RELATIONAL DATABASE DESIGN

Chapters 5 & 9.1

- The Relational Data Model, Relational Database Constraints, and ER-Relational Mapping
- Integrity Violations Caused by Update Operations and How to Cope with the Violations

Contents

- Background
- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Relational Database Design by Entity-Relation Mapping
- Update Operations and Dealing with Constraint Violations (if time)

Background

- We now explore relational databases.
- The *first* relational data model was introduced by E. Codd in 1970 in the following paper:
  - "A Relational Model for Large Shared Data Banks," *Communications of the ACM*, June 1970.
  - Why attracted? Its *simplicity* and *mathematical foundation*.
- This paper caused a major revolution in the field of database management.
  - Awarded him the coveted ACM Turing Award (Nobel prize in Computer Science).

Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

Background (Cont'd)

- The relational model:

  1) Uses the concept of a *mathematical relation* (like a table of values) as its basic building block, and

  2) Has its theoretical basis in *set theory* (집합 이론) and *first-order predicate logic* (1차 술어 논리).

- We discuss the basic characteristics of the relational model and its constraints.

Background (Cont'd)

- The first commercial implementations of the relational model appeared in the early 1980s:
  - The SQL/DS system on the MVS operating system by IBM
  - The Oracle DBMS: used in our lab practice

- Since then, the model has been implemented by many:
  - Commercial, popular relational DBMSs (RDBMSs)
    - DB2 (from IBM), Oracle (from Oracle), Sybase DBMS (now from SAP), and SQL Server and Microsoft Access (from Microsoft)
  - Open-source, popular RDBMSs
    - MySQL (now from Oracle), PostgreSQL, SQLite (for mobile), MariaDB, …

RELATIONAL MODEL CONCEPTS (관계형 모델 개념)

Chapter 5.1

Relational Model Concepts: "Informal" Definitions

- *Relation* (릴레이션): a mathematical concept based on the ideas of "sets".
  - In the relational model, the database => a collection of relations.
- A relation looks like a **table** (informal) of values, or a *flat* file of records (because of its flat structure).
  - Recall the `STUDENT` table in the earlier class.

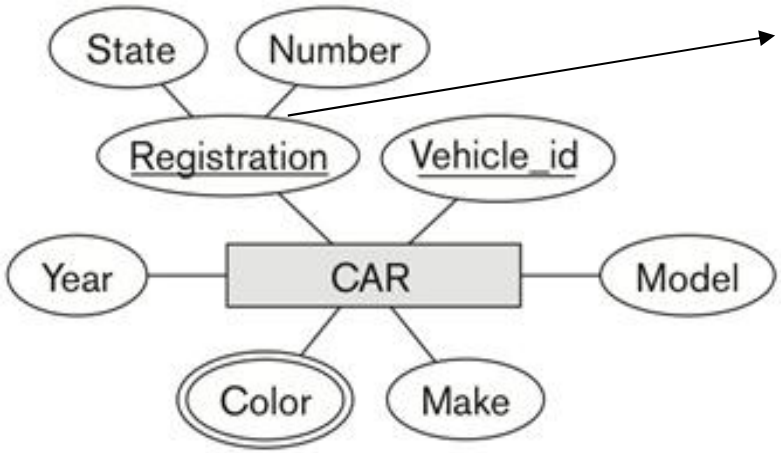
STUDENT

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

- A relation typically contains a set of **rows** (informal).
  - The data elements in each **row** represents certain facts that typically corresponds to a real-world **entity** or **relationship**.
  - In the formal model, a row is called a ***tuple*** (튜플).
- Each column has a column header (informal), delivering the meaning of the data items in that column.
  - In the formal model, the header is called an ***attribute*** (애트리뷰트).

Relational Model Concepts: "Informal" Definitions (Cont'd)

- ***Key*** of a relation (most important):
  - Each row has a value of a data "element", called a **key**, that uniquely identifies that row in the table.
  - What's the key attribute in the `STUDENT` table?
  - Sometimes, a row id or a sequential number can be used as a key, called a ***surrogate*** (or ***artificial***) ***key***, simply to identify a row in a table.



Can be expressed in the relational model as '`Registration_no`' as a surrogate key

Relational Model Concepts: "Formal" Definitions – ***Schema***

- The ***schema*** of a relation:
  - Denoted by $R(A_1, A_2, \ldots, A_n)$
    - R: a relation name,
    - $A_1, A_2, \ldots, A_n$: a list of attributes
  - Used to describe a relation
  - The ***degree*** (or ***arity***) of the relation R: the number of attributes of R.
- Example:
  - `CUSTOMER (Cust-id, Cust-name, Addr, Phone#)`
    - What's the relation name?
    - Defined over the four attributes: what?
- Each attribute has a ***domain*** (to be discussed soon) of a set of valid values.
  - E.g., the domain of `Cust-id`: 6 digit numbers.
    - C.f. the domain of an unsigned 32-bit integer: 0 ~ (2^{32}-1)

Relational Model Concepts: "Formal" Definitions – *Tuple*

- What's a tuple?
  - An ordered set of values: enclosed in angled brackets '< … >'
- Each value of the tuple is derived from an appropriate domain.
- A tuple (row) in the `CUSTOMER` relation (table) looks like:
  - `<632895, "홍길동", "대구 북구 대학로 80 IT-5 41566", "053-950-6372">`
  - Called a *4*-tuple: Why?
- A relation (table) is a set of such tuples (rows).

Relational Model Concepts: "Formal" Definitions – ***Domain***

- A domain (say, **D**): a set of atomic (indivisible) values.
  - Has a logical definition (or, name):
    - E.g., "`Korea_cell_phone_numbers`" (typically) indicates the set of 11 digit phone numbers, starting 01 and being ***valid*** in Korea.
  - Has a data type or a data format defined for it.
    - Ex1) "`Korea_cell_phone_numbers`" follows the format: `(01X)-dddd-dddd`
    - Ex2) Dates have various formats: year, month, date formatted as:
      - `yyyy-mm-dd`, or `dd/mm/yyyy`, and so forth.
- The attribute (column) name plays a *role* for a domain in a relation (table):
  - Used to interpret the meaning of the date elements corresponding to that attribute
    - E.g., The domain Date can be used to define with different meanings two attributes: "`Invoice-date`" or "`Payment-date`"

Relational Model Concepts: "Formal" Definitions – ***State***

- The ***relation state***: a subset of the Cartesian product[1] of the domains of its attributes.
  - Each domain contains the set of all possible values that can be taken by the attribute.
  - E.g. the attribute, "`Cust-name`": defined over the domain of character strings of maximum length, say 30.
    - `dom(Cust-name): varchar(30)`
  - The role played by these strings in the `CUSTOMER` relation is that of the name of a customer.

[1]all possible combinations of values

Relational Model Concepts: "Formal" Definitions – Summary

- Formally speaking,
  - Given a **relation** $R(A_1, A_2, \ldots, A_n)$,
    - $r(R) \subseteq \text{dom}(A_1)$ x $dom(A_2)$ x .... x $\text{dom}(A_n)$
  - $R(A_1, A_2, \ldots, A_n)$: the **schema** of the relation
    - R: the **name** of the relation
    - $A_1, A_2, \ldots, A_n$: the **attributes** of the relation
  - $r(R)$: a specific **state** (or "value" or "population") of the relation R
    - A set of **tuples** (rows) in R.
    - $r(R) = \{t_1, t_2, \ldots, t_m\}$, where t_i is an n-tuple.
    - $t_i = <v_1, v_2, \ldots, v_n>$, where each v_j is an element of $\text{dom}(A_j)$.

Relational Model Concepts: Formal Definitions – Example

- Let a **relation** $R(A_1, A_2)$ to be a relation schema:
  - Let $\text{dom}(A_1) = \{0, 1\}$
  - Let $\text{dom}(A_2) = \{\text{a, b, c}\}$
- Then, $\text{dom}(A_1)$ x $\text{dom}(A_2)$: all possible combinations, like

$$\{<0, \text{a}>, <0, \text{b}>, <0, \text{c}>, <1, \text{a}>, <1, \text{b}>, <1, \text{c}>\}.$$

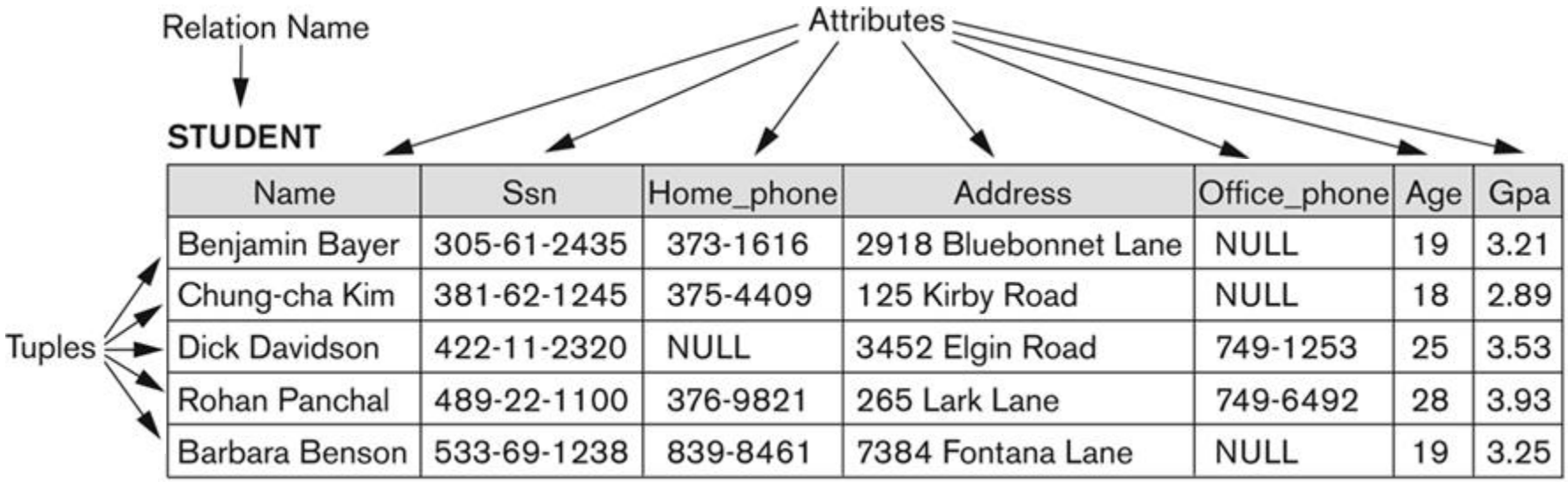
- The relation state, $r(R) \subseteq \text{dom}(A_1)$ x $\text{dom}(A_2)$
- For example, $r(R) = \{<0, \text{a}>, <0, \text{b}>, <1, \text{a}>, <1, \text{b}>\}$.
  - Note that this is just one possible state (or *extension*), called r, of the relation R, defined over A_1 and A_2.
  - In the example, r has four 2-tuples. What?

Relational Model Concepts (Cont'd)

- Definition Summary

Informal Terms	Formal Terms
Table	***Relation***
Column Header	***Attribute***
All possible values in a column	***Domain***
Row	***Tuple***
Table Definition	***Schema (intension) of a Relation***
Populated (Loaded) Table	***State (extension) of the Relation***

Example: A Relation STUDENT



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

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

"Characteristics" of Relations

- Ordering of tuples in a relation
  - The tuples are *not considered to be ordered*, even though they appear to be in the tabular form: like a set
  - Tuple ordering is *not part of a relation definition*, as a relation attempts to represent facts at a logical/abstract level.
  - E.g., the relation `STUDENT` with a different order of tuples

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

=> Considered to be **identical** as the one before

Characteristics of Relations (Cont'd)

- Ordering of attributes in a relation schema R (and of values within each tuple)
  - The attributes in $R(A_1, A_2, \ldots, A_n)$
  - The values in a tuple t_i, or $<v_1, v_2, \ldots, v_n>$
    - *Considered to be ordered.*
  - By the definition of a relation, an n-tuple is an ordered list of n values.
  - However, there exists *no* order in a "more general" alternative definition, including the name and the value for each of the attributes as a pair.

$t = <$ (Name, Dick Davidson),(Ssn, 422-11-2320),(Home_phone, NULL),(Address, 3452 Elgin Road), (Office_phone, (817)749-1253),(Age, 25),(Gpa, 3.53)>

$t = <$ (Address, 3452 Elgin Road),(Name, Dick Davidson),(Ssn, 422-11-2320),(Age, 25), (Office_phone, (817)749-1253),(Gpa, 3.53),(Home_phone, NULL)>

=> *Identical*, when the order of attributes and values is not part of relation definition.

=> The tuples are "self-describing". Why?

Characteristics of Relations (Cont'd)

- Values in a tuple:
  - All values are considered ***atomic***.
    - Composite and multi-valued attributes are *not allowed*.
      - Called the flat relational model, based on **the first normal form assumption**, which will be discussed later in the course
      - Composite attributes => *simple attributes*
      - Multi-valued => separate *relations*
  - Each value in a tuple *must* be from the domain of the attribute for that column.
  - If tuple $t = <v_1, v_2, \ldots, v_n>$ is a tuple (row) in the relation state r of $R(A_1, A_2, \ldots, A_n)$, then
    - Each v_i must be a value from $\mathrm{dom}(A_i)$.

Characteristics of Relations (Cont'd)

- `NULL` values in certain tuples
  - Used to represent values that are:
    - ***Unknown*** (e.g., don't know)
    - ***Not available*** (e.g., don't have a home phone number) or
    - ***Inapplicable*** (*e.g.,* female vs. male for body check).

Characteristics of Relations (Cont'd)

- Notation:
  - $R.A$: an attribute name A of a relation R (e.g., `STUDENT.Name`)
    - However, all attribute names in a particular relation must be distinct.
  - We refer to **component values** of a tuple t by:

$$t[A_i] \text{ or } t.A_i$$

  - This indicates the value v_i of attribute A_i for tuple t.
  - Similarly, $t[A_u, A_v, \ldots, A_w]$ (or, $t.(A_u, A_v, \ldots, A_w)$): the subtuple of t containing the values of attributes $A_u, A_v, \ldots, A_w$, respectively in t.
    - Consider the first tuple t in the `STUDENT` relation:

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

$t[\texttt{Name}] = ?$ $t[\texttt{Ssn}, \texttt{Gpa}, \texttt{Age}] = ?$

RELATIONAL MODEL CONSTRAINTS AND RELATIONAL DATABASE SCHEMAS (관계 모델 제약조건 및 관계 데이터베이스 스키마)

Chapter 5.2

Constraints (제약조건)

- Determine in the database:
  - Which values are permissible and
  - Which values are *not*.
- Has 3 main categories (or types):

  Category 1: *Inherent or implicit constraints*
  - Based on the data model itself.
    E.g., A relation model doesn't allow a list as a value for any attribute.

  Category 2: *Schema-based or explicit constraints*
  - Directly expressed in the data model schema
    E.g., min/max cardinality ratio in the ER model

  Category 3: *Application-based or semantic constraints* (or business rules)
  - Should be enforced by application programs; cannot be described by the model. E.g., An age of an employee must be less than 65.

Category 2: Relational Integrity Constraints

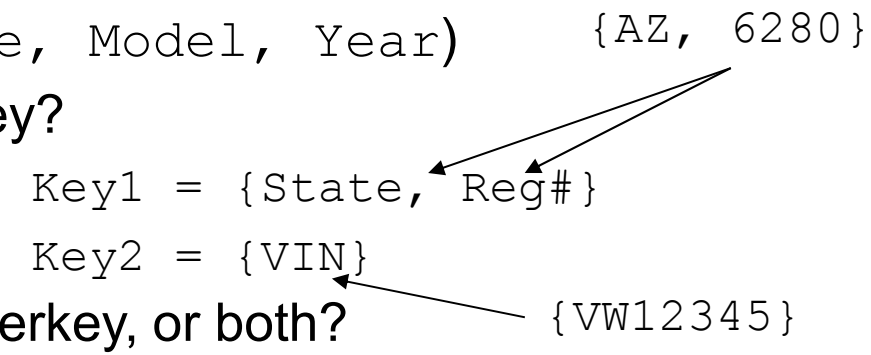
- Constraints are conditions that must hold on “all” valid relation states. (NO exception)
- 3 main types of constraints in the relational model (about Category 2 in the previous slide)
  1) **Key** constraints (키 제약조건)
     - Unique constraints (유일 제약조건)
  2) **Entity integrity** constraints (엔티티 무결성 제약조건)
  3) **Referential integrity** constraints (참조 무결성 제약조건)
- Besides, the **domain** constraint (영역 제약조건)
  - Every value in a tuple **must** be from the domain of its attribute.
  - The value, though, could be **null**, if allowed for that attribute.

C2-1) Key Constraints

- By definition, all tuples in a relation must be distinct.
  - Because a relation is a *set* of tuples, meaning no same element.
- ***Superkey*** of a relation R?
  - A subset of attributes, say SK, of R with the following condition:
  - No two tuples in any valid relation state $r(R)$ will have the same value for SK: called *the uniqueness property*
    - That is, for any distinct tuples t_1 and t_2 in $r(R)$, $t_1[SK] \neq t_2[SK]$.
    - This condition must hold in any "valid" state $r(R)$.
- ***Key*** of a relation R?
  - A "minimal" superkey
  - That is, a key is a superkey K, such that removal of any attribute from K results in a set of attributes that is not a superkey
- [Pair Question (PQ)] Is a superkey a key?

C2-1) Key Constraints (Cont'd)

- Example: Consider the `CAR` relation schema.
  - `CAR (State, Reg#, VIN*, Make, Model, Year)`
  - [Q] `CAR` has two keys. What are they?
  - Both are also superkeys of `CAR`.
  - [PQ] Is `{VIN, Make}` a key, a superkey, or both?
  - [PQ] Is `{State, Reg#}` a key, a superkey, or both?



- In general:
  - *Any key* is a superkey (but NOT vice versa).
  - Any set of attributes that includes a key is a superkey. Why?
  - A minimal superkey is also a key.
  - Also, a relation schema may have many keys, each of which is called a candidate key and one of them is "arbitrarily" chosen as the **primary key**.

* Vehicle Identification Number

C2-1) Key Constraints (Cont'd)

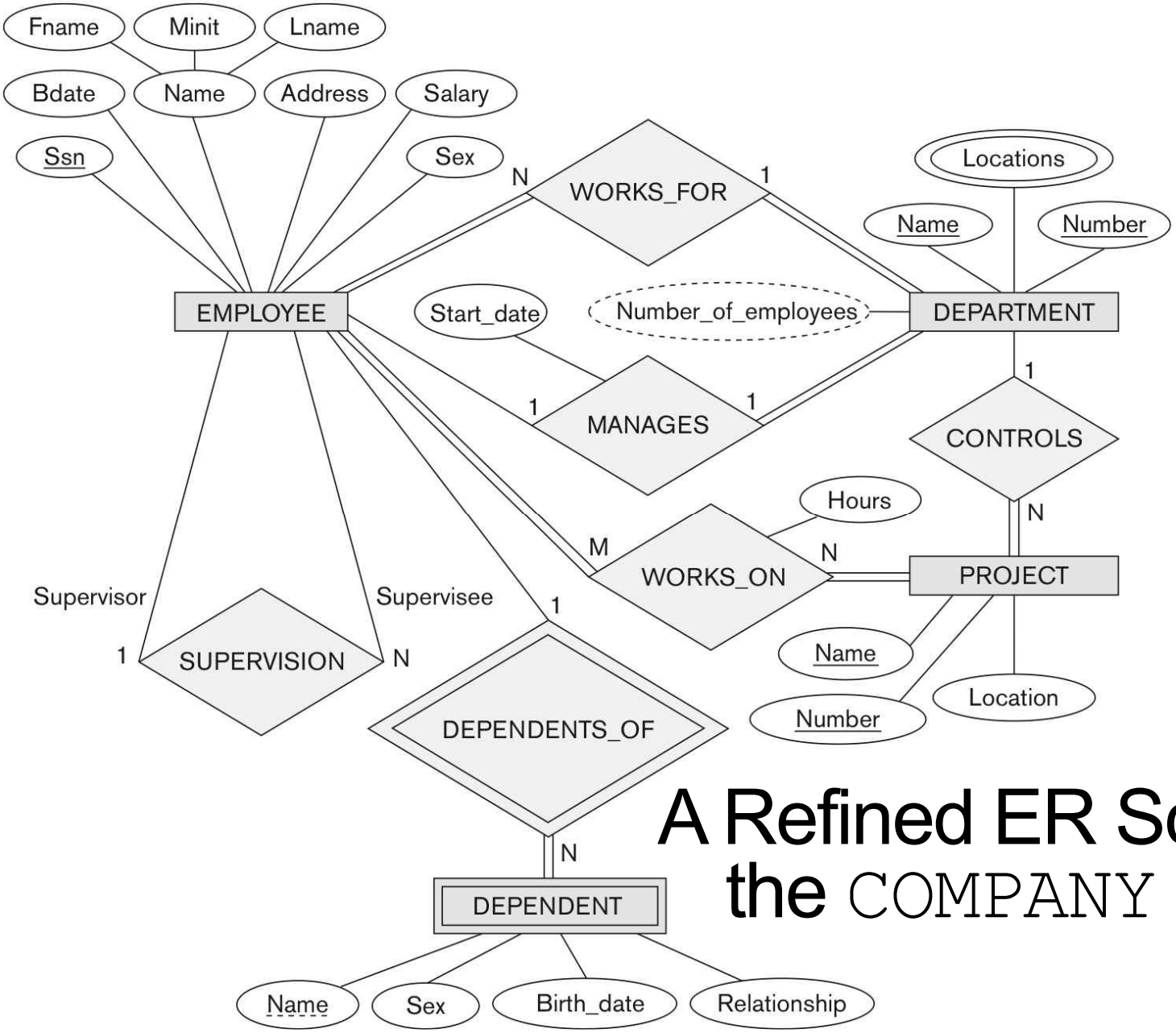
- Ex) `CAR(State, Reg#, VIN, Make, Model, Year)`
  - `VIN`: is a chosen for the primary key attribute. Q: Why?
    - Typically underlined in the relational schema.
- The other candidate keys are designated as unique keys and are not underlined. In SQL, …
- The primary key value is used to

  1) Uniquely identify each tuple in a relation.
  - Providing the tuple identity.

  2) Reference the tuple from another tuple; specified as a *foreign key* in the referencing relation
- General rule:
  - Choose as primary key the smallest of the candidate keys in terms of size– the number of attributes.
    - But not always …. sometimes it's up to a designer.

Relational Database Schema

- Now we turn to our discussion on a "relational database" containing many relations.
- What is a ***relational database schema*** then?
  - A set of S of relation schemas that belong to the same database.
    - S: the name of the whole database schema
    - $S = \{R_1, \ldots, R_n\}$ and a set of **Integrity Constraints** (**IC**s)
      - **IC**: constraints that must be preserved for consistency/integrity in database
        - Key, Not-Null, domain, and two other types (to be discussed later)
    - $R_1, R_2, \ldots, R_n$ are the names of the individual **relation schemas** within the database S.
- We'll recall the ER schema of `COMPANY` and then see its relational schema with six relation schemas.
  - `COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}`



A Refined ER Schema for the COMPANY Database

Schema Diagram for the COMPANY Relational Database Schema

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

- For the full version including the FKs refer to Slide 39.

Relational Database State

- Formally, a relational database state DB of S is defined by:
  - A set of relation states, $DB = \{r_1, r_2, ..., r_m\}$ such that
    - Each r_i is a state of R_i,
    - The r_i relation state satisfies the integrity constraints specified in IC.
- A relational database state is sometimes called a relational database *snapshot* or *instance*.

  (But we don't use the term of *instance*, as the *instance* is also applied to single tuples.)
- A database state that doesn't meet the constraints is said **invalid**.

Populated Database State (채워진 데이터베이스 상태)

- Each *relation* will have a number of tuples in its current relation state.
- The *relational database state* is a union of all the individual relation states.
- Whenever the database is changed (i.e., a tuple is modified, inserted, deleted), a new state is created.
- Basic operations for changing the relational database:
  - `INSERT`: a new tuple in a relation
  - `DELETE`: an existing tuple from a relation,
  - `MODIFY` or `UPDATE`: an attribute of an existing tuple

=> The DML statements in SQL support these operations.

One Possible DB State for COMPANY

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

One Possible DB State for COMPANY (Cont'd)

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
987654321	20	15.0
888665555	20	NULL

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
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

C2-2) Entity Integrity (개체 무결성)

- The primary key attributes, say PK, of each relation schema R in a database schema CANNOT have `NULL` values in any tuple of $r(R)$.
  - No primary key value can be `NULL`.
    - $t[PK] \neq$ null for *any* tuple t in $r(R)$.
    - If PK includes several attributes, `NULL` is NOT allowed in any of them.
  - Why? Because primary key values are used to identify the individual tuples. What if they are `NULL`?
  - Note, however, that for non-PK attributes, `NULL` values could be disallowed or not.

C2-3) Referential Integrity (참조 무결성)

- Specified (more than) two relations.
  - Note that so far the previous constraints have applied to a single relation.
- Used to maintain the "consistency" among tuples in the two relations.
  - E.g. `Dno` of `EMPLOEE`: the department number for which each employee works.
    - Its value in every `EMPLOYEE` tuple must match the `Dnumber` value of some tuple in the `DEPARTMENT` relation.
- Used to specify a **relationship** among tuples in two relations:
  - The referencing relation (참조하는 릴레이션): What's this?
  - The referenced relation (참조되는 릴레이션): What's this?

C2-3) Referential Integrity (Cont'd)

- Tuples in the **referencing relation** R_1 have attributes, say FK (called **foreign key attributes**), that reference the primary key attributes PK of the **referenced relation** R_2.
  - A tuple t_1 in R_1 is said to **reference** a tuple t_2 in R_2, if $t_1[FK] = t_2[PK]$
- A referential integrity constraint can be displayed in a relational database schema as a directed arc:
  - From $R_1.FK$ to $R_2.PK$ (e.g., R_1: `EMPLOYEE`, R_2: `DEPARTMENT`, FK: `Dno`, PK: `Dnumber`)

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

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22

C2-3) Foreign Key (Referential Integrity) Constraint

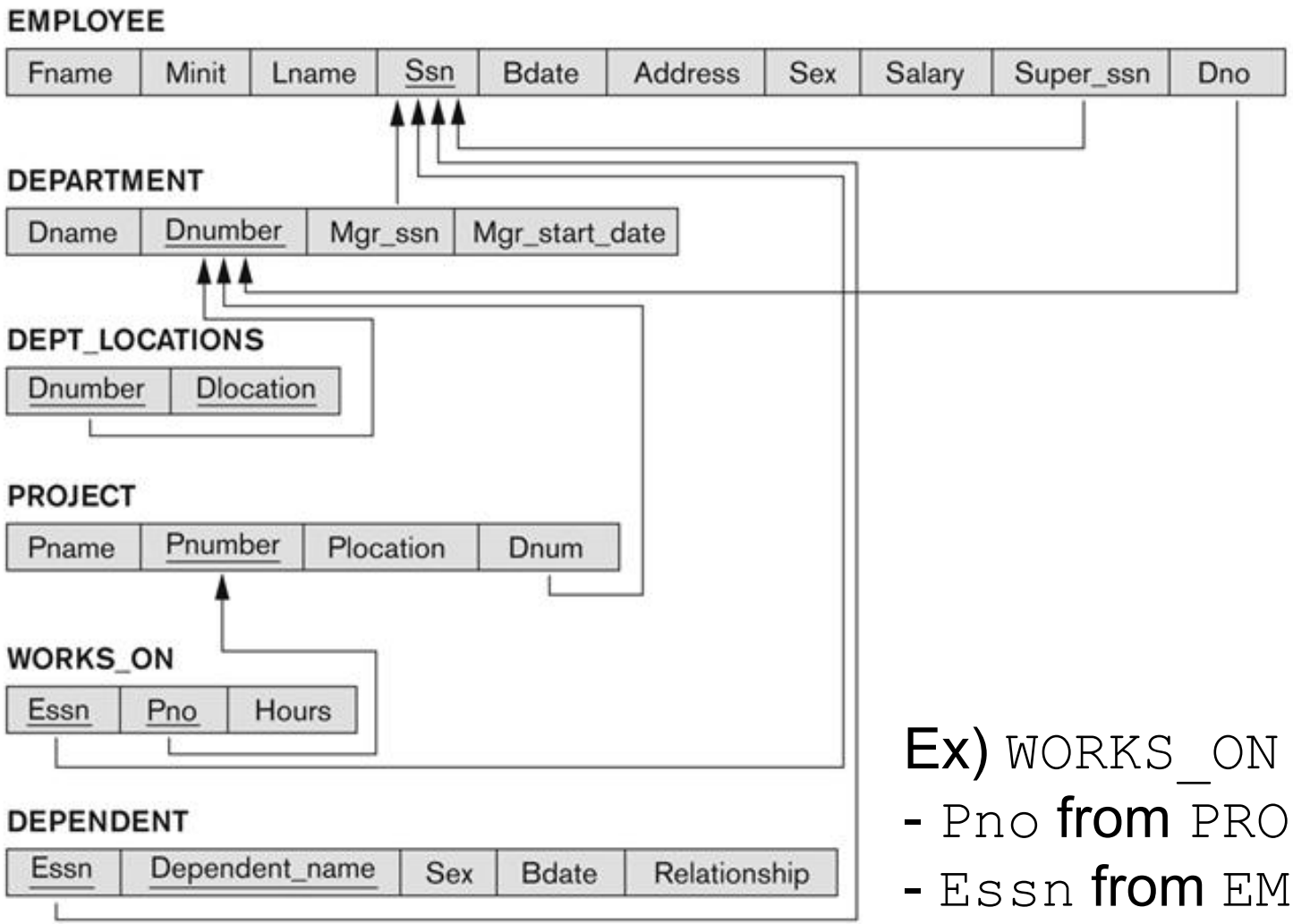
- The value in the foreign key attribute, say FK, of the referencing relation R_1 can be either:

  (1) $t_1[FK] = t_2[PK]$: that is, a value of an existing primary key value of a corresponding primary key, PK, in the referenced relation R_2, or

  (2) a *null* (not yet assigned?!).

  - If FK is null (as seen in case (2)), then the FK in R_1 should **NOT** be a part of its own primary key.
    - Why? Otherwise, it would violate entity integrity constraint.

Example: Referential Integrity Constraints for COMPANY



Ex) WORKS_ON
- Pno from PROJECT
- Essn from EMPLOYEE

C3) Other Types of Constraints

- Semantic Integrity Constraints:
  - Based application semantics
  - Cannot be expressed by the model
    - Example: “The max number of hours per employee for all projects that one works is limited up to **52** hours per week.” No way by the model?!
  - Can be specified within application programs
  - Can be enforced by using a general-purpose constraint specification language:
    - `CREATE TRIGGER` and `CREATE ASSERTION` statements for triggers and assertions (확인) (in SQL-99)

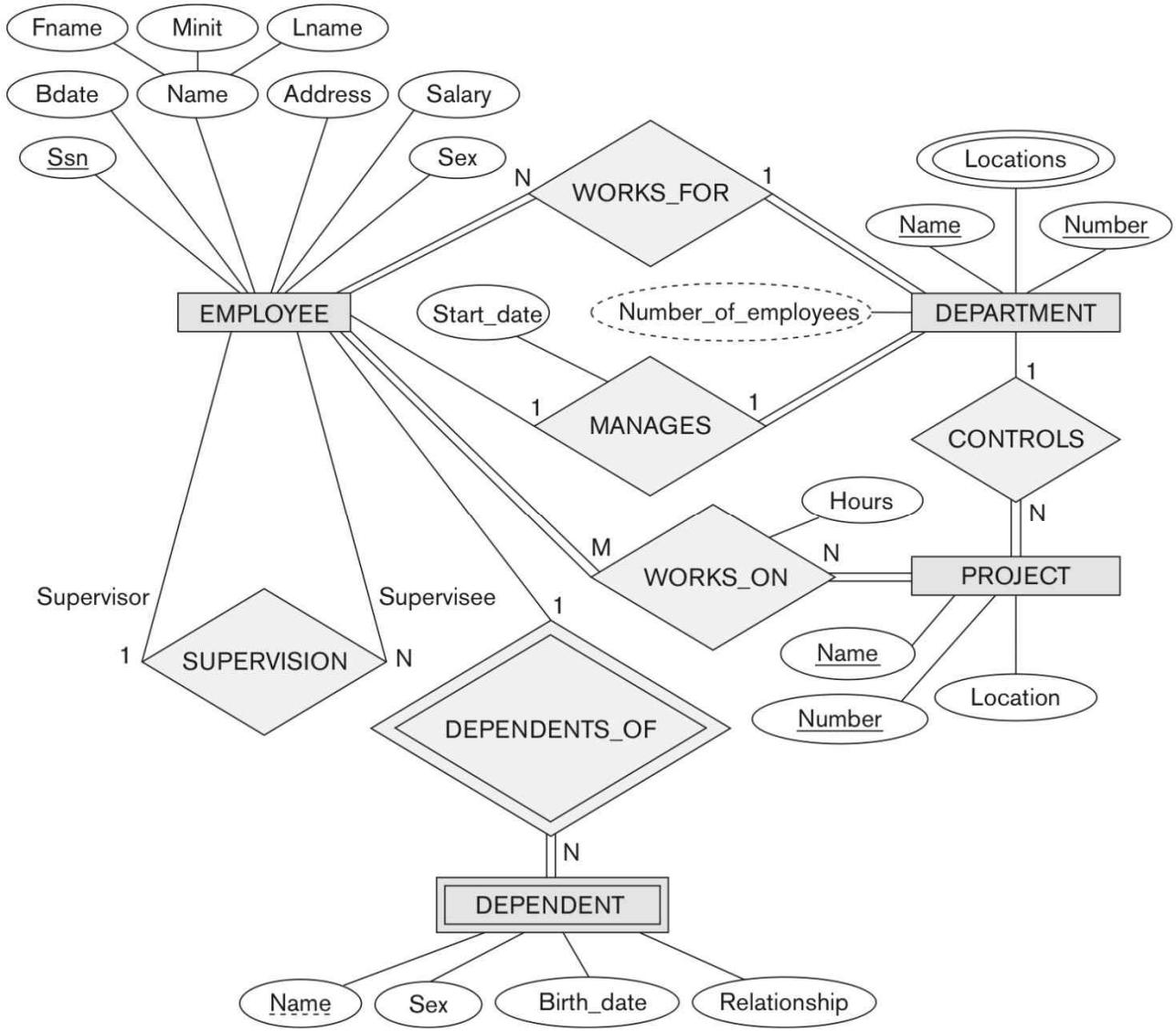
* `CREATE TABLE` statement can allow for declaring:

- Various constraints over (i) keys, (ii) candidate keys (unique constraint), (iii) NOT NULL, (iv) entity integrity, (v) foreign keys, (vi) referential integrity, etc.

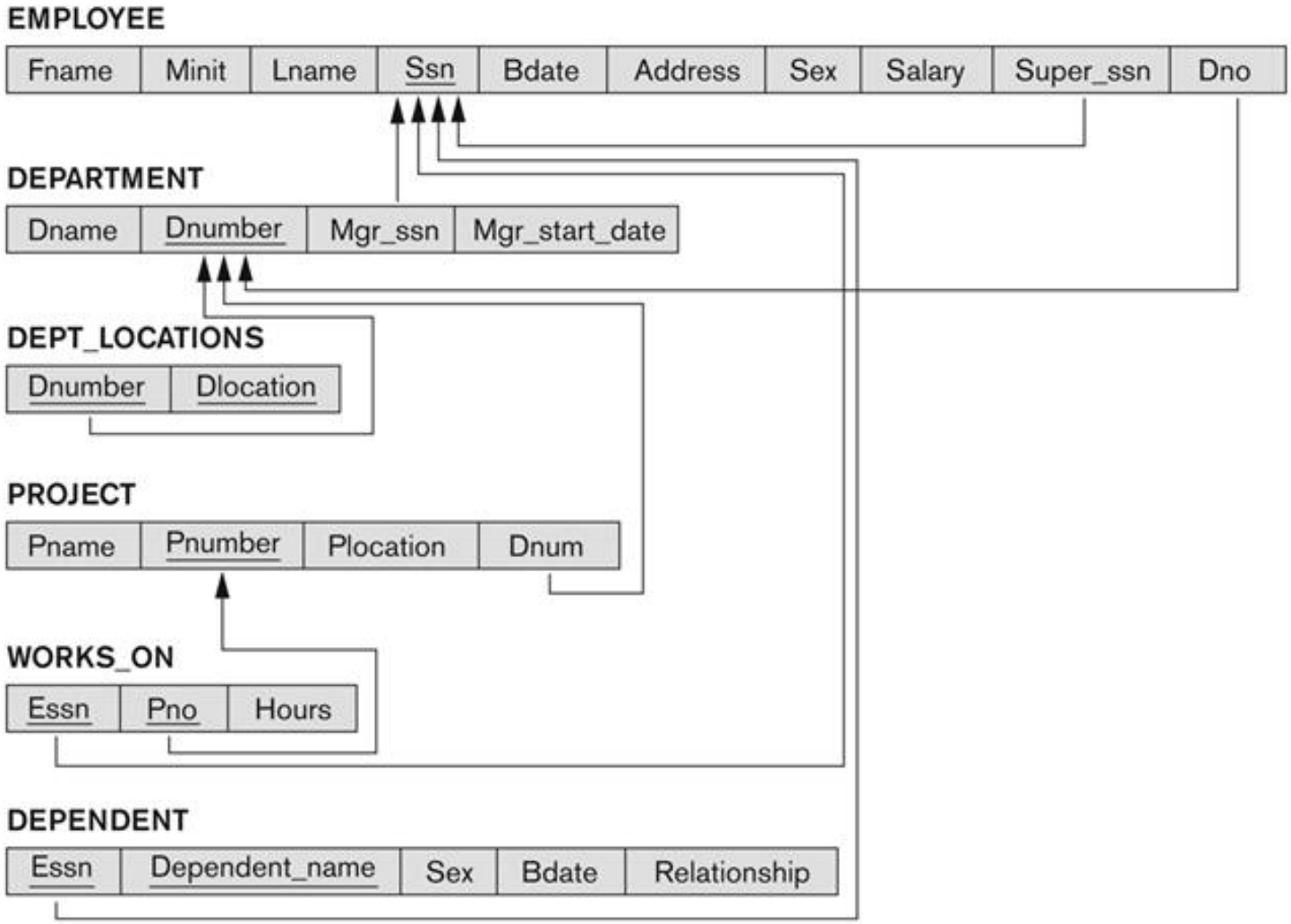
RELATIONAL DATABASE DESIGN BY ER-TO-RELATIONAL MAPPING

Chapter 9

An ER Schema for COMPANY



The Relational Schema for COMPANY



The ER-Relational Mapping Algorithm

- Step 1: Mapping of Regular Entity Types
- Step 2: Mapping of Weak Entity Types
- Step 3: Mapping of Binary 1:1 Relation Types
- Step 4: Mapping of Binary 1:N Relationship Types.
- Step 5: Mapping of Binary M:N Relationship Types.
- Step 6: Mapping of Multivalued attributes.
- Step 7: Mapping of N-ary Relationship Types.

Step 1: Mapping of Regular Entity Types

- For each strong entity type E in the ER schema, create a relation ***R*** that includes all the simple attributes of E.
- Choose one of the key attributes of E as the primary key for ***R***.
- If the chosen key of E is composite, the set of simple attributes chosen for the key will together become the primary key (PK) of ***R***.
- Example: The **EMPLOYEE**, **DEPARTMENT**, and **PROJECT** relations (mapped from the regular entities in the ER schema).
  - Relation ***R***: the `EMPLOYEE`, `DEPARTMENT`, `PROJECT` entity types
  - PKs: `Ssn` for the `EMPLOYEE`, `Dnumber` for `DEPARTMENT`, `Pnumber` for `PROJECT` relations, respectively.

Step 2: Mapping of Weak Entity Types

- For each weak entity type W in the ER schema with owner entity type E,
  - Create a relation ***R*** and
  - Include all simple attributes (or simple components of composite attributes) of W as attributes of ***R***.
- Also, include as foreign key (FK) attributes of ***R*** the PK attribute(s) of the relation(s) that correspond to the owner entity type(s).
- The PK of ***R*** is the **combination** of (i) the PK(s) of the owner and (ii) the partial key of the weak entity type W, if any.
- Example: The **DEPENDENT** relation
  - PK: {Essn, Dependent_name}

Step 3: Mapping of Binary 1:1 Relation Types

- For each binary 1:1 relationship type R in the ER schema, identify the relations ***S*** and ***T*** that correspond to the entity types participating in R. Three possible options:

  1) *Foreign key approach* (2 relations) option:
  - Choose one of the relations, say ***S***.
    - It is better to choose an entity type with total participation in R for ***S***.
  - Include as FK in ***S*** the PK of ***T***.

- Example: The **MANAGES** relationship type in the ER schema
  - Relation ***S***: The `DEPARTMENT` entity type (due to total participation).
  - Relation ***T***: The `EMPLOYEE` entity type
    - The FK of ***S*** (e.g., the `DEPARTMENT` relation), or `Mgr_Ssn` => The PK (e.g., `Ssn`) of ***T*** (e.g., the `EMPLOYEE` relation)

Step 3: Mapping of Binary 1:1 Relation Types (Cont'd)

2) Merged relation (1 relation) option:

- An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation.
- This may be appropriate when both participations are total.

3) Cross-reference or relationship relation (3 relations) option:

- Setting up a third relation ***U*** for the purpose of cross-referencing the PKs of the two relations ***S*** and ***T*** representing the entity types.
- The PK of ***U*** will be one of the FKs from ***S*** and ***T***.
- The other FK will be a unique key of ***U***.
- Extra join operations are required.

Step 4: Mapping of Binary 1:N Relation Types

- For each regular binary 1:N relationship type R, identify the relation ***S*** that represent the participating entity type at the *N*-side of R.
- Include as FK in ***S*** the primary key of the relation ***T*** that represents the other entity type participating in R.
- Include any simple attributes of the 1:N relation type as attributes of ***S***.
- Example: The **WORKS_FOR** relationship type in the ER schema
  - Relation ***S***: The `EMPLOYEE` entity type (N-side)
  - Relation ***T***: The `DEPARTMENT` entity type (1-side)
    - The FK (`Dno`) of ***S***: borrowed from the PK, `Dnumber` of ***T***

Step 5: Mapping of Binary M:N Relation Types

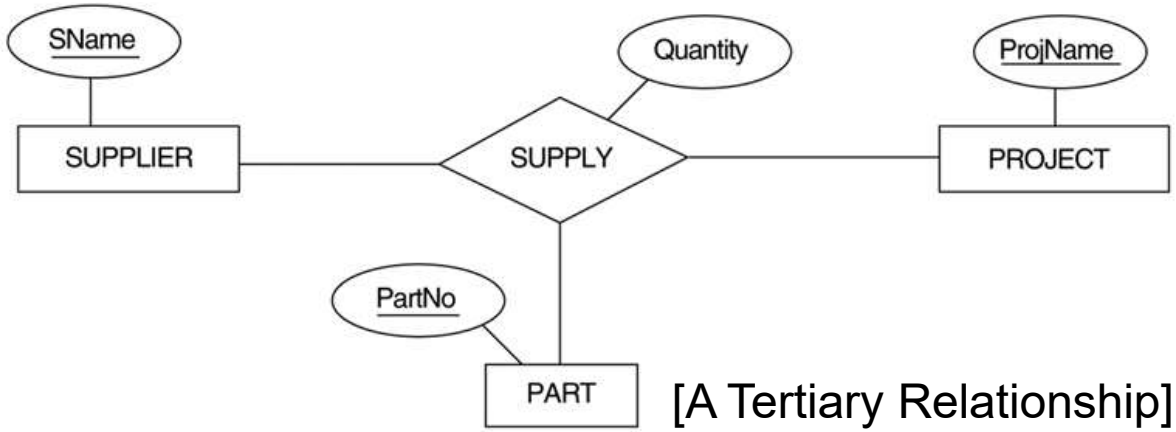
- For each regular binary M:N relationship type R, create a **new** relation ***S***, called a *relationship relation,* to represent R.
- Include as FK attributes in ***S*** the PKs of the relations that represent the participating entity types; their combination will form the PK of ***S***.
- Include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of ***S***.
- Example: The **WORKS_ON** relationship type in the ER schema
  - Relation ***S***: The `WORKS_ON` relationship type
    - FKs (`Essn` and `Pno`): borrowed from the PKs (`Ssn` and `Pnumber`) of the `EMPLOYEE` and `PROJECT` relations, respectively
    - PK: `{Essn, Pno}`
    - One attribute: `Hours`, from `Hours` of the `WORKS_ON` relationship type

Step 6: Mapping of Multivalued Attributes

- For each multivalued attribute A, create a new relation **R**.
- This relation **R** will include
  - An attribute corresponding to A,
  - Its FK, say K, or the PK of the entity type that owns A as an attribute, and
  - The PK: $\{A, K\}$
- Example: The **Locations** multi-valued attribute of `Department`
  - Relation **R**: the **DEPT_LOCATIONS** relation
    - A: `Dlocation`, from `Locations` of the `DEPARTMENT` entity type
    - FK: `Dnumber`, from the PK of the `DEPARTMENT` entity type
    - PK: `{Dlocation, Dnumber}`

Step 7: Mapping of *N*-ary Relationship Types

- For each n-ary relationship type R, where n > 2, create a new relationship ***S*** to represent R.
- Include as foreign key attributes in ***S*** the primary keys of the relations that represent the participating entity types.
- Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of ***S***.
- Example
  - Relation ***S***: the **SUPPLY** relation
  - PKs/FKs => `{Sname, Partno, Projname}`
  - The other attribute: `Quantity`



[A Tertiary Relationship]

Correspondence between ER and Relational Models

ER MODEL		RELATIONAL MODEL
Entity type	→	*Entity* relation
1:1 or 1:N relationship type	→	Foreign key (or *relationship* relation)
M:N relationship type	→	*Relationship* relation and *two* foreign keys
n-ary relationship type	→	*Relationship* relation and *n* foreign keys
Simple attribute	→	Attribute
Composite attribute	→	Set of simple component attributes
Multivalued attribute	→	Relation and foreign key
Value set	→	Domain
Key attribute	→	Primary (or secondary) key

UPDATE OPERATIONS AND DEALING WITH CONSTRAINT VIOLATIONS

Chapter 5.3

Update Operations on Relations (릴레이션에 대한 갱신 연산)

- (1) INSERT, (2)DELETE, or (3) MODIFY(UPDATE) a tuple.
- Whenever these operations are applied, **NO** violation of integrity constraints should happen on the relational DB.
  - We discuss the following things:
    - The *types of constraints* that may be violated
    - The *types of actions* that may be taken if the violation occurs
- Several update operations may have to be grouped together.
- Updates may propagate to cause other updates automatically.
  - This may be necessary to maintain integrity constraints.
  - E.g., A new employee comes, and then the total number of employees increments by one.

Update Operations on Relations (Cont'd)

- In case of integrity violation, several actions can be taken:

1) Cancel the operation, causing the violation
   - `RESTRICT` (no action) or `REJECT` option; taken by many DBMSes
2) Perform the operation but inform the user of the violation
   - Not desired; most of the DBMSes don't do so
3) Trigger additional updates so the violation is corrected
   - `CASCADE` (갱신이 전파됨), `SET NULL`, or `SET DEFAULT` option: from referencing to referenced
4) Execute a user-specified error-correction routine.
   - A program needs to be written.

Possible Violation Cases for INSERT Operation

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4

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

- Insert <'Cecillia', 'F', 'Kolonsky', **NULL**, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4> into EMPLOYEE.

  Any problem? If any, which constraint is violated?

- Insert <'Alicia', 'J', 'Zelaya', '**999887777**', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, '987654321', 4> into EMPLOYEE. *Any problem?*

- Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windswept, Katy, TX', F, 28000, '987654321', **7**> into EMPLOYEE. *Any problem?*

Besides, *domain constraints* could be violated, or insertion can be accepted.

Possible Violation Cases for DELETE Operation

- Delete the `WORKS_ON` tuple with `Essn = '999887777'` and `Pno = 10`.

WORKS_ON

Essn	Pno	Hours
999887777	10	10.0

- Delete the `EMPLOYEE` tuple with `Ssn = '999887777'`.
  - 2 options: 1) *reject* due to many tuples referencing the tuple, or 2) *deleting it and then cascading the deletion* to `WORKS_ON`.

- Delete the `EMPLOYEE` tuple with `Ssn = '333445555'`.
  - Worst case as many tuples are affected. See the tables…

Possible Violation Cases for DELETE Operation - Summary (Cont'd)

- DELETE may violate only referential integrity.
  - Why? Because the tuple being deleted is "referenced" by foreign keys from other tuples in the database. Three options:

  1) `RESTRICT` option: reject the deletion
  2) `CASCADE` option: do the deletion of the tuple followed by the deletion of the tuples that referencing the tuple being deleted
  3) `SET NULL` or `SET DEFAULT` option: set the foreign keys of the referencing tuples to `NULL` or default values after the deletion

- One of the above options must be specified during database design for each foreign key constraint.
  - We will have some hands-on labs, probably next week.

Possible Violation Cases for UPDATE Operation

- Update the salary of the `EMPLOYEE` tuple with `Ssn = '999887777'` and `28000`. Problem?

- Update the `Dno` of the `EMPLOYEE` tuple with `Ssn = '999887777'` to `7`. Problem?

- Update the `Ssn` of the `EMPLOYEE` tuple with `Ssn = '999887777'` to `'987654321'`. Problems?

Possible Violation Cases for UPDATE Operation - Summary (Cont'd)

- UPDATE operations may violate (i) domain constraint and (ii) NOT NULL constraint on an attribute being modified.
- Any of the other constraints may also be violated, depending on the attribute being updated.

1) If that attribute is a primary key (PK) attribute,
   - Similar to deleting one tuple and inserting a new tuple in its place
   - To handle, similar options to DELETE need to be specified.
2) If that attribute is a foreign key (FK) attribute,
   - Most likely to violate referential integrity constraint
   - To handle, similar options to DELETE need to be specified.
3) If that attribute is neither a PK nor a FK attribute,
   - Can only violate domain constraints or Not null constraints
   - Perhaps, simply reject