Logical Database Design and the Relational Model

Learning Objectives
This topic is intended to introduce the Logical Database Design and the Relational Model. At the end of the topic it is desired from the reader to be able to:

- Concisely define each of the following key terms:
  - Key or Primary Key
  - Foreign Key
  - Null Values
  - Entity Integrity Rule
  - Referential Integrity Constraint
  - Well-Structured Relation
  - Database Anomalies
  - Surrogate Primary Key
  - Recursive Foreign Key
  - Normalization
  - Normal Form
  - Functional Dependency
  - Determinant
  - Candidate Key
  - First Normal Form
  - Second Normal Form
  - Partial Functional Dependency
  - Third Normal Form
  - Transitive Dependency
  - Synonyms, Homonym and Alias
  - Enterprise Key

- List five properties of relations.
- State two essential properties of a candidate key.
- Give a concise definition of each of the following: first normal form, second normal form, and third normal form.
- Briefly describe four problems that may arise when merging relations.
- Transform an E-R (or EER) diagram into a logically equivalent set of relations.
- Create relational tables that incorporate entity integrity and referential integrity constraints.
- Use normalization to decompose a relation with anomalies into well-structured relations.

1. Introduction

Logical Database Design is the process of transforming the conceptual data model into a logical data model.

The logical data model is consistent and compatible with a specific type of database technology.

Conceptual data modeling is about understanding the organization—getting the right requirements. While logical database design is about creating stable database structures—correctly expressing the requirements in a technical language.

The objective of logical database design is to translate the conceptual design (which represents an organization’s requirements for data) into a logical database design that can be implemented by using a chosen database management system.

The resulting databases must meet the user needs for data sharing, flexibility, and ease of access.

An experienced database designer often will do logical database design in parallel with conceptual data modeling if he knows the type of database technology that will be used.

Although there are other data models, we have two reasons for emphasizing the relational data model.

- First, this model is the one most commonly used in contemporary database applications.
- Second, some of the principles of logical database design for the relational model apply to the other logical models as well.

The Relational Data Model is a form of logical data model, and as such it is different from the conceptual data models.

The E-R data model is not a relational data model, and an E-R model may not obey the rules for a well-structured relational data model, called normalization.

- The E-R model was developed for other purposes like understanding data requirements and business rules about the data—not structuring the data for sound database processing, which is the goal of logical database design.
The process of transforming an Entity Relational model into the relational model is supported by Many CASE tools today at the technical level; however, it is important that you understand the underlying principles and procedures.

Normalization is the process of designing well-structured relations. It is an important component of logical design for the relational model.

2. The Relational Data Model

The Relational Data Model was first introduced in 1970 by E. F. Codd.

In the beginning some early research projects were launched to prove the feasibility of the relational model and to develop prototype systems.

Commercial RDBMS products started to appear about 1980.

Today RDBMSs have become the dominant technology for database management, and there are literally hundreds of RDBMS products for computers ranging from smartphones and personal computers to mainframes.

- Basic Definitions
  - The relational data model represents data in the form of tables or Relations.
  - The Model is based on mathematical theory and therefore has a solid theoretical foundation. It consists of the following three components:
    - **Data Structure**: Data are organized in the form of tables, with rows and columns.
    - **Data Manipulation**: Powerful operations (using the SQL language) are used to manipulate data stored in the relations.
    - **Data Integrity**: The model includes mechanisms to specify business rules that maintain the integrity of data when they are manipulated.

- Relational Data Structure
  - A Relation is a named, two-dimensional table of data. An Attribute is a named column of a relation.
  - We can express the structure of a relation by using its name followed (in parentheses) by the names of the attributes in that relation.
  - Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows (records).
  - Each row of a relation corresponds to a record that contains data (attribute) values for a single entity.
  - Addition or Deletion or records (rows) does not change the relation. In fact, we could delete all of the rows of a relation without losing its structure.

- Relational Keys
  - We must be able to store and retrieve a row of data in a relation, based on the data values stored in that row. To achieve this goal, every relation must have a primary key.
  - A Primary Key is an attribute or a combination of attributes that uniquely identifies each row in a relation. We designate a primary key by underlining the attribute name(s).
  - The concept of a primary key is related to the term identifier. The same attribute or a collection of attributes indicated as an entity’s identifier in an E-R diagram may be the same attributes that compose the primary key for the relation representing that entity.
  - Associative entities do not have to have an identifier, and the (partial) identifier of a weak entity forms only part of a weak entity’s primary key.
  - There may be several attributes of an entity that may serve as the associated relation’s primary key.
  - A Composite Key is a primary key that consists of more than one attribute. For example, the primary key for a relation DEPENDENT would likely consist of the combination EmpID and DependentName.
Often we need to represent the relationship between two tables or relations. This is accomplished through the use of foreign keys.

A **Foreign Key** is an attribute (possibly composite) in a relation that serves as the primary key of another relation.

### Properties Of Relations

Relations have following properties that distinguish them from non-relational tables.

- Each relation (or table) in a database has a **unique name**.
- There can be only one value associated with each attribute on a specific row of a table; no multivalued attributes are allowed in a relation. Alternatively it can be said that an entry at the intersection of each row and column is **Atomic (Or Single Valued)**.
- Each row is unique; no two rows in a relation can be identical.
- Each attribute (or column) within a table has a **unique name**.
- The **sequence of columns (left to right) is insignificant**. The order of the columns in a relation can be changed without changing the meaning or use of the relation.
- The **sequence of rows (top to bottom) is insignificant**. As with columns, the order of the rows of a relation may be changed or stored in any sequence.

### Database Schema

A relational database may consist of any number of relations. The structure of the database is described through the use of a **Schema**, which is a description of the overall logical structure of the database.

There are two common methods for expressing a schema:

- Short text statements, in which each relation is named and the names of its attributes follow in parentheses.
- A graphical representation, in which each relation is represented by a rectangle containing the attributes for the relation.

Text statements have the advantage of simplicity. However, a graphical representation provides a better means of expressing referential integrity constraints.

| CUSTOMER(CustomerID, CustomerName, CustomerAddress, CustomerCity, CustomerState, CustomerPostalCode) |
| ORDER(OrderID, OrderDate, CustomerID) |
| ORDER LINE(OrderID, ProductID, OrderedQuantity) |
| PRODUCT(ProductID, ProductDescription, ProductFinish, ProductStandardPrice, ProductLineID) |

A schema for four relations at Pine Valley Furniture Company

It is a good idea to create an instance of your relational schema with sample data for four reasons:

- The sample data allow you to test your assumptions regarding the design.
- The sample data provide a convenient way to check the accuracy of your design.
- The sample data help improve communications with users in discussing your design.
- You can use the sample data to develop prototype applications and to test queries.

### Integrity Constraints

The relational data model has several types of constraints or rules to ensure the database integrity.

- These constraints control the acceptable data values and actions on the data.
- They also facilitates the DBMS to maintain accuracy and integrity of the data in the database.
- The major types of integrity constraints are Domain Constraints, Entity Integrity, and Referential Integrity.
a) Domain Constraints
   - A domain is the set of values that may be assigned to an attribute.
   - All of the values that appear in a column of a relation must be from the same domain.
   - A domain definition usually consists of the following components: domain name, meaning, data type, size (or length), and allowable values or allowable range (if applicable).

b) Entity Integrity
   - The entity integrity rule is designed to ensure that every relation has a primary key and that the data values for that primary key are all valid. In particular, it guarantees that every primary key attribute is non-null.
   - In some cases, a particular attribute cannot be assigned a data value. There are two situations in which this is likely to occur: Either there is no applicable data value or the applicable data value is not known when values are assigned.
   - The relational data model allows us to assign a null value to an attribute in the just described situations.
   - A Null is a value that may be assigned to an attribute when no other value applies or when the applicable value is unknown. In reality, a null is not a value but rather it indicates the absence of a value.
   - For example, it is not the same as a numeric zero or a string of blanks. The inclusion of nulls in the relational model is somewhat controversial, because it sometimes leads to anomalous results (Date, 2003).
   - Entity integrity rule states that No primary key attribute (or component of a primary key attribute) may be null.

c) Referential Integrity
   - In the relational data model, associations between tables are defined through the use of foreign keys.
   - A Referential Integrity Constraint is a rule that maintains consistency among the rows of two relations. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in another relation or the foreign key value must be null.
   - A referential integrity constraint must be defined for each of these arrows in the schema. If the relationship is optional, then the foreign key could be null. Whether a foreign key can be null must be specified as a property of the foreign key attribute when the database is defined.
   - Actually, whether a foreign key can be null is more complex to model on an E-R diagram and to determine than we have shown so far. For example, what happens to Referential Data if we choose to delete a Primary Key. Three choices are possible:
     a) Delete the associated Records
     b) Prohibit deletion of the Primary Key until all Referential Data are first deleted
     c) Place a null value in the foreign key and keep the Referential Data.
   - In practice, organizational rules and various regulations regarding data retention determine what data can be deleted and when, and they therefore govern the choice between various deletion options.

Well-Structured Relations
- A well-structured relation contains minimal redundancy and allows users to insert, modify, and delete the rows in a table without errors or inconsistencies.
- Redundancies in a table may result in errors or inconsistencies (called anomalies) when a user attempts to update the data in the table.
- Anomaly is an error or inconsistency that may result when a user attempts to update a table that contains redundant data. The three types of anomalies are insertion, deletion, and modification anomalies.

a) Insertion anomaly
- A relation has insertion anomaly if does not allow insertion of some of the attributes of a relation unless the values for some specific or all attributes are provided.
A Relation having a **Composite Primary Key** does not allow insertion of any other attribute unless all parts of the composite key are inserted. A composite key having combination of **Emp_ID** and **Course_Title** requires that the user must supply values for both Emp_ID and Course_Title to insert any other attribute of a record.

b) **Deletion anomaly**

A relation has the deletion anomaly if the deletion operation on it causes deletion of some required attributes.

For example deletion of a record from the “Employee” table may cause deletion of the department information.

c) **Modification anomaly**

A relation has the modification anomaly if the modification or update operation results in inconsistency in some other records in the relation.

3. **Transforming EER Diagrams Into Relations**

The logical design transforms the E-R (and EER) diagrams that were developed during conceptual design into relational database schemas.

The inputs to this process are the Entity-Relationship (and Enhanced E-R) diagrams while the outputs are the relational schemas.

Transforming the EER diagrams into relations is a straightforward process with a well-defined set of rules.

Database designer can use the CASE tools to perform many of the conversion steps. Nevertheless it is required to have understanding of the conversion process because of the following reasons:

- CASE tools often cannot model more complex data relationships such as ternary relationships and supertype/subtype relationships. In these situations, you may have to perform the steps manually.
- There are sometimes legitimate alternatives where you will need to choose a particular solution.
- You must be prepared to perform a quality check on the results obtained with a CASE tool.
- Understanding the transformation process helps you understand why conceptual data modeling (modeling the real-world domain).

### Mapping the Regular Entities

**Regular Entities** are entities that have an independent existence and generally represent real-world objects, such as persons and products. Each regular entity type in an E-R diagram is transformed into a relation.

Regular entity types are represented by **rectangles with a single line**.

The name given to the relation is generally the same as the entity type.

Each simple attribute of the entity type becomes an attribute of the relation.

The identifier of the entity type becomes the primary key of the corresponding relation.

### Mapping the Composite Attributes:

When a regular entity type has a **Composite Attribute**, we split it into its simple components and include each component separately in the relation.

### Mapping the Multivalued Attribute

When the regular entity type contains a **Multivalued Attribute**, two new relations (rather than one) are created.

- The first relation contains all of the attributes of the entity type except the multivalued attribute.
- The second relation contains two attributes that form the primary key of the second relation.

The first of these attributes is the primary key from the first relation, which becomes a foreign key in the second relation.

If an entity type contains multiple multivalued attributes, each of them will be converted to a separate relation.

### Mapping the Weak Entities

The weak entity type does not have an independent existence but exists only through an identifying relationship with another entity type called the owner. Weak entities are identified by a **rectangle with a double line**.
A weak entity type does not have a complete identifier but must have an attribute called a **partial identifier** that permits distinguishing the various occurrences of the weak entity for each owner entity instance.

For each weak entity type, create a new relation and include all of the simple attributes (or simple components of composite attributes) as attributes of this relation.

Then **include the primary key of the identifying relation as a foreign key** attribute in this new relation. The primary key of the new relation is the combination of this primary key of the identifying and the partial identifier of the weak entity type.

An alternative approach to simplify the primary key of the DEPENDENT relation is use of the surrogate key.

**Surrogate Primary Key**

It is a serial number or other system assigned primary key for a relation.

A surrogate key is usually created to simplify the key structures. It is recommended to create a surrogate key when any of the following conditions hold:

- There is a composite primary key.
- The natural primary key (i.e., the key used in the organization and identified in conceptual data modeling as the identifier) is inefficient (e.g., it may be very long)
- The natural primary key is recycled (i.e., the key is reused or repeated periodically)

Whenever a surrogate key is created, the natural key is always kept as non-key data in the same relation because the natural key has organizational meaning.

In fact, surrogate keys mean nothing to the end users, so they are usually never displayed; rather, the natural keys are shown to the user as the primary keys and used as identifiers in searches.

**Map Binary Relationships**

The procedure for representing relationships depends on both the degree of the relationships (unary, binary, or ternary) and the cardinalities of the relationships.

**To map a Binary: One to Many (1:M) relationship:**

- First create a relation for each of the two entity types participating in the relationship.
- Next, include the primary key attribute(s) of the entity on the one-side of the relationship as a foreign key in the relation that is on the many-side of the relationship. (The primary key migrates to the many side.)

**To map a Binary: Many to Many (M:N) relationship:**

- Suppose that there is a binary many-to-many (M: N) relationship between two entity types, A and B.
- Create a new relation, C. Include as foreign key attributes in C the primary key for each of the two participating entity types.
- These attributes together become the primary key of C. Any non-key attributes that are associated with the relationship are included in the relation C.

**To map a Binary: One to One (1:1) relationship:**

- First create a relation for each of the two entity types participating in the relationship.
- Second, the primary key of one of the relations is included as a foreign key in the other relation.
- In a 1:1 relationship, the association in one direction is nearly always an optional one, whereas the association in the other direction is mandatory one.
- Include in the relation on the optional side of the relationship the foreign key of the entity type that has the mandatory participation in the 1:1 relationship. This approach will prevent the need to store null values in the foreign key attribute.
- It is also recommended to use the Primary key of the relation having more durable records, as the foreign key in the other relation.
Map Associative Entities (also called Gerunds)
- The Associative Entities are formed from many-to-many relationships between other entity types.
- In ER-Diagram these entities are represented by a rectangle with rounded corners.
- Mapping the associative entity involves essentially the same steps as mapping an M:N relationship.
  - First, create three relations: one for each of the two participating entity types and a third relation (associative relation) for the associative entity.
  - Second step then depends on whether on the E-R diagram an identifier was assigned to the associative entity.
    - If an identifier was not assigned, the default primary key for the associative relation consists of the two primary key attributes from the other two relations. These attributes are then foreign keys that reference the other two relations.
    - If an identifier was assigned, the primary key for this relation is the identifier assigned on the E-R diagram (rather than the default key). The primary keys for the two participating entity types are then included as foreign keys in the associative relation.

Map Unary Relationships
- A unary relationship as a relationship between the instances of a single entity type. Unary relationships are also called Recursive Relationships.
- The two most important cases of unary relationships are one-to-many and many-to-many relationships.

Mapping the Unary One-To-Many Relationships
- A relation is created to map the entity type in the unary relationship.
- Then a foreign key attribute is added to the same relation; this attribute references the primary key values in the same relation. (This foreign key must have the same domain as the primary key.) This type of a foreign key is called a Recursive Foreign Key.

Unary Many-To-Many Relationships
- With this type of relationship, two relations are created: one to represent the entity type in the relationship and an associative relation to represent the M:N relationship itself.
- The primary key of the associative relation consists of two attributes. These attributes (which need not have the same name) both take their values from the primary key of the other relation. Any non-key attribute of the relationship is included in the associative relation.

Map Ternary (and N-Ary) Relationships
- A ternary relationship is a relationship among three entity types.
- It is recommended to convert a ternary relationship to an associative entity to represent participation constraints more accurately.
- To map an associative entity type that links three regular entity types:
  - We create a new associative relation.
  - The default primary key of this relation consists of the three primary key attributes for the participating entity types. (In some cases, additional attributes are required to form a unique primary key.) These attributes then act in the role of foreign keys that reference the individual primary keys of the participating entity types.
  - Any attributes of the associative entity type become attributes of the new relation.

Map Supertype/Subtype Relationships
- The relational data model does not directly support supertype/subtype relationships. So various strategies are used to represent these relationships. A commonly used strategy is following:
  - Create a separate relation for the supertype and for each of its subtypes.
f) Assign to the relation created for the supertype the attributes that are common to all members of the supertype, including the primary key.

g) Assign to the relation for each subtype the primary key of the supertype and only those attributes that are unique to that subtype.

h) Assign one (or more) attributes of the supertype to function as the subtype discriminator.

<table>
<thead>
<tr>
<th>EER Structure</th>
<th>Relational Representation (Sample Figure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular entity</td>
<td>Create a relation with primary key and nonkey attributes (Figure 4-8)</td>
</tr>
<tr>
<td>Composite attribute</td>
<td>Each component of a composite attribute becomes a separate attribute in the target relation (Figure 4-9)</td>
</tr>
<tr>
<td>Multivalued attribute</td>
<td>Create a separate relation for multivalued attribute with composite primary key, including the primary key of the entity (Figure 4-10)</td>
</tr>
<tr>
<td>Weak entity</td>
<td>Create a relation with a composite primary key (which includes the primary key of the entity on which this entity depends) and nonkey attributes (Figure 4-11)</td>
</tr>
<tr>
<td>Binary or unary 1:N relationship</td>
<td>Place the primary key of the entity on the one side of the relationship as a foreign key in the relation for the entity on the many side (Figure 4-12; Figure 4-17 for unary relationship)</td>
</tr>
<tr>
<td>Binary or unary M:N relationship or associative entity without its own key</td>
<td>Create a relation with a composite primary key using the primary keys of the related entities plus any nonkey attributes of the relationship or associative entity (Figure 4-13; Figure 4-15 for associative entity; Figure 4-18 for unary relationship)</td>
</tr>
<tr>
<td>Binary or unary 1:1 relationship</td>
<td>Place the primary key of either entity in the relation for the other entity; if one side of the relationship is optional, place the foreign key of the entity on the mandatory side in the relation for the entity on the optional side (Figure 4-14)</td>
</tr>
<tr>
<td>Binary or unary M:N relationship or associative entity with its own key</td>
<td>Create a relation with the primary key associated with the associative entity plus any nonkey attributes of the associative entity and the primary keys of the related entities as foreign keys (Figure 4-16)</td>
</tr>
<tr>
<td>Ternary and n-ary relationships</td>
<td>Same as binary M:N relationships above; without its own key, include as part of primary key of relation for the relationship or associative entity the primary keys from all related entities; with its own surrogate key, the primary keys of the associated entities are included as foreign keys in the relation for the relationship or associative entity (Figure 4-19)</td>
</tr>
<tr>
<td>Supertype/subtype relationship</td>
<td>Create a relation for the superclass, which contains the primary and all nonkey attributes in common with all subclasses, plus create a separate relation for each subclass with the same primary key (with the same or local name) but with only the nonkey attributes related to that subclass (Figure 4-20 and 4-21)</td>
</tr>
</tbody>
</table>

4. Introduction to Normalization

→ Normalization is the process of successively reducing relations with anomalies to produce smaller, well-structured relations.

→ Normalization is a formal process for deciding which attributes should be grouped together in a relation so that all the database anomalies are removed.

→ The Normalization is a state of a relation that requires that certain rules regarding relationships between attributes (or functional dependencies) are satisfied. We need formal definitions of such relations, together with a process for designing them.

→ Two Major Occasions during the overall database development process to benefit from the normalization:
  a) During logical database design you should use normalization concepts as a quality check for the relations that are obtained from mapping E-R diagrams.
b) When reverse-engineering older systems many of the tables and user views for older systems are redundant and subject to the anomalies.

- **Goals of the Database Normalization**
  a) Minimize data redundancy, thereby avoiding anomalies and saving storage space
  b) Simplify the enforcement of referential integrity constraints
  c) Make it easier to maintain data (insert, update, and delete)
  d) Provide a better design that is an improved representation of the real world and a stronger basis for future growth

- Normalization makes no assumptions about how data will be used in displays, queries, or reports.
- It is based on what we will call normal forms and functional dependencies, defines rules of the business, not data usage.
- Further, remember that data are normalized by the end of logical database design. Thus, normalization places no constraints on how data can or should be physically stored or, therefore, on processing performance.
- Normalization is a logical data modeling technique used to ensure that data are well structured from an organization-wide view.

- **Steps in Normalization**
  - Normalization can be accomplished and understood in stages (Normal forms).
  - A normal form is a state of a relation that requires that certain rules regarding relationships between attributes (or functional dependencies) are satisfied.

  1. **First Normal Form (1NF)**
     - A relation is in First Normal Form if:
       - It does not contain the repeating groups of attributes in it.
  2. **Second Normal Form (2NF)**
     - A relation is in Second Normal Form if:
       - It is in First Normal Form AND
       - Any partial functional dependencies have been removed (i.e., non-key attributes are identified by the whole primary key).
  3. **Third Normal Form (3NF)**
     - A relation is in Third normal form if:
       - It is in Second Normal Form AND
       - Any transitive dependencies have been removed (i.e. non-key attributes are identified by only the primary key).
  4. **Boyce-Codd Normal Form (BCNF)**
     - A relation R, is in Boyce-Codd Normal Form if:
       - It is in Third Normal Form AND
       - For every FD \( X \rightarrow Y \) in the Relation R, X is a **Super Key**.
  5. **Fourth Normal Form (4NF)**
     - A relation R, is in Fourth Normal Form if:
       - It is in Boyce-Codd Normal Form (BCNF) AND
       - Any Multivalued Dependencies have been removed.
  6. **Fifth Normal Form (5NF)**
     - A relation R, is in Boyce-Codd Normal Form (BCNF) if:
       - It is in Third Normal Form AND
       - There is no Projection Join-dependency (PJD) in it.
- **Functional Dependencies and Keys**
  - Up to the Boyce-Codd normal form, normalization is based on the analysis of functional dependencies.
  - A functional dependency is a constraint between two attributes or two sets of attributes.
  - For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B.
  - The functional dependency of B on A is represented by an arrow: A → B.
  - A functional dependency is not a mathematical dependency: B cannot be computed from A. Rather, if you know the value of A, there can be only one value for B.
  - An attribute may be functionally dependent on a combination of two (or more) attributes rather than on a single attribute.

  **For example:** EmpID, Course_Title → Date_Completed
  - The functional dependency in this statement implies that the date of a course completion is determined by the identity of the employee and the title of the course.
  - Typical examples of functional dependencies are the following:
    - **CNIC_No. → Name, Address, Birthdate** A person’s name, address, and birth date are functionally dependent on that person’s Social Security number (in other words, there can be only one Name, one Address, and one Birthdate for each SSN).
    - **VIN → Make, Model, Color** The make, model, and color of a vehicle are functionally dependent on the vehicle identification number (there can be only one value of Make, Model, and Color associated with each VIN).
    - **ISBN → Title, FirstAuthorName, Publisher** The title of a book, the name of the first author, and the publisher are functionally dependent on the book’s international standard book number (ISBN).

- **Steps in normalization**

  ![Steps in normalization diagram](image-url)
Determinants
→ The attribute on the left side of the arrow in a functional dependency is called a determinant. SSN, VIN, and ISBN are determinants (respectively) in the preceding three examples. In the EMP COURSE relation, the combination of EmpID and CourseTitle is a determinant.

Candidate Keys
→ A candidate key is an attribute, or combination of attributes, that uniquely identifies a row in a relation.
→ A candidate key must satisfy the following properties, which are a subset of the six properties of a relation previously listed:
  1. Unique Identification: For every row, the value of the key must uniquely identify that row. This property implies that each non-key attribute is functionally dependent on that key.
  2. Non Redundant: No attribute in the key can be deleted without destroying the property of unique identification.
→ A candidate key is always a determinant, whereas a determinant may or may not be a candidate key.
→ A candidate key is a determinant that uniquely identifies the remaining (non-key) attributes in a relation.
→ A determinant may be a candidate key, part of a composite candidate key, or a non-key attribute.

5. Achieving the Normalization Forms
→ We are required to take multiple steps to achieve the various normal forms.
  ▪ Convert to First Normal Form (1NF)
→ A relation is in first normal form (1NF) if it satisfies the following two constraints:
  1. The relation has a Primary Key, which uniquely identifies each row in the relation.
  2. There are No Repeating Groups in the relation. So, there is a single fact at the intersection of each row and column of the table.
→ Remember that removal of the repeating groups does not ensure a well-structured relation. So the relation in 1NF may have Insertion, Deletion and the Update anomalies.

Anomalies in First Normal Form (1NF)
1) Insertion Anomaly
→ A relation has “Insertion Anomaly”
  ✓ If it is not possible to insert some attributes of a record in it without providing the some other attributes of the record. OR
  ✓ If insertion of some of the attributes of the relation requires us to insert repeatedly, some other attributes of the same record. This leads to data replication and potential data entry errors.

2) Deletion Anomaly
→ A relation has “Deletion Anomaly”
  ✓ If it is not possible to delete some attributes of a record in it without “Losing” some other attributes of the record, those are required.

3) Update Anomaly
→ A relation has “Update Anomaly”
  ✓ If update in a single record ( or row ) causes update in some other rows of the relation.
  ▪ Convert to Second Normal Form
→ A relation is in second normal form (2NF) if it is in first normal form and contains no partial functional dependencies.
→ A partial functional dependency exists when a non-key attribute is functionally dependent on part (but not all) of the primary key.
→ To convert a relation with partial dependencies to second normal form, the following steps are required:
1. Create a new relation for each primary key attribute (or combination of attributes) that is a determinant in a partial dependency. That attribute is the primary key in the new relation.
2. Move the non-key attributes that are dependent on this primary key attribute (or attributes) from the old relation to the new relation.

→ Removal of the partial dependencies results in the formation of multiple new relations:
→ Remember that a relation that is in first normal form will be in second normal form if any one of the following conditions applies:
   1. The primary key consists of only one attribute (it is not composite). By definition, there cannot be a partial dependency in such a relation.
   2. No non-key attributes exist in the relation (thus all of the attributes in the relation are components of the primary key). There are no functional dependencies in such a relation.
   3. Every non-key attribute is functionally dependent on the full set of primary key attributes.

### Convert to Third Normal Form
→ A relation is in third normal form (3NF) if it is in second normal form and no transitive dependencies exist.
→ A **Transitive Dependency** in a relation is a functional dependency between the primary key and one or more non-key attributes that are dependent on the primary key via another non-key attribute.
→ Transitive dependencies create unnecessary redundancy that may lead to the database anomalies.

#### Removing Transitive Dependencies
→ Three-step procedure to remove transitive dependencies from a relation:
   1. For each non-key attribute (or set of attributes) that is a determinant in a relation, create a new relation. That attribute (or set of attributes) becomes the primary key of the new relation.
   2. Move all of the attributes that are functionally dependent on the primary key of the new relation from the old to the new relation.
   3. Leave the attribute that serves as a primary key in the new relation in the old relation to serve as a foreign key that allows you to associate the two relations.

### 6. Merging Relations
→ We transform EER diagrams into relations when we take the results of a top-down analysis of data requirements and begin to structure them for implementation in a database.
→ We apply the normalization rules (third or higher normal form) on the relations to get a database having the well-structured relations.
→ As part of the logical design process, normalized relations may have been created from a number of separate EER diagrams and (possibly) other user views (i.e., there may be bottom-up or parallel database development activities for different areas of the organization as well as top-down ones).
→ The three-schema architecture for databases encourages the simultaneous use of both top-down and bottom-up database development processes.
→ In reality, most medium to large organizations have many reasonably independent systems development activities that at some point need to come together to create a shared database.
→ The result is that some of the relations generated from these various processes may be redundant; that is, they may refer to the same entities. In such cases, we should merge those relations to remove the redundancy. This section describes merging relations (also called view integration).

#### Reasons of merging the relations:
1) Sometime more than one teams (sub-teams) work on a single large project; when the output of their Logical Database Design comes together, there is often a need to the merge relations.
2) **Integrating existing databases with new information requirements** often leads to the need to integrate different relations or views.

3) **New data requirements** may arise during the life cycle, so there is a need to merge any new relations with what has already been developed.

**View Integration Problems**

→ While merging or integrating relations a database analyst must understand the meaning of the data and must be prepared to resolve any problems that may arise in that process.

→ Some of the problems that arise in view integration are:
  
  A) Synonyms,  
  B) Homonyms  
  C) Transitive Dependencies  
  D) Supertype/Subtype Relationships

**Synonyms**

→ In some situations, two (or more) attributes may have different names but the same meaning (e.g., when they describe the same characteristic of an entity). Such attributes are called synonyms. For example, Employee_ID and Employee_No may be synonyms.

→ When merging the relations that contain synonyms, you should obtain agreement (if possible) from users on a single, standardized name for the attribute and eliminate any other synonyms.

→ Another alternative is to choose a third name to replace the synonyms.

→ When there are synonyms, there is a need to allow some database users to refer to the same data by different names. Users may need to use familiar names that are consistent with terminology in their part of the organization.

→ An **Alias** is an alternative name used for an attribute. Many DBMS allow the definition of an alias that may be used interchangeably with the primary attribute label.

**Homonyms**

→ An attribute name that may have more than one meaning is called a homonym.

→ For example, the term account might refer to a bank’s checking account, savings account, loan account, or other type of account (and therefore account refers to different data, depending on how it is used).

**Transitive Dependencies**

→ When two 3NF relations are merged to form a single relation, transitive dependencies may result. The analyst should create 3NF relations by removing the transitive dependency.

**Supertype/Subtype Relationships**

→ These relationships may be hidden in relations or the user views. The database analyst should recognize and preserve the Super type/Sub type relationship while merging the relations.

**Enterprise Key**

→ A primary key that is **unique across the whole database** not just within the relation or table to which it applies, is called an **Enterprise Key**.

→ This criterion makes a primary key more like what in object oriented databases is called an object identifier.

→ With this recommendation, the primary key of a relation becomes a value internal to the database system and has no business meaning.

→ One of the main motivations for using an enterprise key is database evolve-ability, in which the existing relations are merged with the new relations after the database is created.