CHAPTER 1

# EXAMPLES- CHAPTER 1

## 1.2 Relational Model

**Example 1.2.1.** Consider the following problem:

> A company is organised into departments. Each department has a unique number, and a particular employee who manages the department. We keep track of the start date that the employee began managing the department. A department may have several locations. A department controls a number of projects, each of which has a unique name, number and a single location. The company stores the information about the employee (e.g. name, salary, birth date, etc.) and the unique social security number (SSN). An employee is assigned only to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current hours per week that an employee works on each project and their direct supervisor, who is another employee. The company keeps track of the dependent (e.g. child) of each employee for insurance purposes and the corresponding relationship with the employee (e.g. son, daughter).

Create a relational model for the description above.

From the description, we find that:

- the schema is the company;
- the entities are: department, employee, project and dependent; and
- we have the following relationships:
  - an employee manages a department;
  - a department may have several locations;
  - a department controls a number of projects;
  - an employee is assigned to one department;
  - an employee may work on several projects which are not necessarily controlled by the same department;
  - a department controls several projects;
  - an employee is supervised by a supervisor, who is another employee;
  - an employee has several dependents, each one corresponding to a specific relationship with the employee.

Using this, we get the following relations:

- Employee(Name, SSN, BDate, Address, Salary, Supervisor, Department),
- Department(Name, Number, Manager, ManagerStartDate),
- Dept_Location(Department, Location),
- Project(Name, Number, Location, Department),
- Works_In(Employee, Project, Hours),
- Dependent(Employee, Name, BDate, Relationship).

The relations Employee, Department, Project and Dependent come from the identified entities. We have the relations Dept_Location and Works_In to represent relationships. The underlined attributes are primary keys and the dotted attributes are foreign keys.

## 1.3 Functional Dependency

**Example 1.3.1.** Assume we have the following relational schema.

- `Reader(Name, Age, Book Title, ID)`,
- `Book(Title, Subject, Number of Pages, ISBN)`.

Adapt the schema so that we obey relational constraints.

The PK of the reader record is the `ID`, and the PK of book is `ISBN`. The two relations are related by the `title` attribute. However, since title is not unique, it should be related by the `ISBN` attribute. Moreover, this relationship is going from `Book` to `Reader`- a person cannot read a book with ISBN that doesn't exist. So, the records should actually be the following:

- `Reader(Name, Age, ISBN, ID)`,
- `Book(Title, Subject, Number of Pages, ISBN)`.

This obeys the relational constraints.

**Example 1.3.2.** Assume that we have the following relation.

$$R(B, O, I, S, Q, D)$$

We have the following FDs.

- FD1: $S \to D$
- FD2: $I \to B$
- FD3: $\{I, S\} \to Q$
- FD4: $B \to O$.

Show that $\{I, S\}$ is a candidate key but $\{I, B\}$ is not.

- We show that we have the $\{I, S\}$ is a candidate key. We know that $I \to B$ and $B \to O$. So, transitivity tells us that $I \to O$. Therefore, $I \to \{B, O\}$. Moreover, $S \to D$, so $\{I, S\} \to \{B, O, D\}$. Since we also have $\{I, S\} \to Q$, we find that $\{I, S\} \to \{B, O, Q, D\}$. So, we have shown that $\{I, S\}$ determines all the attributes in the relation. This means that $\{I, S\}$ is a candidate key.
- Now, we show that $\{I, B\}$ cannot be a candidate key. We know that $I \to B$ and $B \to O$, so $I \to O$. So, $\{I, B\} \to \{I, B, O\}$. We cannot add any further attribute into the set using the FDs given above- $D$, $Q$ and $S$ are not dependent on $\{I, B\}$. So, $\{I, B\}$ cannot be a candidate key.

## 1.4 Normalisation Theory

**Example 1.4.1.** Consider the Department relation below.

| `DName` | `DNumber` | `DManager` | `DLocations` |
|---|---|---|---|
| Research | 5 | 333445555 | {Beltaire, Sugarland, Houston} |
| Administration | 4 | 987654321 | {Stattford} |
| Headquarters | 1 | 888866555 | {Houston} |

Convert this relation into 1NF.

This relation is not in 1NF since the `DLocations` attribute is multi-valued. We normalise it by introducing a tuple for each of the value present, i.e. the following record:

| `DName` | `DNumber` | `DManager` | `DLocations` |
|---|---|---|---|
| Research | 5 | 333445555 | Beltaire |
| Research | 5 | 333445555 | Sugarland |
| Research | 5 | 333445555 | Houston |
| Administration | 4 | 987654321 | Stattford |
| Headquarters | 1 | 888866555 | Houston |

**Example 1.4.2.** Consider the following relation.

`EMP_PROJ(`<u>`SSN`</u>`, `<u>`PNumber`</u>`, Hours, EName, PName, PLocation)`

Normalise it to 2NF.

This relation is in 1NF since every attribute is single-valued. We have the following FDs:

- {`SSN`, `PNumber`} → `Hours` is a full FD;
- {`SSN`, `PNumber`} → `EName` is a partial FD with `SSN` → `EName` a full FD;
- {`SSN`, `PNumber`} → `PName` is a partial FD with `PNumber` → `PName` a full FD;
- {`SSN`, `PNumber`} → `PLocation` is a partial FD with `PNumber` → `PLocation` a full FD;

So, we will create 3 relations:

`R1(`<u>`SSN`</u>`, `<u>`PNumber`</u>`, Hours)`
`R2(`<u>`SSN`</u>`, EName)`
`R3(`<u>`PNumber`</u>`, PName, PLocation)`

Now, each of the three relations are in 2NF- each non-prime attribute fully depends on the primary key.

**Example 1.4.3.** Assume that we have the following relation.

`EMP_DEPT(SSN, EName, BYear, Address, DNumber, DName, DMgr_SSN)`

Normalise it to 3NF.

The relation is in 1NF since every attribute is single-valued. Moreover, since the primary key is a single attribute, the relation is in 1NF. Also, the following are the FDs.

- the FD `SSN` $\to$ `DMgr_SSN` is a transitive dependency via the non-prime attribute `DNumber`;
- the FD `SSN` $\to$ `DName` is a transitive dependency via the non-prime attribute `DNumber`;
- the FD `SSN` $\to$ `EName` is a direct dependency since there is no non-prime attribute that determines `EName`;
- the FD `SSN` $\to$ `BYear` is a direct dependency since there is no non-prime attribute that determines `BYear`;
- the FD `SSN` $\to$ `Address` is a direct dependency since there is no non-prime attribute that determines `Address`.

We transform a record in 2NF to a record in 3NF by splitting it into the non-prime transitive attribute and the other attributes. So, we split this record into the 2 records:

`R1(SSN, EName, BYear, Address, DNumber)`
`R2(DNumber, DName, DMgr_SSN).`

By storing `DNumber` in `R1`, we can join the two records correctly. It is a foreign key. The two relations are in 3NF.

**Example 1.4.4.** Consider the following instance of the `TEACH` relation.

| Student | Course | Instructor |
|---|---|---|
| Naranyan | Database | Mark |
| Smith | Database | Navathe |
| Smith | Operating Systems | Ammar |
| Smith | Theory | Schulman |
| Wallace | Database | Mark |
| Wallace | Operating Systems | Ahamad |
| Zelaya | Database | Navathe |

Normalise it to BCNF.

This relation is in 1NF since the attributes are single-valued. Moreover, the FDs are:

- $\{$`Student`, `Course`$\}$ $\to$ `Instructor`
- `Instructor` $\to$ `Course`

So, the relation is in 2NF and 3NF, but not in BCNF. The violating attribute here is `Instructor` $\rightarrow$ `Course`, so $X = \{$`Instructor`$\}$ and $A = \{$`Course`$\}$. By the BCNF theorem, we know that we should create the relations:

```
R1(Student, Instructor)
R2(Instructor, Course).
```

This gives us the following records:

| Student | Instructor |
|---|---|
| Naranyan | Mark |
| Smith | Navathe |
| Smith | Ammar |
| Smith | Schulman |
| Wallace | Mark |
| Wallace | Ahamad |
| Zelaya | Navathe |

| Course | Instructor |
|---|---|
| Database | Mark |
| Database | Navathe |
| Operating Systems | Ammar |
| Theory | Schulman |
| Operating Systems | Ahamad |

Joining these two records will give us back the original record.