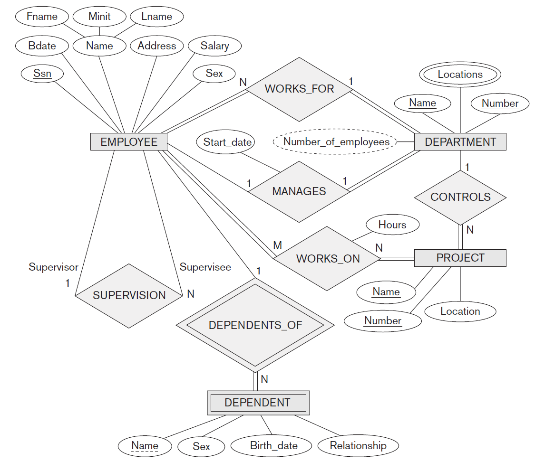
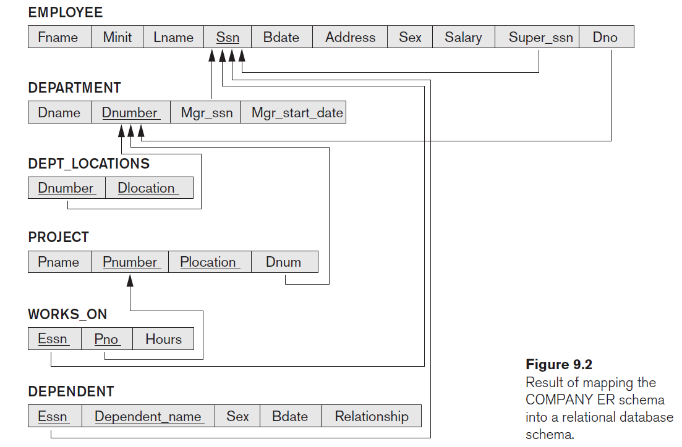
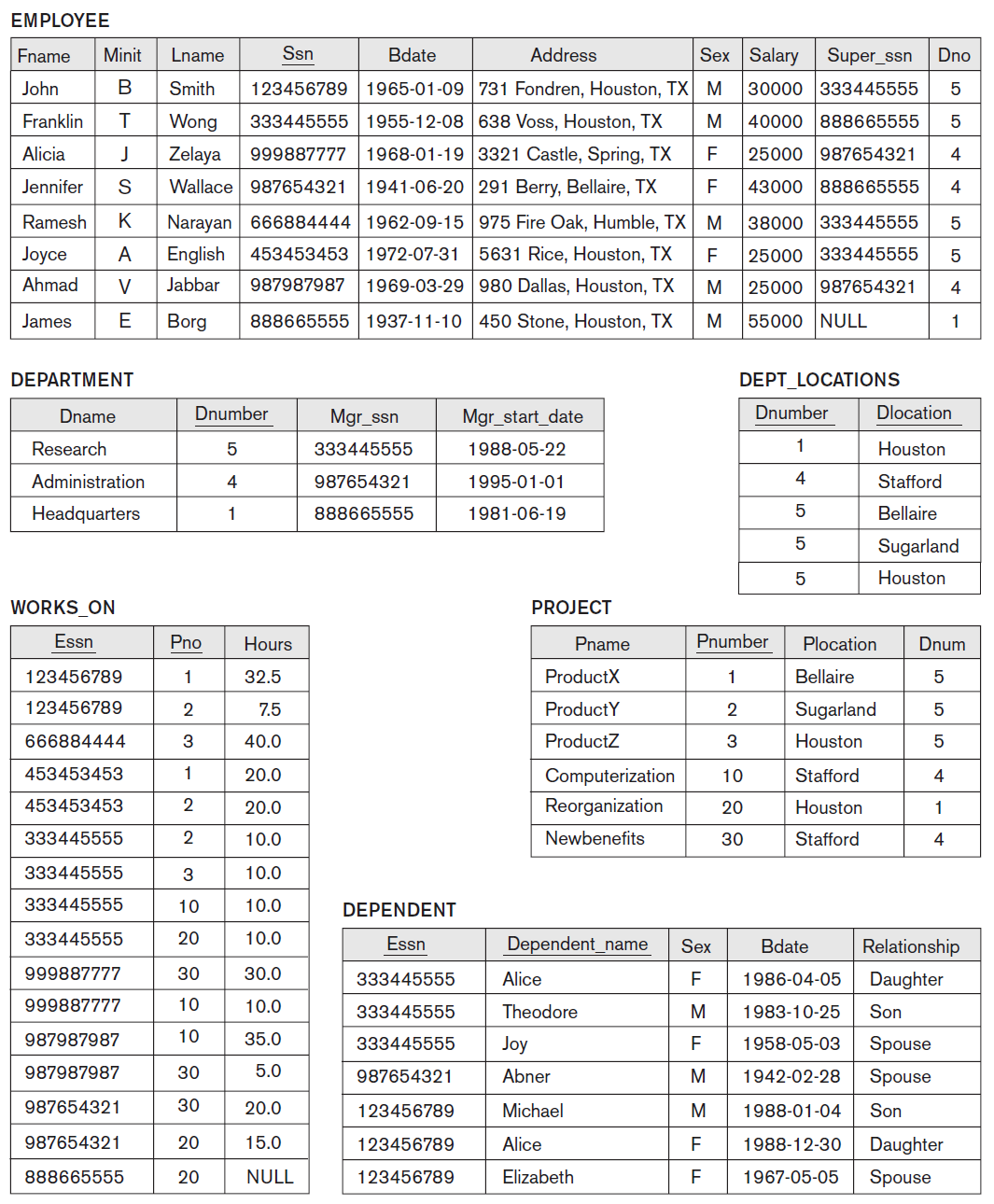
# Chapter 5: The Relational Data Model and Relational Database Constraints

In order to design a database for a project, first we have the project description as a text or some paragraphs. In chapter 3, we learned ER model, and how to convert the project description text to graphical ER diagram. Then we introduced a 7-steps algorithm to convert that ER to RM model. The ER and RM models for the COMPANY example are shown below;



→

The content of RM model could look like the following figure;

This content is also called a database …………….

The content of each database table could be written as a set of tuples, for example content of DEPARTMENT table is the following set:

{(Research, 5, 333445555, 1988-05-22), (Administration, 4, 987654321, 1995-01-01), (Headquarters, 1, 888665555, 1981-06-19)}

And this is a relation (in mathematical terms).

Table == relation

Table row == record == tuple

## Constraints

Constraints determine which values are permissible and which are not in the database.

They are of three main types:

1. **Inherent or Implicit Constraints**: These are based on the data model itself.

Example: relational model does not allow a list as a value for an attribute.

1. **Schema-based or Explicit Constraints**: They are expressed in the schema by using the tools provided by the model. (E.g., min-max cardinality ratio constraint in the ER model)
2. **Application based or semantic constraints**: These are beyond the expressive power of the model and must be specified and enforced by programming in the application programs.

For example; a student whose GPA drops below 2.0 in two consecutive semesters, is not allowed to register for the next semester.

* Constraints are **conditions** that must be held true on **all** valid database states.

In this chapter we focus on schema-based constraints; there are three *main types* of such constraints that can be expressed in the relational model:

* + **Key** constraints
  + **Entity** **integrity** constraints
  + **Referential integrity** constraints

### Key constraints

* **Superkey** of a relation R is a set of attributes SK of R such that no two tuples will have the same value for SK:
  + Notation: t[Ai] is the value of attribute Ai in the tuple t.

For example t = (Research, 5, 333445555, 1988-05-22), a tuple for DEPARTMENT, t[Dname] = …………….

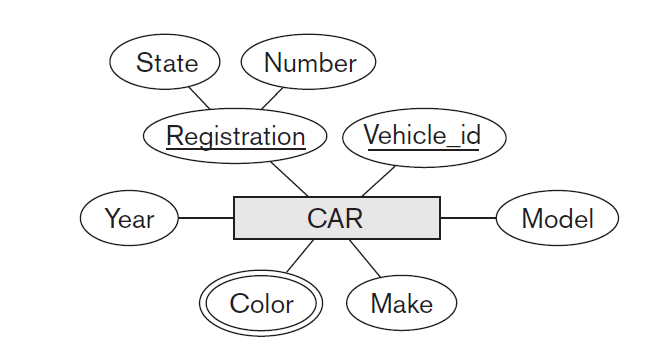
* + If SK is a superkey, then for distinct tuples t1 and t2 in R, t1[SK] ≠ t2[SK].

For example, for t1 and t2 to be two different tuples in DEPARTMENT relation t1[Dnumber] ≠ t2[Dnumber].

You can also say **Superkey** of a relation R is a set of attributes SK of R that uniquely identifies each tuple in that relation.

* **Key** of R is a "minimal" superkey!
  + 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 anymore (does not have the superkey uniqueness property).

Example:

* consider the CAR relation defined as below:
  + CAR(State, Number, Vehicle\_id, Make, Model, Year)
  + CAR has two keys (in ER model):
    - Key1 = {State, Number}
    - Key2 = {Vehicle\_id}
  + Both are also superkeys of CAR

Minimum vs. minimal

* + Key1 = {State, Reg#} 🡺 minimal
  + Key2 = {Vehicle\_id} 🡺 minimum

In general:

* + Any *key* is a *superkey* (but not vice versa).
  + Any set of attributes that *includes a key* is a *superkey.*
  + A *minimal* superkey is a key.

In CAR example, { Vehicle\_id, Make} is a superkey but not a key. Why?

If a relation has several **candidate keys**, one is chosen arbitrarily to be the **primary key**. The primary key attributes are underlined.

* Example: Consider the CAR relation schema:

CAR(State, Reg#, Vehicle\_id, Make, Model, Year)

We chose Vehicle\_id as the primary key.

Choosing primary key is subjective!

* May police dep. use {State, Reg#} as key. (easier to read).

Primary key also used to *reference* to a tuple from another tuple; so better to choose the smallest of the candidate keys for primary key (a general rule).

### Entity Integrity:

* The *primary key attributes* PK of relation R **cannot have null values in a tuple** of R.
  + This is because primary key values are used to *identify* the individual tuples.
  + t[PK] ≠ null for any tuple t in R.
  + If PK contains several attributes, null is not allowed in any of the attributes.
* **Not-null constraint** may be applied on non-primary key attributes as well.

### Referential Integrity

* Consider relations R1 and R2, **foreign key** FK and **primary key** PK.

A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].

* R1 is called **referencing relation**.
* R2 is called **referenced relation**.
* A referential integrity can be displayed as a directed arc from R1.FK 🡺 R2.PK.

### Referential Integrity (or foreign key) Constraint

* The value in the foreign key attribute (or attributes) FK of the **referencing relation** R1 can be **either**:

(1) a value of an existing primary key PK in the **referenced relation** R2, or

(2) a **null**.

* In case (2), the FK in R1 should **not** be a part of its primary key. What kinds of constraint would be violated if we allow it to be null?

**Domain** constraint

* + Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)