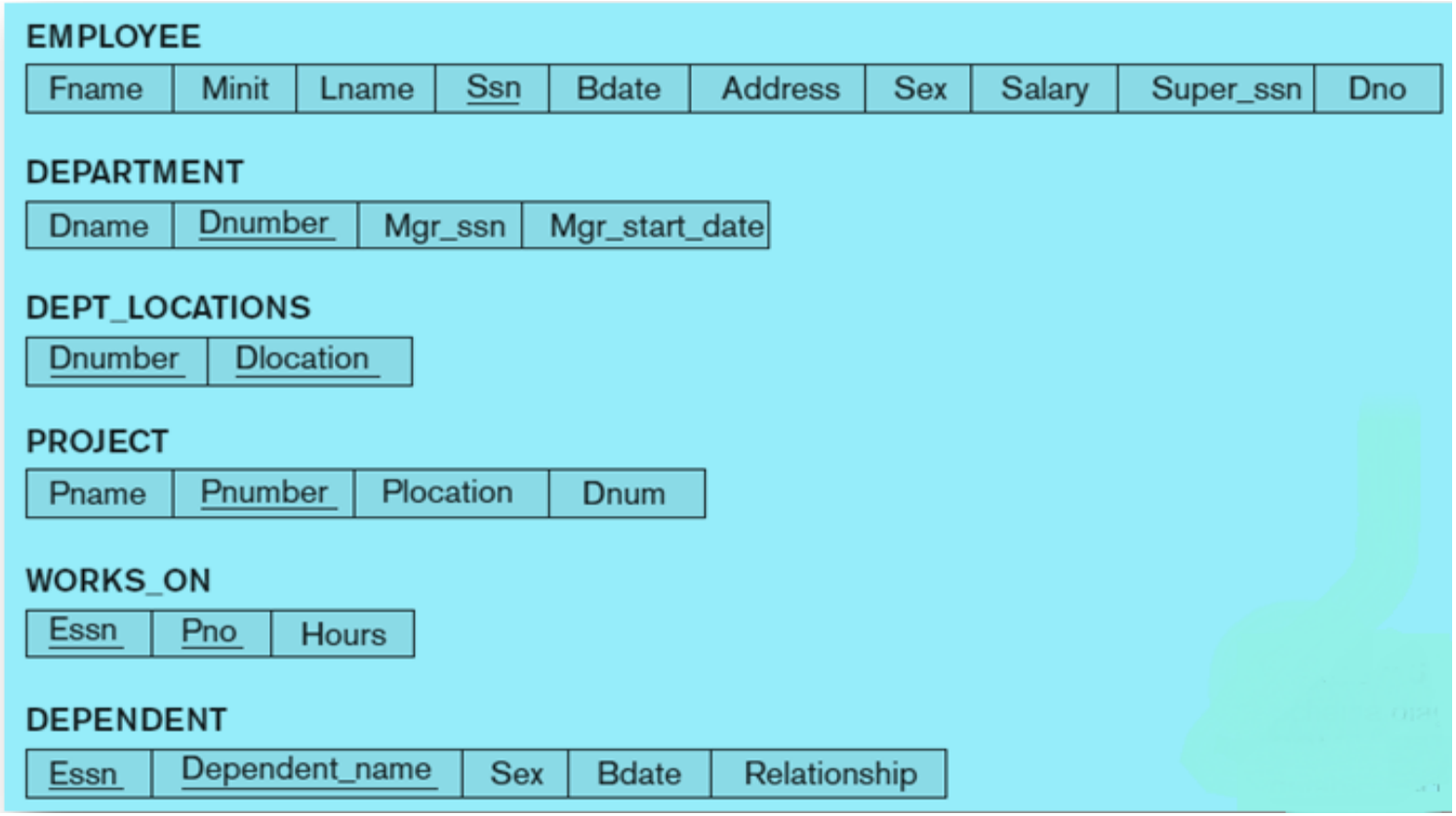
Relational Database Schemas

- A **relational database schema *S*** is a set of **relation schemas S = {*R*1, *R*2, ..., *Rm*}** and a set of **integrity constraints IC.**
- Example: A **relational database schema**

COMPANY = *{EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}*

- A **relational database state DB** of ***S*** is a **set of relation states *DB = {r1, r2, ..., rm}*** such that each ***ri*** is a **state of *Ri*** and such that the ***ri* relation** states satisfy the **integrity constraints** specified in **IC.**
- A **relational database** implicitly include both its **schema** and its **current state.**
- A **database state** that does not obey all the integrity constraints is called an **invalid state**.
- A **state** that **satisfies all the constraints** in the defined set of **integrity constraints IC** is called a **valid state.**

Schema diagram



- Each **relational DBMS** must have a **Data definition language (DDL)** for **defining** a **relational database schema.**
- Current relational DBMSs are mostly using **SQL(Structured Query Language)** for this purpose.

Database state

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son

3. Integrity Constraints

- **Entity integrity constraint**
- The **entity integrity constraint** states that **no primary key value** can be **NULL.**
- This is because the **primary key value** is used to identify individual **tuples** in a **relation**.
- Having **NULL values** for the **primary key** implies that we cannot identify some tuples.
- For **example**, if two or more tuples had **NULL** for their **primary keys**, we may not be able to **distinguish** them if we try to reference them from other relations.
- **Key constraints** and **entity integrity** constraints are **specified on individual relations.**

3. Integrity Constraints

- **Referential integrity constraint**
- The **referential integrity constraint** is specified between **two relations** and is used to maintain the **consistency** among tuples in the **two relations.**
- The **referential integrity constraint** states that a **tuple** in one **relation** that **refers** to another relation **must refer** to an ***existing tuple*** in that **relation**.
- For **example,**the attribute **Dno** of **EMPLOYEE** gives the department number for which each employee works.
- Hence, its value in every **EMPLOYEE tuple** must match the **Dnumber value** of some tuple in the **DEPARTMENT relation**.
- **Foreign key** is used to specify the **referential integrity constraint** between the **two relation schemas R1** and **R2.**

3. Integrity Constraints

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

Foreign key

- A **set of attributes FK** in **relation schema *R*1** is a **foreign key** of ***R*1** that **references relation *R*2** if it satisfies the following two **rules:**

  Rule 1:

  - **The attributes in FK** have the **same domain(s)** as the **primary key attributes PK** of ***R*2**; the **attributes FK** are said to **reference** or **refer to** the relation ***R*2.**

  Rule 2:

  - A value of **FK** in a **tuple *t*1** of the current state ***r*1(*R*1)** either occurs as a value of **PK** for some **tuple *t*2** in the current state ***r*2(*R*2)** *or is* ***NULL*.**
  - In the former case, we have ***t*1[FK] = *t*2[PK],** and we say that the **tuple *t*1 references** or **refers to** the **tuple *t*2.**

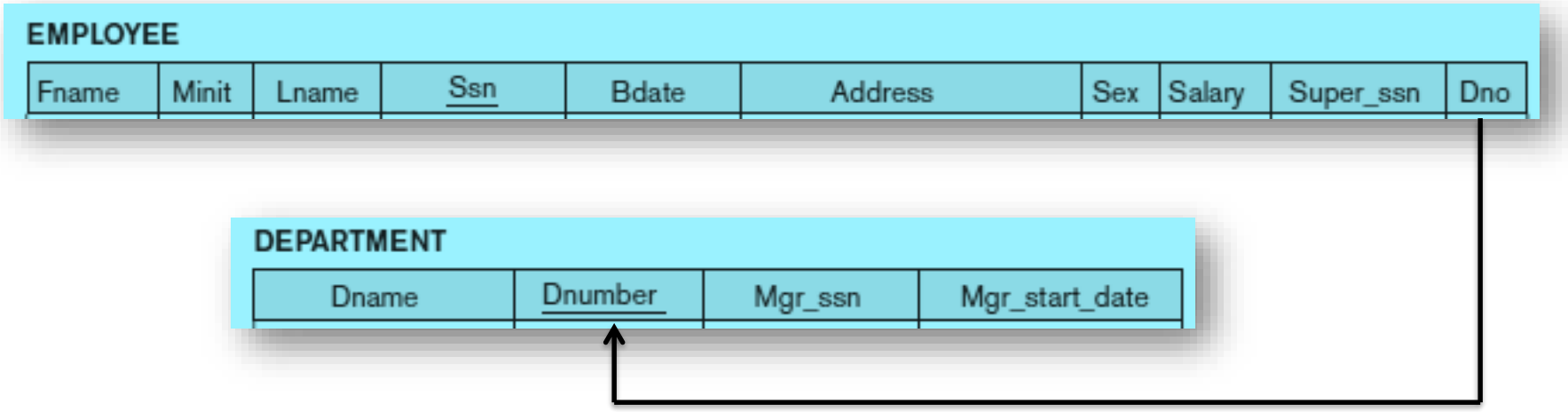
- In this definition, ***R*1** is called the **referencing relation** and ***R*2** is the **referenced relation**.

Foreign key

- In a **Database** of **many relations**, there are usually **many referential integrity constraints.**
- **Referential integrity constraints** typically **arise** from the ***relationships among the entities*** .
- **Example:** In **COMPANY database**, the **EMPLOYEE relation,** the **attribute Dno** refers to the **department** for which an **employee works**; hence, we designate **Dno** to be a **foreign key** of **EMPLOYEE** referencing the **DEPARTMENT relation**.
- This means that a value of **Dno** in any **tuple *t*1** of the **EMPLOYEE relation** must **match** a value of the **primary key of DEPARTMENT**—the **Dnumber attribute**—in some **tuple *t*2** of the **DEPARTMENT relation**, or
- The value of **Dno *can be NULL*** if the employee does not belong to a department or will be assigned to a department later.

Foreign key

- We can ***diagrammatically display referential integrity constraints*** by drawing a **directed arc** from **each foreign key to the relation** it references.
- The **arrowhead** may **point to the primary key** of the **referenced relation**.



Self Referential Integrity

- A **foreign key can *refer to its own relation*.**
- Such **referential integrity constraint** is referred as **Self Referential Integrity constraint.**
- For **example**, the attribute **Super_ssn** in **EMPLOYEE** refers to the **supervisor** of an **employee**; this is another employee, represented by a tuple in the **EMPLOYEE** relation.
- Hence, **Super_ssn** is a **foreign key** that **references** the **EMPLOYEE relation itself**.
- In **EMPLOYEE** relation the tuple for employee 'John Smith' references the tuple for employee 'Franklin Wong,' indicating that 'Franklin Wong' is the supervisor of 'John Smith.'

Integrity Constraints

- All **integrity constraints** should be specified on the **relational database schema** (i.e., defined as part of its definition).
- Hence, the **DDL** includes **provisions for specifying** the various types of **constraints** so that the DBMS can automatically enforce them.
- Most **relational DBMSs** support **key, entity integrity,** and **referential integrity constraints.**
- These **constraints** are **specified** as a **part of data definition** in the **DDL.**

Referential integrity using FOREIGN KEY

- Referential integrity is specified via the **FOREIGN KEY** clause at **table level**, as shown below:

```
CREATE TABLE EMPLOYEE
(.....,
Ssn CHAR(9) NOT NULL,
Salary NUMBER(10,2),
Super_ssn CHAR(9),
Dno NUMBER NOT NULL,
PRIMARY KEY (Ssn),
FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

Referential integrity using FOREIGN KEY

- A **Referential integrity constraint** can be **violated** when tuples are **inserted** or **deleted,** or when a foreign key or primary key attribute value is modified.
- The **default action** that SQL takes for an **integrity violation** is to **reject** the **update operation** that will cause a **violation.**
- However, the schema designer can specify an alternative action to be taken by attaching a **referential triggered action** clause to any **foreign key constraint.**
- The options include **SET NULL**, **CASCADE**, and **SET DEFAULT.**
- An option must be qualified with either **ON DELETE** or **ON UPDATE**.
- **Example:**
- The database designer chooses **ON DELETE SET NULL** and **ON UPDATE CASCADE** for the **foreign key Super_ssn** of **EMPLOYEE.**

Referential integrity using FOREIGN KEY

- What is a foreign key with **Cascade DELETE** in Oracle?
- A foreign key with **cascade delete** means that ***if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted.***
- A **foreign key** with a **cascade/set null** can be defined in either a **CREATE TABLE** statement or an **ALTER TABLE** statement.
- **Example:**

  CREATE TABLE EMPLOYEE

  (…..,

  PRIMARY KEY (Ssn),

  FOREIGN KEY (Super_ssn) **REFERENCES** EMPLOYEE(Ssn) **ON DELETE SET NULL**)

Referential integrity using FOREIGN KEY

- This means that if the tuple for a ***supervisor employee*** is ***deleted***, the value of **Super_ssn** is automatically set to **NULL** for all employee tuples that were referencing the deleted employee tuple.
- On the other hand, if the **Ssn** value for a **supervisor employee** is *updated* (say, because it was entered incorrectly), the new value is ***cascaded*** to Super_ssn for all employee tuples referencing the updated employee tuple.

Example:

CREATE TABLE EMPLOYEE

(. . . ,

Dno INT **NOT NULL DEFAULT** 1,**PRIMARY KEY** (Ssn),

FOREIGN KEY (Super_ssn) **REFERENCES** EMPLOYEE(Ssn) **ON DELETE** SET NULL **ON UPDATE** CASCADE,

FOREIGN KEY(Dno) **REFERENCES** DEPARTMENT(Dnumber) **ON DELETE** SET DEFAULT **ON UPDATE** CASCADE);

Referential integrity using FOREIGN KEY

ON DELETE	**SET NULL**	*Sets the rows in the child table to NULL when the corresponding row in the parent table is deleted.*
	SET DEFAULT	*Sets the rows in the child table to their default values if the corresponding rows in the parent table are deleted.*
	CASCADE	*Delete the rows from the child table automatically, when the rows from the parent table are deleted.*
ON UPDATE	**SET NULL**	*Sets the rows in the child table to NULL when the corresponding row in the parent table is updated.*
	SET DEFAULT	*Sets the rows in the child table to their default values if the corresponding rows in the parent table are updated.*
	CASCADE	*Update the rows from the child table automatically, when the rows from the parent table are updated.*