Database Development & Design

LOGICAL DESIGN (DATA MODEL MAPPING)

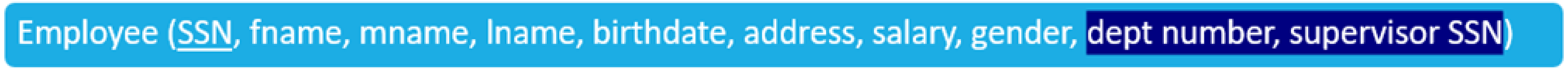
Outline

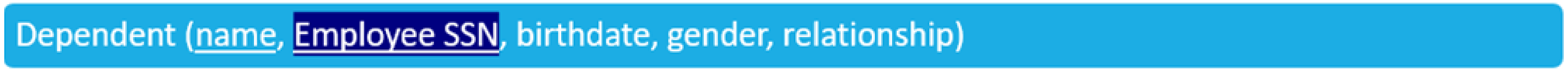
- Strong Entity Vs. Weak Entity
- Logical Model Design
- Mapping: Rules and Steps
- Logical Design Scenarios

Strong and weak entities

- A **strong entity** has a primary key, meaning that it can be uniquely identified by its attributes alone.
- A **weak entity** is an entity that can't be uniquely identified by its attributes alone. It is existence-dependent on its parent entity. It must use a foreign key in conjunction with its attributes to create a primary key.
- we call the relationship type that relates a weak entity type to its owner the **identifying relationship**
- The identifying entity type is also sometimes called the **parent entity** type or the dominant entity type.
- The weak entity type is also sometimes called the **child entity** type or the subordinate entity type.
- A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.
- The partial key is sometimes called the **discriminator**

Strong and weak entities


Employee (SSN, fname, mname, lname, birthdate, address, salary, gender, dept number, supervisor SSN)


Dependent (name, Employee SSN, birthdate, gender, relationship)

- **Employee is a strong entity** because it has a **primary key** attribute (SSN) which is capable of uniquely identifying all the employee.
- Unlike Employee, **Dependents is weak entity.**

Logical Model

- A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Logical model include:
  - All entities and relationships among them.
  - All attributes for each entity are specified.
  - The primary and foreign keys for each entity is specified.
  - Resolves M:N relationships.
  - Create normalized business rules.

Database Schema

- A schema for a database is an outline of how data is organized. It can be a graphic illustration, or another kind of chart used by programmers to understand how each table is laid out, including the columns and the types of data they hold and how tables connect.
- A database schema usually specifies which columns are primary keys in tables and which other columns have special constraints such as being required to have unique values in each record.
- It also usually specifies which columns in which tables contain references to data in other tables, often by including primary keys from other table records so that rows can be easily joined (foreign key columns).

Mapping Steps

Mapping

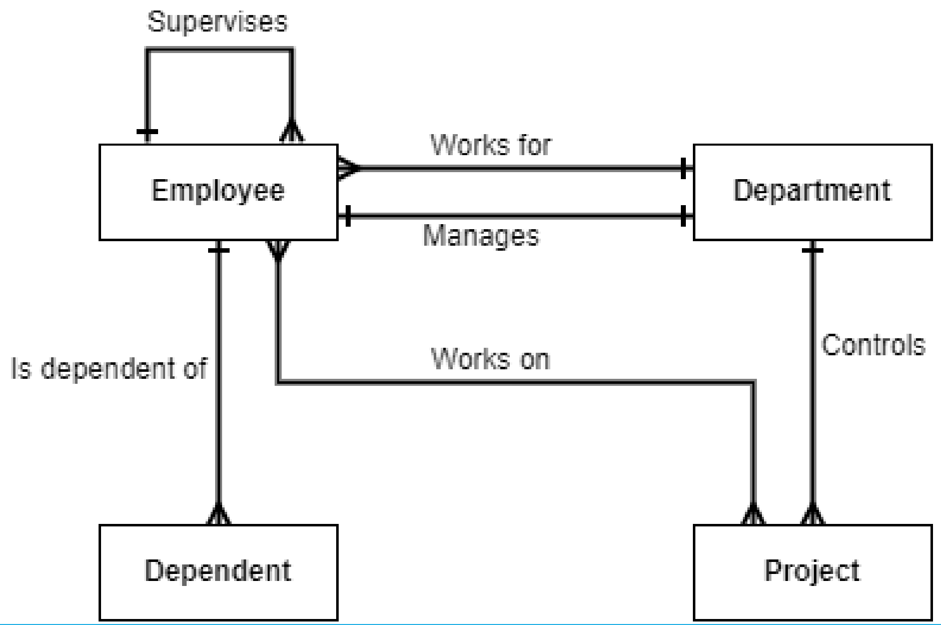
- **Mapping**: is the procedures to create a relational schema from an entity–relationship (ER).
- **Rules of Mapping:**
  1. Every entity is a table.
  2. Every attribute is represented as a column.
  3. Any derived attribute **should not** be mapped because we can calculate it from another attribute.
  4. Any **multi-valued** attribute should be extracted to **new table**.
  5. Any **composite** attribute should be **separated** so that every attribute has its **individual column**.
  6. Any **complex attribute** should be extracted to a **new table**.
  7. In **1:1** relation and **1:M** relation, we should make a link between tables by adding reference from the **PK** to **FK**.
  8. Any many to many (**M:N**) relation will be represented as **new table**. And the primary key of this table will be a combination of the primary keys of the two ends of the relation.

Company Example

Scenario:

The company is organized into **departments**. Each department has an **employee** who manages the department. Each department controls a number of **projects**. Each employee works for one department but may work on several projects. Each employee may have a number of **dependents**.

Conceptual Model



Company Example - Details

Scenario:

The company is organized into **DEPARTMENTs**. Each department has a **name**, **number** and an **employee** who manages the department. We keep track of the **start date** of the department manager. A department may have several **locations**. Each department controls a number of **PROJECTs**. Each project has a unique **name**, unique **number** and is located at a single **location**. Each EMPLOYEE has a **social security number**, **salary**, **gender**, and **birthdate**. Each employee works for one department but may work on several projects. We keep track of **the number of hours per week** that an employee currently works on each project. It is required to keep track of the **direct supervisor** of each employee. Each employee may have a number of **DEPENDENTs**. For each dependent, we keep a record of **name**, **sex**, **birthdate**, and **relationship** to the employee.

Mapping Steps

- **Step 1:** Mapping of Regular (Strong) Entity Types
- **Step 2:** Mapping of Weak Entity Types
- **Step 3:** Mapping of 1:1 Relationship Types.
- **Step 4:** Mapping of 1:N Relationship Types.
- **Step 5:** Mapping of M:N Relationship Types.
- **Step 6:** Mapping of Multivalued Attributes.

Mapping - Composite

- Employee - Before Mapping:

SSN	name	birthdate	gender	salary
123456789	John B Smith	1/9/1965	M	30000

- Employee - After Mapping:

SSN	fname	mname	lname	birthdate	gender	salary
123456789	John	B	Smith	1/9/1965	M	30000

Mapping – Multi-value

- Department - Before Mapping:

number	name	locations
1	HeadQuarters	Houston
4	Administration	Stafford
5	Research	{Bellaire, Houston, Sugarland}

- Department - After Mapping

number	name
1	HeadQuarters
4	Administration
5	Research

- Department-Location - After Mapping

dept_number	location
5	Bellaire
1	Houston
5	Houston
4	Stafford
5	Sugarland

Database Schema –With strong Entities (Relations) Only

Employee (SSN, fname, mname, lname, birthdate, salary, gender)

Department (number, name)

Department location (dept number, location)

Project (number, name, location)

Database Schema – With Weak Entities (Relations)

Employee (SSN, fname, mname, lname, birthdate, salary, gender)

Department (number, name)

Department location (dept number, location)

Project (number, name, location)

Dependent (name, Employee SSN, birthdate, gender, relationship)

Mapping – 1:1 Relations

Relation between Employee and Department (Mange Relationship)

- Department - Before Mapping

Department (<u>number</u>, name)

Mapping:

- Include the primary key **SSN** of the **Employee** relation as a foreign key attribute of Department (renamed to **manager SSN**).
- Attribute **start date** in **department** represents the **start date** attribute of the relationship type.

- Department - After Mapping

Department (<u>number</u>, name, start date, manager SSN)

Mapping – 1:M Relations

- **Examples:**
  - For **works for**, we include the primary key **number** of the **Department** relation as foreign key in the **Employee** relation and call it **dept number**.
  - For **controls**, we include the primary key **number** of the **Department** relation as foreign key in the **Project** relation and call it **dept number**.
  - For **supervises**, we include the primary key **SSN** of the **Employee** relation as foreign key in the **Employee** relation and call it **supervisor SSn**.

Employee (SSN, fname, mname, lname, birthdate, salary, gender, dept number, supervisor SSN)

Department (number, name, start date, manager SSN)

Depart loc (dept number, location)

Project (number, name, location, dept number)

Dependent (name, Employee SSN, birthdate, gender, relationship)

Mapping – M:N Relations

- **Example**: The M:N relationship type **works on** is mapped by creating a relation **Employee Project**.
- The primary keys of the **Project** and **Employee** relations are included as foreign keys in **Employee Project** and renamed **proj number** and **emp SSN**, respectively.
- Attribute **hours** in **Employee Project** represents the **hours** attribute of the relationship type. The primary key of the **Employee Project** relation is the combination of the foreign key attributes {**proj number**, **emp SSN**}.

Employee Project (Emp SSN, Proj number, hours)

Finally - Database Schema

Employee (SSN, fname, mname, lname, birthdate, salary, gender, dept number, supervisor SSN)

Department (number, name, start date, manager SSN)

Depart loc (dept number, location)

Project (number, name, location, dept number)

Dependent (name, Employee SSN, birthdate, gender, relationship)

Employee Project (Emp SSN, Proj number, hours)

Note: Primary keys are underlined, and foreign keys are highlighted in dark blue

Finally - Database Schema

Employee (SSN, fname, mname, lname, birthdate, salary, gender, dept number, supervisor SSN)

Department (number, name, start date, manager SSN)

Depart loc (dept number, location)

Project (number, name, location, dept number)

Dependent (name, Employee SSN, birthdate, gender, relationship)

Employee Project (Emp SSN, Proj number, hours)

Note: Primary keys are underlined, and foreign keys are highlighted in dark blue

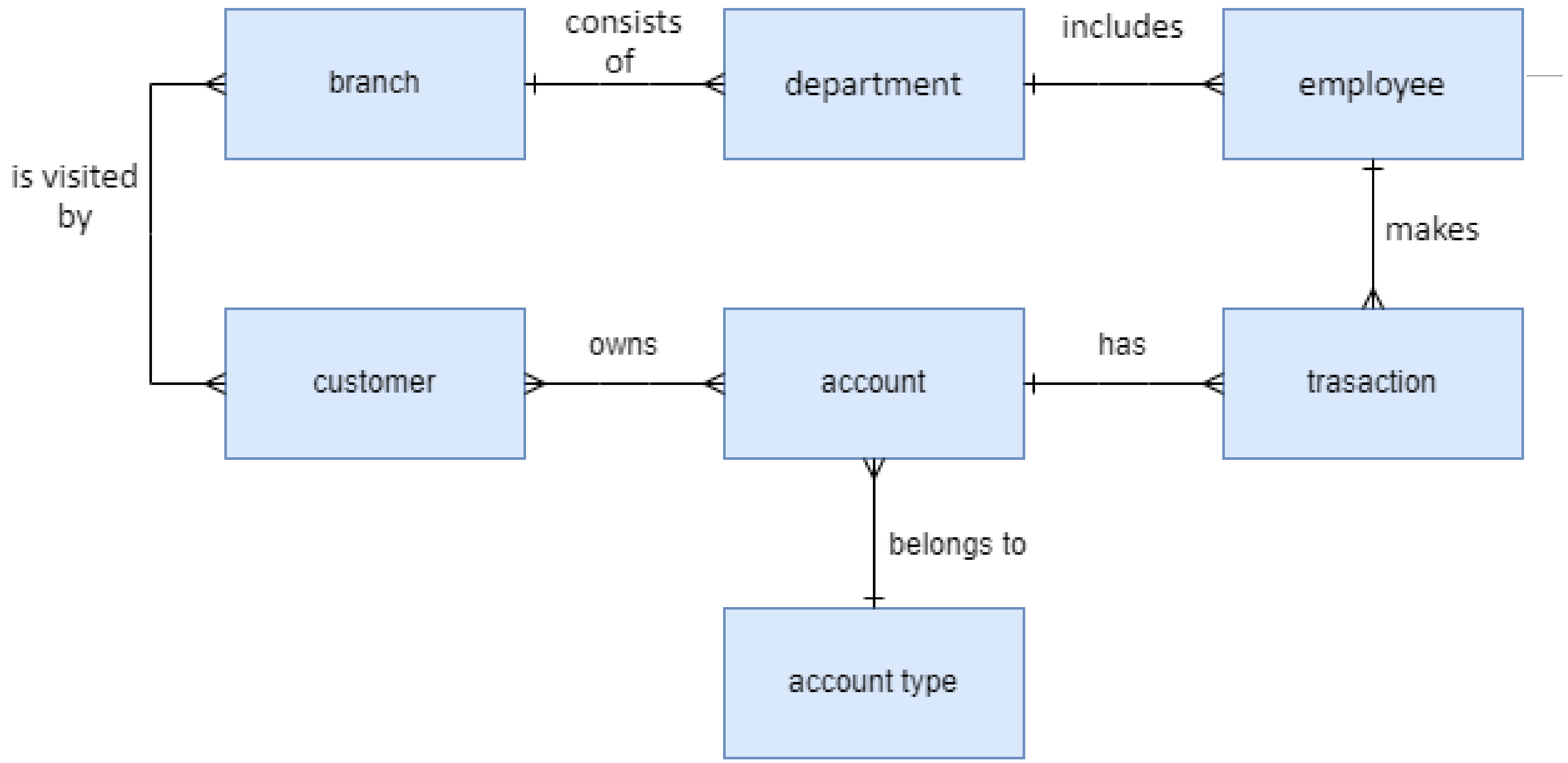
Database Scenario – with Crow’s Foot Notations

Bank Management System Example

Scenario:

The bank has many **branches**. Each branch consists of multiple **departments**, and each department includes many **employees**. The employee should work in one department. An employee can make many **transactions**. Each transaction is performed on one **account**. Note: the account may have multiple transactions. In addition, the account is owned by one or more **customers** and customers can have multiple accounts. There are many **account types** like (saving, checking, and money market accounts). For each account, there is one account type. Customers can visit multiple branches.

Conceptual Model



Bank Management System Example

Scenario:

The bank has many **branches**. Each branch has a number, name, phone, and Address. Each branch consists of multiple **departments.** Each department has a name and phone and includes many **employees**. Each employee has an id, name, password, email, salary, level, and address. The employee should work in one department. An employee can make many **transactions**. Each transaction has an id, type, date, time, and channel. Each transaction is performed on one **account**. Each account has a type, number, status, and balance. Note: the account may have multiple transactions. In addition, the account is owned by one or more **customers** and customers can have multiple accounts. Each customer has an id, name, SSN, date of birth, email, and can have multiple addresses and multiple phone. There are many **account types** like (saving, checking, and money market accounts). Each account type has an id, name and description. For each account, there is one account type. Customers can visit multiple branches. Each visit is recorded with date and time.

Basic Step

- **Branch** (number, name, phone, address)
- **Department** (name, phone)
- **Employee** (id, name, password, email, salary, level, address)
- **Transaction** (id, type, date, time, channel)
- **Customer** (id, name, SSN, date of birth, email, phones, addresses)
- **Account** (number, status, balance, type)
- **Account type** (id, name, description)

Remember the Rule:
Every entity is a table.
Every attribute is
represented as a column,
and determine PKs.

Derived Attributes

- **Branch** (<u>number</u>, name, phone, address)
- **Department** (name, phone)
- **Employee** (<u>id</u>, name, password, email, salary, level, address)
- **Transaction** (<u>id</u>, type, date, time, channel)
- **Customer** (<u>id</u>, name, SSN, date of birth, email, phones, addresses)
- **Account** (<u>number</u>, status, balance, type)
- **Account type** (<u>id</u>, name, description)

Remember the rule:
Any derived attribute should not be mapped because we can calculate it from another attribute.

No Derived attributes → so no change!

Multi-valued Attributes

- **Branch** (<u>number</u>, name, phone, address)
- **Department** (name, phone)
- **Employee** (<u>id</u>, name, password, email, salary, level, address)
- **Transaction** (<u>id</u>, type, date, time, channel)
- **Customer** (<u>id</u>, name, SSN, date of birth, email, **phones**, **addresses**)
- **Account** (<u>number</u>, status, balance, type)
- **Account type** (<u>id</u>, name, description)

Remember the Rule: Any **multi-valued** attribute should be extracted to **new table**

Multi-valued Attributes

- **Branch** (<u>number</u>, name, phone, address)
- **Department** (name, phone)
- **Employee** (<u>id</u>, name, password, email, salary, level, address)
- **Transaction** (<u>id</u>, type, date, time, channel)
- **Customer** (<u>id</u>, name, SSN, date of birth, email)
- **Customer_addresses** (<u>customer_id, address</u>)
- **Customer_phones** (<u>customer_id, phone</u>)
- **Account** (<u>number</u>, status, balance, type)
- **Account type** (<u>id</u>, name, description)

Composite attributes

- **Branch** (<u>number</u>, name, phone, address)
- **Department** (name, phone)
- **Employee** (<u>id</u>, name, password, email, salary, level, address)
- **Transaction** (<u>id</u>, type, date, time, channel)
- **Customer** (<u>id</u>, name, SSN, date of birth, email)
- **Customer_addresses** (<u>customer_id, address</u>)
- **Customer_phones** (<u>customer_id, phone</u>)
- **Account** (<u>number</u>, status, balance, type)
- **Account type** (<u>id</u>, name, description)

Remember the Rule:
Any composite attribute should be separated so that every attribute has its individual column.

Composite Attributes

- **Branch** (number, name, phone, country, city, street name, building number)
- **Department** (name, phone)
- **Employee** (id, fname, lname, password, email, salary, level, country, city, street name, building number)
- **Transaction** (id, type, date, time, channel)
- **Customer** (id, fname, lname, SSN, date of birth, email)
- **Customer_addresses** (customer_id, country, city, street name, building number)
- **Customer_phones** (customer_id, phone)
- **Account** (number, status, balance, type)
- **Account type** (id, name, description)

Complex Attributes

- **Branch** (<u>number</u>, name, phone, country, city, street name, building number)
- **Department** (name, phone)
- **Employee** (<u>id</u>, fname, lname, password, email, salary, level, country, city, street name, building number)
- **Transaction** (<u>id</u>, type, date, time, channel)
- **Customer** (<u>id</u>, fname, lname, SSN, date of birth, email)
- **Customer_addresses** (<u>customer_id,country, city, street name, building number</u>)
- **Customer_phones** (<u>customer_id, phone</u>)
- **Account** (<u>number</u>, status, balance, type)
- **Account type** (<u>id</u>, name, description)

No complex attributes (already covered) → so no change!

Relations (1:1) and (1:M)

- Branch : Department → 1:M (consists of)
- Department : Employee → 1:M (includes)
- Employee : Transaction → 1:M (make)
- Account: Transaction → 1:M (has)
- Account Type: Account → 1:M (has)

Remember the Rule:
In 1:1 relation and 1:M relation, we should make a link between tables by adding reference from the PK to FK.

Relations (1:1) and (1:M)

- **Branch** (number, name, phone, country, city, street name, building number)
- **Department** (name, **branch_number** **,** phone)
- **Employee** (id, fname, lname, password, email, salary, level, country, city, street name, building number, **department_name, branch_number**)
- **Transaction** (id, type, date, time, channel, **employee_id, account_number**)
- **Customer** (id, fname, lname, SSN, date of birth, email)
- **Customer_addresses** (customer_id,country, city, street name, building number)
- **Customer_phones** (customer_id, phone)
- **Account** (number, status, balance, **type_id**)
- **Account type** (id, name, description)

Relations (M:N)

- Customer : Account → M:N (own)
- Customer : Branch → M:N (visit)

Remember the Rule:
Any many to many (M:N) relation will be represented as new table.
And the primary key of this table will be a combination of the primary keys of the two ends of the relation

Relations (M:N)

- **Branch** (number, name, phone, country, city, street name, building number)
- **Department** (name, branch_number, phone)
- **Employee** (id, fname, lname, password, email, salary, level, country, city, street name, building number, department_name, branch_number)
- **Transaction** (id, type, date, time, channel, employee_id, account_number)
- **Customer** (id, fname, lname, SSN, date of birth, email)
- **Customer_addresses** (customer_id,country, city, street name, building number)
- **Customer_phones** (customer_id, phone)
- **Account** (number, status, balance, type_id)
- **Account type** (id, name, description)
- **Ownership** (customer_id, account_number)
- **Visit** (customer_id, branch_number, date, time)

Final Schema!

- **Branch** (number, name, phone, country, city, street name, building number)
- **Department** (name, branch_number, phone)
- **Employee** (id, fname, lname, password, email, salary, level, country, city, street name, building number, department_name, branch_number)
- **Transaction** (id, type, date, time, channel, employee_id, account_number)
- **Customer** (id, fname, lname, SSN, date of birth, email)
- **Customer_addresses** (customer_id,country, city, street name, building number)
- **Customer_phones** (customer_id, phone)
- **Account** (number, status, balance, type_id)
- **Account type** (id, name, description)
- **Ownership** (customer_id, account_number)
- **Visit** (customer_id, branch_number, date, time)

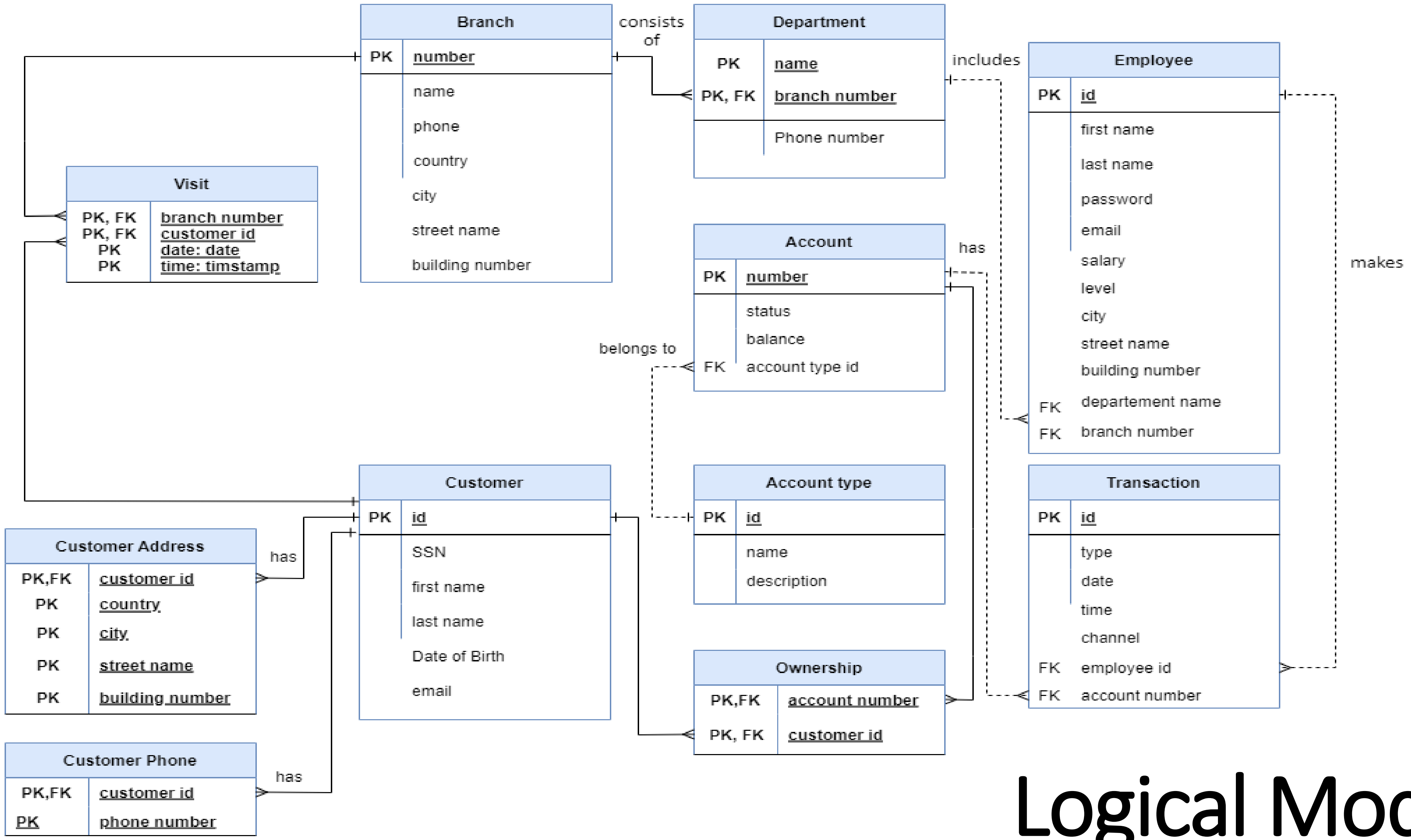
Note: Primary keys are underlined, and foreign keys are colored with red

Strong and Weak Entities

- **Branch** (number, name, phone, country, city, street name, building number)
- **Department** (name, branch_number, phone)
- **Employee** (id, fname, lname, password, email, salary, level, country, city, street name, building number, department_name, branch_number)
- **Transaction** (id, type, date, time, channel, employee_id, account_number)
- **Customer** (id, fname, lname, SSN, date of birth, email)
- **Customer_addresses** (customer_id,country, city, street name, building number)
- **Customer_phones** (customer_id, phone)
- **Account** (number, status, balance, type_id)
- **Account type** (id, name, description)
- **Ownership** (customer_id, account_number)
- **Visit** (customer_id, branch_number, date, time)

⬛ Strong Entities

🟦 Weak Entities



Logical Model

References

- Elmasri, R., & Navathe, S. (2017). Fundamentals of database systems (Vol. 7). Pearson
- Eng. Lina’s slides
- Dr. Raneem’s slides.