Transform – ER Model to Relational Model

Both ER-Model and Relational Model are abstract logical representations of real world enterprises. Because the two models imply similar design principles, we can convert ER design into Relational design. Converting a DB representation from an ER diagram to a table format is the way we arrive at Relational DB-design from an ER diagram.

ER diagram notations to relations:

1. **Strong Entity**
   - **1.**1. Becomes an individual table with entity name, attributes become columns of the relation.
   - **1.**2. Entity's Primary Key (PK) is used as Relation's PK.
   - **1.**3. Foreign Keys (FK) are added to establish relationships with other relations.
2. **Weak Entity**
   - **2.**1. A table is formed with all the attributes of the entity.
   - **2.**2. PK of its corresponding Strong Entity will be added as FK.
   - **2.**3. PK of the relation will be a composite PK, {FK + Partial discriminator Key}.
3. **Single Valued Attributes**
   - **3.**1. Represented as columns directly in the tables/relations.
4. **Composite Attributes**
   - **4.**1. Handled by creating a separate attribute itself in the original relation for each composite attribute.
   - **4.**2. *Example:* Address: {street-name, house-no} is a composite attribute in customer relation. We add `address-street-name` and `address-house-no` as new columns and ignore "address" as an attribute.
5. **Multivalued Attributes**
   - **5.**1. New tables (named as original attribute name) are created for each multivalued attribute.
   - **5.**2. PK of the entity is used as column FK in the new table.
   - **5.**3. Multivalued attribute's similar name is added as a column to define multiple values.
   - **5.**4. PK of the new table would be {FK + multivalued name}.
   - **5.**5. *Example:* For strong entity Employee, `dependent-name` is a multivalued attribute.

Prepared by: Anik Kirtania
Computer Science and Engineering,
University of Chittagong.

5.5.1. New table named `dependent-name` will be formed with columns `emp-id`, and `dname`.

5.5.2. PK: {emp-id, dname}

5.5.3. FK: {emp-id}

6. Derived Attributes: Not considered in the tables.

7. Generalisation

7.1. **Method-1:** Create a table for the higher level entity set. For each lower-level entity set, create a table that includes a column for each of the attributes of that entity set plus a column for each attribute of the primary key of the higher-level entity set.

- `account` (account-number, balance)
- `savings-account` (account-number, interest-rate, daily-withdrawal-limit)
- `current-account` (account-number, overdraft-amount, per-transaction-charges)

7.2. **Method-2:** If the generalisation is disjoint and complete (no entity is a member of two lower-level entity sets, and every entity in the higher level entity set is a member of one lower-level set), do not create a table for the higher-level entity set. For each lower-level entity set, create a table that includes all attributes of that set plus those of the higher-level set.

- `savings-account` (account-number, balance, interest-rate, daily-withdrawal-limit)
- `current-account` (account-number, balance, overdraft-amount, per-transaction-charges)

7.3. **Drawbacks of Method-2:** If used for overlapping generalisation, some values (like `balance`) would be stored twice unnecessarily. If the generalisation is not complete (some accounts are neither savings nor current accounts), such accounts cannot be represented with the second method.

8. Aggregation

8.1. Table of the relationship set is made.

8.2. Attributes include primary keys of entity set and aggregation set's entities.

8.3. Also, add descriptive attribute if any on the relationship.

9. Unary (Recursive) Relationships:

9.1. For 1:1 recursive relationships, add a foreign key in the same table that references the primary key of the same relation.

9.2. For 1:N recursive relationships, add a foreign key in the same table that allows multiple rows to reference a single primary key in the same relation.

9.3. For M:N recursive relationships, create a new relation with two foreign keys both referencing the primary key of the original entity, and use both as the composite primary key of the new table.