CHAPTER IV

VIEW INTEGRATION

4.1. INTRODUCTION

View integration is an important phase of logical database design and is also necessary to define a global schema that describes all the data in several existing databases participating in a distributed database management system. In logical database design, each class of user designs a view of a part of a database that they need to access. The objective is to produce a single global view known as the conceptual schema by integrating all of these individual user views, that represent the entire content of the database application.

The objective of the present study is to develop a computer-aided methodology for view integration. The proposed methodology is based on an extended version of the entity relationship (EER) model and covers all the stages of view integration process as suggested by Batini et al. [15] in their survey work on schema integration. Another aspect of the purposed methodology is that it facilitates the design of an expert system VIS (view Integration System) which is a part of the expert system VMIT.
The four main stages of view integration are:

1) Pre-integration stage: during which decisions are made regarding the order of integration of the user views and the type of interschema properties to be used in the integration process.

2) Conflict analysis and resolution stage: in which the various conflicts such as naming conflicts, structural conflicts, etc. are identified and resolved. The interschema properties between the views are also identified during this stage.

3) Merging stage: during which the conflict-free views are merged into a single view by superimposition of common concepts.

4) Restructuring involves enrichment of the merged view by adding the interschema properties, new objects, hierarchical links and relationships; and eliminating redundant hierarchical links and relationships from it.

The objective of this stage is to achieve a well-formed EER schema to make the final integrated view coherent with the representation of data chosen. The view integration strategies proposed by various authors can be broadly classified into the following two types:

1) Binary Strategy: This allows the integration of two views at a time. They are called ladder strategies when
a new component (individual user) view is integrated with an existing intermediate view resulting at each step. A binary-strategy is balanced when the views are divided into pairs at the start and are integrated in a systematic manner.

2) N-ary-Strategy: These strategies allow integration of n views (n > 2) at a time. An n-any-strategy is one-slot when all the n-views are integrated in a single step, iterative otherwise.

The view integration strategy implemented in the present expert system VMITS is of the first type, that is, each time two views are selected by the designer are integrated to produce an integrated view. At each integration step, the designer can decide to integrate any two views, whether the two are user (original) views, or one is a user view and the other is an intermediate result, or both are intermediate results. This approach is more flexible than both the ladder and balanced binary integration strategies.

4.2. REVIEW OF VIEW INTEGRATION METHODOLOGIES

Several methodologies have been proposed for view integration for both the purposes of conceptual schema design and global view design. Batini et al. [15] surveys twelve of such methodologies for both purposes. The most significant of
these are described in the remainder of this section.

One of the earliest significant work is that of Motro and Buneman [98]. They defined a collection of schema operators and several merging algorithms based on these operators. Their schema operators include the addition and deletion of subtypes, and attributes. Their automatic merge algorithm perform the merge without any user interaction, while their cooperative merge algorithm performs the merge with the user interaction at each step. Their work is important because it is the first substantial work on global view design and their method is supported by computer based system. However their results are limited because they neglected the problems of data incompatibility and partial overlapping databases which are very common to heterogeneous databases. Their merging algorithms only address merging of entities and not generalization hierarchies.

Navathe et al. [101,102,103,104,105] in a series of papers proposed view integration methodologies employing an extended entity-relationship model ECR, and they provide algorithmic solutions to integration problems. They present a small number of rules to merge entities and relationships and general algorithms to order the relationships and entities for merging. Larson et al [82] have discussed different types of attribute equivalences and how they are to
be treated during entity and relationship merging. The limitation of their approach are:

1) They have not discussed the conflict analysis stage in detail.

2) The whole process is manual and puts a tremendous pressure on the designer who has to trace all the attributes of entities and relationships in all schemes and check equivalences by following certain criteria.

3) The methodology is not supported by computer based system, and the algorithms are not specific enough for implementation.

Dogac and Spaccapietre [47] propose an expert system VDIES for view integration. The methodology is based on an extended entity-relationship model (EER) and employs a binary-ladder strategy. Their results are limited because:

1) Although they employee the EER model they do not allow general relationships (Relationships with degree > 2) and generalized hierarchies for source schemes while the target schema allows the generalization hierarchies, which limits the applicability for the system and also during the integration process if at any stage an intermediate integrated schema resulting from integration of two source schemes, contain any generalized hierarchy.
then it can not be further integrated with any other scheme.

2) Although the system does some analyses and resolve some conflicts such as Naming and type conflicts, it does so by using a knowledge base which should be specified as a set of integrating assertions by the designer and the methodology does not specify the algorithms for merging entities and relationships.

The most recent work on view integration is the BRIM (Binary Relationship Integration Methodology) by Shoval and Zohn [118] which utilizes a Binary-Relationship model and use the Binary-Integration strategy. The methodology cover all the stages of view integration process. During pre-integration stage, it allows the designer to prepare a plan of the integration step. During conflict resolution stage, it identifies for the user most of the conflicts and systematically assists the designer to identity conflicts which cannot be identified automatically. During merging stage the two source schemes are merged by superimposition of common concepts. The methodology is implemented as a part of a PC-based expert system for database design. Some of the limitations of this approach are:
1) It utilizes a Binary-Relationship model which puts a restriction on the applicability of the system.

2) It has not described the algorithm for object and relationship integration.

3) Some of the conflicts such as attribute-entity type conflicts and role conflicts are not analyzed in it.

Our approach compensates these deficiencies by allowing generalization hierarchies and full aggregation capabilities and it resolves all types of conflicts. Also our system contain more heuristics to automatically identify and resolve some of the conflicts, such as synonyms. Our methodology describes the algorithms for entity and relationship merging and these algorithms are implemented in our expert system.

4.3. VIEW INTEGRATION METHODOLOGY

The view integration methodology described in this work is based on the binary integration strategy. Each time two views are selected by the user are integrated to produce an intermediate integrated view. The strategy does not distinguish between the original user view and the intermediate integrated view for selection, and it covers all the major stages of view integration proposed by Batini et al. [15] and other researchers.
The objective of the methodology is to extend previous research on view integration for database design by providing the followings:

1) Rules and heuristics for identifying and resolving various conflicts between pair of views.
2) Rules for identifying interschema properties.
3) Rules for matching and integrating entities.
4) Rules for relationship integration based on edge merging.

Finally the methodology is explained in a step-by-step manner to facilitate the development of an automatic tool for database design (such as the present expert system VMITS).

The steps for the view integration methodology are shown in Figure 4.1.

The notations used in the following discussions are:

- \( \text{Obj(A)} \) : represents the attribute A of the object Obj.
- \( \text{sub(E1,E2)} \) : represent that E1 is a subset entity of the generic entity E2 in the ISA hierarchy such as \( \text{E1 ISA E2} \).
- \( \text{mean(C)} \) : represents the meaning and value set of concept C.
- \( \text{is(E1,E2,V)} \) : E1 is a subset entity of E2 in view V.
FIGURE 4.1 VIEW INTEGRATION METHODOLOGY
The input to this procedure is a list of user view and the final output is a single view represented in the EER model and forms the conceptual schema of the required database application.

4.3.1. View Selection

The first step in the view integration is to decide which view are to be integrated in which order. The system lists the set of user/intermediate integrated views and asks the designer/analyst to select any two views for integration. The views selected are loaded into the fact base of the system for further processing. Let us call the two views selected in this step as view-1 and view-2.

4.3.2. Naming Conflict Resolution

During this step, names of concepts of the pair of views are compared and analyzed to discover homonyms and synonyms. For each type of these conflicts different cases are considered according to different possible pairs of basic concepts in the EER model, i.e. Entity-Entity, Relationship-Relationship, Entity-Relationship, Attribute-Attribute, Attribute-Entity and Attribute-Relationship. Some of the cases are further categorized depending upon the hierarchical links between entities, type of attributes (key or non-key), and the type of relationships. For each case and category, the problem is described, rules and heuristics are defined, and the resolution is described and illustrated with examples.
4.3.2.1. Homonym Conflicts

Homonym conflicts arise when different concepts in the two views have same name but have different meanings in the two views. They are easy to detect and are resolved by renaming the concepts in one or both the views. The following three type of homonym conflicts are distinguished :

a. Homonym conflict (HC_1) : Entity_Entity

Description : Two entities have same name but have different meanings in the two views.

Definition :
If : (E_i = E_j) and mean(E_i) <> mean(E_j).
Then : rename E_i or E_j or both and add interschema properties according to the following cases :

Case_1. E_i is a subset of E_j .
Solution : Rename E_i to NE (New entity name) and assert that "NE ISA E_j".
Example : (See Figure 4.2a)
Let the entity EMPLOYEE of View_1 refers to only the employee of managerial cadre whereas it refers to all the employees of the organization in View_2. The resolution is to rename the entity EMPLOYEE of view_1 as MANAGER and the interschema properties "MANAGER ISA EMPLOYEE" is asserted.
Interschema Property: Manager is an employee

Figure 4.2a Homonym conflict HC-1

Interschema Properties:
1. Manager is an employee
2. Engineer is an employee

Figure 4.2b Homonym CONFLICT HC-1
Case-2 : \( E_j \) is a subset of \( E_i \)
Solution : Rename \( E_j \) by \( NE \) and add "NE ISA \( E_j \)" to the fact base.

Case-3 : Both \( E_i \) and \( E_j \) are subsets of a common entity.
Solution : Rename \( E_i \) by \( NE_1 \) and \( E_j \) by \( NE_2 \) and add "NE_1 ISA E" and "NE_2 ISA E" to the fact base.

Example :

Let the entity EMPLOYEE of view_1 refers to managerial cadre while they refer to engineer cadre in view_2. The solution is to rename EMPLOYEE by MANAGER in view_1 and EMPLOYEE by ENGINEER in view_2 and the properties "MANAGER ISA EMPLOYEE" and "ENGINEER ISA EMPLOYEE" are asserted (see Figure 4.2b).

Case 4 : The entities, \( E_i \) and \( E_j \) are completely different.
Solution : No interschema properties are asserted in this case.

b. Homonym conflict (HC_2) : Relationship_relationship.
Description : Two relationships have same name but have different meanings in the two views.

Definition :
If \( (R_i=R_j) \) and mean (\( R_i \)) \( <> \) mean (\( R_j \))
Then : rename one or both the relationships.

Example : (See Figure 4.3)
The relationship Supervise is between the entities faculty and course in view_1, while it is between faculty and student in view_2. A possible solution is to rename Supervise by Course_Supervise in view_1.

c. Hominem conflict (HC_3) : Entity_Relationship.

**Description**: The name of an entity E in one view is identical with the name of a relationship in another view, but their meanings are different. If the meanings are same then this is a case of type conflict.

**Definition**:

If : (E=R) and mean(E) <> mean(R)

Then : rename E or R or both.

**Example**: (See Figure 4.4)

In view_1 OWNER is a relationship between entities PERSON and VEHICLE, while in view_2 it is an entity and means factory owner. A possible solution is to rename the relationship OWNER by OWN in view_1.

The other types of homonym conflicts such as Attribute_Attribute, Attribute_Entity and Attribute_Relationship are not considered because they do not cause any integration problem and also before producing the final integrated view (conceptual schema) the name of the entity to
which the attribute belongs can simply be added to the name of the attribute to distinguish it. For example, the attribute NAME (See Figure 4.4) of the entity PERSON refers to the person name, while it refers to the factory name in View_2. So to distinguish these two attributes we simply rename them respectively as PERSON_NAME and FACTORY_NAME.

4.3.2.2 Synonym Conflict

Synonym conflicts arise when different names are assigned to the same class of concepts in the separate views. The solution is to assign the same name to both the concepts which may be one of the original names or a new name. Unlike homonym conflicts, synonym conflicts can not be easily identified and the identification is entirely guided by the designer/analyst. We propose several heuristics and rules which will assist the designer/analyst to systematically identify synonym conflicts. The following types of synonyms conflicts are distinguished:


Description : Two entities have different names but have same meaning in the two views.

Definition :

If \( (E_i \neq E_j) \) and \( \text{mean}(E_i) = \text{mean}(E_j) \)

Then : rename both \( E_i \) and \( E_j \) by the same name.
AFTER RESOLUTION VIEW-1 BECOMES:

Fig. 4.4 Homonym conflict HC-3
**Example**: (See Figure 4.5a)

The entity Phd_student in view_1 has the same meaning as that of Research_scholar in view_2. A possible solution is to rename both by Phd_student.

**Heuristics**: (See Figure 4.5a, 4.5b, and 4.5c)

If the entities have identical keys or have similar names (like Department and Dept) or have participated in similar relationships;

Then the entities are possible candidates for synonyms.

b. **Synonym Conflict (SC_2): Relationship_Relationship**

**Description**: Two relationships have different names but have same meaning in the two views.

**Definition**: 
If \((R_i \neq R_j)\) and mean \((R_i) = \text{mean} (R_j)\)

Then rename both \(R_i\) and \(R_j\) by the same name.

**Heuristics**: 
If the relationships are defined over equivalent entity sets or have similar names, then the relationships are possible candidates for synonyms.

**Examples**: (See Figure 4.6a and 4.6b)

1. The relationship guide between the entities faculty and student in view_1 is same as the relationship supervise
Figure 4.5a Synonym Conflict SC1(Entities with identical keys)

Figure 4.5b Synonym conflict SC-1
(Entities with similar names)

Figure 4.5c Synonym conflict SC-1
between entities faculty and student in view_2. The resolution is to rename both the relationships as SUPERVISE in both the view.

2. The relationship SUPERVISE is between the entities TEACHER and STUDENT in view_1, and is same as the relationship PROJECT_SUPERVISE between entities FACULTY and PHD_STUDENT in view_2. The resolution is to rename project_supervise as SUPERVISE in view_2.

c. Synonym Conflict (SC_3): Entity_Relationship

Description: An entity and a relationships have different names but have same meaning in the two views. This is also a case of type conflict and will be treated in type_conflict analysis.

Definition:

If : (E <> R) and mean(E) = mean(R)
Then : rename both E and R by the same name.

Heuristics: If the key of an entity is same as the identifier of a relationship then the entity and relationship are possible candidates for synonyms.

Example: (See Figure 4.7)

The entity WORK of view_1 has same meaning as that of the relationship WORK_ON between entities EMPLOYEE and
PROJECT in view 2. The resolution is to rename both the entity and relationship as WORK in the two views.

d. Synonym Conflict (SC_4) : Attribute_Attribute

Description : Two attributes have different names but have same meaning in the separate views. The following three cases are distinguished:

Case-1. Both attributes belongs to entities.

Definition :

If : E₁ (Aᵢ) <> E₂ (Aⱼ)
and mean (Aᵢ) = mean(Aⱼ)
and [ (E₁ = E₂)
or { sub (E₁, E₂) or sub (E₂, E₁)}
or { sub (E₁, E) and sub(E₂, E) } ]
Then : rename Aᵢ and Aⱼ by the same name.

Example : (See Figure 4.8)

1) The attribute Course# of entity Course in view_1 is same as the attribute Course_ID of entity Course in view_2. A possible solution is to rename both attributes by Course_ID.

2) The attribute Amount of the entity Account in view_1 is same as the attribute Balance of the entity
Fig. 4.6a Synonym Conflict SC-2
(Relation Defined Over Equivalent sets of Entities)

Fig. 4.6b Synonym Conflict SC-2
(Relationships Having Similar Names)
Figure 4.7 Synonym Conflict SC-3
Savings_Account in view_2, and Savings_Account is a subset of Account. The resolution is to rename both the attributes by Balance.

Case_2 : Both the attributes belong to relationships.

Definition :

If : \( R_1(A_i) = R_2(A_j) \)
and \( \text{mean}(A_i) = \text{mean}(A_j) \)
and \( \text{mean}(R_1) = \text{mean}(R_2) \)

Then : rename both \( A_i \) and \( A_j \) by the same name.

Example : (See Figure 4.9)

The attribute quantity of the relationship SUPPLY in view_1 is same as the attribute AMOUNT of the same relationship in view_2 a possible solution is to rename both by QUANTITY.

Since the identification of attribute synonyms is entirely depends upon the user's assertions, the system presents a matrix as shown in SCREEN-41(appendix B), for attribute synonym assertion and with the help of the designer/analyst identifies and resolves them. It only presents those attributes which are not common to both the concepts.
fig 4.8a

fig 4.8b

fig 4.8 Synonym Conflict (SC4)
Case 3: One attribute belongs to an entity while the other attribute belongs to a relationship.

Definition:

If \( E(A_i) = R(A_j) \) and \( \text{mean}(A_i) = \text{mean}(A_j) \)
and \( E = R; \)

Then: rename both attributes \( A_i \) and \( A_j \) by the same name.

Example:

The attribute Loan_Type of the relationship LOAN in view_1 is identical with the attribute Type of the entity LOAN in view_2. A possible resolution is to rename both by LOAN_TYPE (See Figure 4.10).

4.3.3. Type Conflict Resolution

The goal of this step is to assign the same structure and type to concepts which are represented differently in the two views. The possible type conflicts arise in the EER model are attribute_attribute type conflict (the same attribute belongs to two different concepts in the two views), Attribute_entity conflict (An attribute in one view is equivalent to an entity in the other view), Entity_Relationship type conflict (An entity in one view is same as a relationship in the other view). We consider here only the first two types of the type_conflicts because the other
Figure 4.9 Synonym conflict SC-4
type_conflict, i.e., the entity-relationship type conflict does not cause any integration problem.

**Type conflict TC_1**:

**Description**: An attribute belongs to an entity in one view, while it belongs to a different entity or a relationship in the other view. The following cases are distinguished:

**Case_1**: An attribute of an entity in one view is same as a relationship attribute in the other view.

**Definition**:

\[
\text{If } E(A) = R(A) \text{ and mean } (E) \neq \text{ mean } (R) \\
\text{Then } \text{ delete A from R.}
\]

**Example** : (See Figure 4.11)

Quantity is an attribute of the relationship supplying view_1, while it is an attribute of the entity Part in view_2. A possible resolution is to simply delete the attribute Quantity.

**Case_2**: The key of an entity in one view is a part of the key of a different entity in the other view.

**Definition** :

\[
\text{IF } \text{ key } ("\text{View}_1", E_1, K_1) \text{ and Key } ("\text{View}_2", E_2, K_2) \text{ and } K_1 \subseteq K_2
\]
Figure 4.10 Synonym Conflict SC-4
(Attribute-Attribute)

Figure 4.11 Type Conflict TC-1
(Attribute-Attribute)
Then: 1. Delete key attributes of $E_1$ from $E_2$.
   2. Add the entity $E_1$ to view 2.
   3. Add the ID_relationship "$E_2$ ID $E_1$" to view 2.

Example: The key of the entity Account is [AC_NO] in the view 1, while it is a part of the key [TRANS_NO, AC_NO] of the entity Transaction in view 2. These conflicts are resolved by deleting the attribute AC_NO from the entity Transaction and adding the entity Account to View 2 and making the entity Transaction a weak_entity by adding the ID_relationship "Transaction ID ACCOUNT" to view 2 (See Figure 4.12).

Case-3: The key of an entity in one view is a part of the non_key attributes of a different entity in the other view.

Definition:
If:
   key ("View_1", $E_1$, K) and att ("View_2", $E_1$, A) and K \subseteq A;
Then:
   (1) Delete the key attributes of $E_1$ from the attributes of $E_2$.
   (2) Add $E_1$ and a 1:M relationship between $E_1$ and $E_2$ to the view "View_2".

Example:
The key of the entity Company is Company_Name in view "view 1", while it is a non_key attribute of the entity...
Employee in "view_2". This type conflict is resolved by deleting the attribute Company_Name from the entity Employee and adding the entity Company and a M:1 relationship "Works_for" between the entities Employee and Company to "View_2".

**Type Conflict TC2: Attribute_Entity**

**Description**: If an attribute of an entity/relationship in one view is equivalent to an entity in the other view, an attribute-entity conflict arises and such conflicts are resolved by transforming the attribute into an entity.

An attribute $A$ is said to be equivalent to an entity $E$ if:

(i) $A$ and $E$ have same name and meaning.

(ii) $A$ and $E$ have similar names and meaning (for example Department and Dept are equivalent).

(iii) $A$ is of the form $E_X$ or $E_2_X$ where $E$ and $E_1$ are similar names and $X$ is a key_type attribute, and $A$ and $E$ have same meaning.

The following two cases of Attribute-entity conflicts are distinguished:

**Case_1**: The attribute belongs to a relationship.
Definition:
If \( R(A) \) and \( E \) are equivalent and \( \text{mean}(A) = \text{mean}(E) \)
Then:
1. Delete the attribute \( A \) from the relationship \( R \).
2. Add the entity \( E \) to \( R \).
3. If \( A \) is a multivalued attribute of \( R_1 \), then set the cardinality of \( E \) in \( R \) to "M", else set it to "1".

Example:
The entity Grade in View_1 is an attribute of the relationship Enroll in view_2. The resolution is to replace the attribute Grade by the entity Grade in the relationship Enroll in the view_2.

Case_2: The attribute belongs to an entity.
Definition:
If \( E_1(A) \) and \( E_2 \) are equivalent and \( \text{mean}(E_1) \neq \text{mean}(E_2) \)
and \( \text{ent}(V_1, E_1) \)
and \( \text{ent}(V_2, E_2) \);
Then: do subcase:
Subcase_1:
If: the attribute belongs to the only key of \( E_1 \) in view \( V_1 \)
(i.e., the entity has no other alternative key);
Then: (i) Delete the attribute \( A \) from \( E_1 \).
(ii) Add the entity \( E_2 \) and an ID_relationship between \( E_1 \) and \( E_1 \) to the view \( V_1 \).
Subcase_2 :
If : the attribute belongs the key of E₁ and E₁ has an alternative key say (K) in view V₁;
Then : (i) Delete the attribute A from E₁.
(ii) Add the entity E₂ and a M:N relationship between E₁ and E₂ to the view V₂.
(iii) Declare K as the key of E₁ in view V₁.

Subcase_3 :
If : the attribute belongs to an alternative key of E₁ in view V₁;
Then : (i) Delete the attribute A from E₁.
(ii) Add the entity E₂ and a M:N relationship between E₁ and E₂ to view V₁.

Subcase_4 :
If : the attribute is a non-key attribute of the entity E₁ in view V₁;
Then : (i) Delete attribute A from E₁.
(ii) Add the entity E₂ and a M:1 relationship between E₁ and E₂ to the view V₁.

Example :

(1) In view_1 there is an entity AUTHOR; while in view_2 AUTHOR is an attribute of the entity Book and the only key of the entity Book is [Book_Name, Author]. The solution is to delete the attribute AUTHOR from the entity BOOK and
make the entity BOOK as a weak entity by adding the ID_relationship "Book ID AUTHOR" into view_2 (see Figure 4.13a).

2) AUTHOR is an entity in view_1, while in view_2 Author_Name is a key attribute of the entity BOOK. The solution is to delete the attribute Author_Name from the entity Book and add the M:N relationship "Written_by" between the entities BOOK and AUTHOR to view_1 (See figure 4.13b).

3) The attribute Dept_No. of the entity Employee in view_1 is the key of the entity Department of view_2. The solution is to transform the attribute Dept_No. by the entity Department and adding then M:1 relationship "Work_IN" between the entities Employee and Department to view_1 (See Fig. 4.13).

4.3.4. Degree Conflict Resolution

Degree conflict arise when the number of entities associated with identical relationship in the two views are different. They are easy to identify and are resolved in such a manner that the restructured relationships of both the views will produce relations with identical keys when translated into relational model, and hence are compatible for merging. Following three types of degree conflicts are distinguished:
INTEGRATED VIEW:

![Diagram of integrated view 1](image1)

Figure 4.13a

INTEGRATED VIEW:

![Diagram of integrated view 2](image2)

Type Conflict TC-2

Figure 4.13b
DEPARTMENT
[Deptt_no, Name, ...]

EMPLOYEE
[Emp_id, Add, Deptt_no, Name]

View-1
After Resolution View-2 will become:

EMPLOYEE
[Emp_id, Name, Address]

WORK-IN

DEPARTMENT
[Deptt_no, Name, ...]

Type Conflict TC-2

fig 4.13c
(i) **Degree Conflicts DC_1**: (Binary - N_ary-Relationship)

**Description**: A relationship is defined over two entities in one view, while it is defined over more than two entities in the other view, and have the same meaning in both the view:

**Definition**: 
If : $R_1(E_1, E_2) = R_2(E_1, E_2, E_3)$ 
and $\text{mean}(R_1) = \text{mean}(R_2)$; 
Then: if the cardinality of $E_3$ in $R_2$ is not 1, 
then set it to 1.

**Example** : (See Figure 4.14)  
The relationship ENROLL is defined over entities Student and Course in view_1, while it is defined over entities Student, Course and Grade in view_2 and has same meaning in both the views. The resolution is to set the cardinality of Grade to 1 in the relationship ENROLL in view_2 if it is not.

(ii) **Degree Conflict DC_2**: Binary-relationship Aggregation

**Description**: A relationship is defined over two entities in one view, while it is defined over an entity and a relationship in the other view.
Figure 4.14: Degree Conflict DC-1.

Figure 4.15: Degree Conflict DC-2.
Definition:

If \( R_1(E_1, E_2) = R_2(E_1, R) \)
and \( \text{mean}(R_1) = \text{mean}(R_1) \);
Then : do case :

Case_1 : Key \((E_2, V_1) = \text{Ident}(R, V_2)\).
Solution : Replace relationship \( R \) by entity \( E_2 \) in the relationship \( R_2 \).

Case_2 : Key \((E_2, V_1) <> \text{key}(R, V_2)\).
Solution : Replace entity \( E_2 \) by the relationship \( R \) in the relationship \( R_1 \).

Example : (See Figure 4.15).

1) The relationship Teach is between the entities Faculty and Course in view_1 while it is between the entity Faculty and the relationship OFFER, and the key of Course is same as that of OFFER. The resolution is to replace OFFER by Course in the relationship Teach in view_1.

2) In the above example if the relationship OFFER is a M:N (many to many) relationship, then its key is \([\text{Course}_\text{No}, \text{Dept}_\text{ID}]\) which is not equal to the key of the entity Course. In this case the degree conflict can be resolved by replacing the entity Course by the relationship OFFER in the relationship Teach in view_1, and adding the entity Department to it if it is not already there.
Degree Conflict DC_3 (N-ary Relationships_Aggregation)

Description: A relationship is defined over an entity and a relationship in one view, while it is defined over more than two entities in the other view.

Definition:

IF $R_1(E_1, R_1)$ and $R_2(E_1, E_2, E_3, \ldots)$
and $\text{mean}(R_1) = \text{mean}(R_2)$
and $R$ is defined over $E_2$ and $E_3$.

Then: do case:

Case 1: The cardinalities of $E_1$ and $E_2$ in $R_2$ are some as the cardinalities of $E_2$ and $E_3$ respectively in $R$.

Solution: 1) Delete the edge between $R$ and $R_1$.

2) Add an edge between $R_1$ and $E_2$ and between $R_1$ and $E_3$.

Case 2: The cardinalities of $E_2$ and $E_3$ in $R_2$ are not the same as the cardinalities of $E_2$ and $E_3$ in $R$.

Solution: 1) Delete edges between $E_2$ and $R_2$ and between $E_3$ and $R_2$.

2) Add an edge between $R$ and $R_2$.

Example (see Figure 4.16)

The relationship Supply is defined between the entity Supplier and the relationship Use in view_1, while it is defined over three entities Supplier, Part and Project in view_2. The relationship Use in view_1 is also defined over

127
FIGURE 4.16  DEGREE CONFLICT
the entities Part and Project. The resolution is done according to case_1 and case_2.

4.3.5. Role Conflict Resolution:

**Description:** Role Conflicts arise when different roles are associated with identical entities in identical relationships in the two views. They are easy to identify and are resolved by assigning the same roles to both the entities.

**Definition:**

If 

\[
\text{edge ("View_1", R, E_1, ROLE1, _ )}
\]

and 

\[
\text{edge ("View_2", R, E_2, ROLE2, _ )}
\]

and 

\[
[E_1 = E_2 \text{ or sub}(E_1, E_2) \text{ or sub}(E_2, E_1) \text{ or } \{\text{sub}(E_1, E) \text{ and sub}(E_2, E) \}]
\]

and 

\[
\text{ROLE1 <> ROLE2}
\]

Then: rename roles ROLE1 and ROLE2 by the same role name which may be ROLE1 or ROLE2 or a new one.

**Example** (See Figure 4.17)

The roles of the entity Person in view_1 are Male and Female in the relationship Spouse while they are Husband and Wife in the view_2. The resolution is to change to Male and Female in view_2.
FIGURE 4.17 Role Conflict
4.3.6. Cardinality Conflict Resolution:

Cardinality conflicts arise when different cardinalities are associated with "Similar entities" identical relationships in the two views.

Description: Two different cardinalities are associated with similar entities in the same relationship in the two views, in this case the designer has to decide which is the correct cardinality and change them accordingly.

Definition:
If \( \text{edge ("View_1", R, E_1, _, C_1)} \)
and \( \text{edge ("View_2", R, E_2, _, C_2)} \)
and \( [ E_1 = E_2 \text{ or } \text{sub}(E_1, E_2) \text{ or } \text{sub}(E_2, E_1) \text{ or } \{ \text{sub}(E_1, E) \text{ and } \text{sub}(E_2, E) \} ] \)
and \( C_1 <> C_2; \)
Then : change the cardinality of \( E_1 \) and \( E_2 \) to the same value.

Example (See Figure 4.18(A) and 4.18(B))
1. The relationship offer between entities department and course in M:N in view_1, while it is 1:M in view_2. The resolution in to change the cardinalate of the entity department to one in the relationship offer in view_2.
2. The cardinality of the entity course in the relationship enrollment in many in view_1 which it is one in view_2,
Figure 4.18 (A) Cardinality Conflict.

Figure 4.18 (B) Cardinality Conflict.
and the entity pg_student in a subset of the entity student. The resolution is to change the cardinality of the entity course by many in view_2.

4.3.7. Hierarchical Conflict Resolution

Hierarchical conflicts arise when there exist contradictory hierarchical links between the same entities in the two view, or if there exists a hierarchy link between two entities in one view, while a relationship exists between the same two entities in the other view. The following two types of hierarchical conflicts are distinguished:

Hierarchy conflict (HYC_1): Hierarchy_Hierarchy.

Description: The entity E1 is a subset of entity E2 in one view, while in the other view the entity E2 is a subset of the entity E1. The resolution is to recheck the pair of entities for correct hierarchical link between them and change the structure of the view accordingly.

Definition:

If \( \text{is}(E_1, E_2, V_1) \) and \( \text{is}(E_2, E_1, V_1) \);
then \[ \text{ask the designer which hierarchical link between the two entities } E_1 \text{ and } E_2 \text{ is correct and change the link structure in the respective view.} \]
We define the "is" rule recursively as follows:

\[
is (E_1, E_2, V) :: \text{edge}(V, E_1, E_2, "ISA", ").
\]

\[
is (E_1, E_2, V) :: \text{edge}(V, E, E_3, "ISA", ")", \text{is}(E_1, E_3, V).
\]

**Example:** (See Figure 4.19)

In the view_1 the entity Employee is a subset of the entity Person, while in view_2, the entity Person is a subset of Employee. The designer identifies that Employee is a subset of Person is the correct hierarchical link, and the subset relationship between Employee and Person is changed in view_2 and the attributes of the entity employee are set accordingly.

**Hierarchy conflict HVC_2.** Hierarchy_Relationship

**Description:** There is a hierarchical link exists between two entities in one view, while in the other view there exists a relationship between the two entities. The resolution is to recheck the pair of entities and decide whether to keep the relationship information or the subset hierarchy information between the two entities and change the structure of the view accordingly.

**Definition:**

If \( E_1 \) is a subset of \( E_2 \) in view_1.
and \( R(E_1, E_2) \) in view_2.

Then do case:
Figure 4.19. Hierarchical Conflict HVC-1.

Figure 4.20 (A) Hierarchy Conflict HVC-2.
Case 1. R is an n_ary relationship with n>2.
solution: Delete entity E_2 from the entity sets of R, i.e.,
delete the link between E_1 and R)

Case 2. R is a binary relationship and is of the type
1) Contains
2) Possesses
3) instance_of
4) has/have
5) is_a
6) components_of
7) part_of
Solution: Delete R from view_2 and add the subset hierarchy
relationship " E_1 is a subset of E_2 " to it.

Case 3. R is a binary relationship and E_1 is transitively
subset of E_2.
Solution: Delete the relationship R from view_2.

Case 4. R is binary relationship of any other key.
Solution: If the designer/user wants to keep the relationship
information then delete the subset hierarchy between
E_1 and E_2 from view_1, else delete the relationship
information from view_2.
Example:

1. In view_1 the entity Coupon is a subset of the entity Ticket; in view_2 Contains is a binary relationship between the entities Ticket and coupon. The resolution is to replace the relationship Contains by the subset hierarchy "Coupon ISA ticket" in view_2 (see Figure 4.20(A))

2. In view_1 Work_in is a relationship between entities Manager, Employee and Department, while in view_2, the hierarchy "Manager ISA employee" exists. The resolution is to delete the entity Manager from the relationship Work_in and adding the hierarchical link "Manager ISA Employee" to it (See Figure 4.20(B))

3. In view_1 Prerequisite is a relationship between entities Course and Pre_course while in view_2, Pre_course is a subset of entity Course. The designer decided to keep the relationship Prerequisite information and delete the subset relationship information (see Figure 4.20(C))

4.3.8. Key Conflict Resolution

Key conflicts arise when the same entity or equivalent entities (for example student and Pg_student entities are equivalent) have different identifier in different views. Most of the key conflicts have already been resolved during attribute_synonym resolution. In this step the
Figure 4.20 (B) Hierarchy Conflict HVC-2.

View 1.

Figure 4.20 (C) Hierarchy Conflict HVC-2.

View 2.

Figure 4.20 Hierarchy - Relationship Conflict.
remaining key conflicts are identified and resolved by assigning the same identifier to both the entities.

**Description:**
If:

\[
\text{mean}(E_1) = \text{mean}(E_2)
\]

or
\[
\text{Sub}(E_1, E_2) \text{ or sub } (E_2, E_1)
\]

or
\[
\{ \text{Sub}(E_1, E) \text{ and Sub}(E_2, E) \} \]

and
\[
\text{key}(E_1, V_1) <> \text{ and Sub key } (E_2, V_2)
\]

Then:

\[
\text{set alt}(E_1, V_1) = \text{Key } (E_1, V_1) \text{ and Key } (E_1, V_1) = \text{Key } (E_2, V_2)
\]

or
\[
\text{set alt}(E_2, V_2) = \text{Key } (E_2, V_2) \text{ and Key } (E_2, V_2) = \text{key } (E_1, V_1)
\]

**Example:** (See Figure 4.21)

In view_1 the key of the entity Student is [STUD_ID] while in view_2 the key of Pg_Student is [STUD_NAME, STUD_ADD] and Pg_student is a subset of student. The resolution is to set the key of Pg_student to [STUD_ID] and the set [STUD_NAME, STUD_ADD] as an alternative key of Pg_student in view_2.

The following heuristics are used to select the key from the two alternatives.

i) The simplest key is chosen over the others, i.e. the one with the least number of attributes.
Figure 4.21 Key Conflict.

After integration:

Integrated View

FIGURE 4.22 (Entity-Relationship Integration
ii) If there is a tie for the simplest key, then choose the one that is:
   a) of the form "generic entity name concatenated with a key attribute name.

   b) The most application specific as suggested by the designer.

4.3.9. Merging

Once various conflict between the two views are resolved and interschema properties are identified the two views are merged into a single partial integrated views and the interschema properties are added to it. This step is carried out automatically without the aid of the designer and is done in three phases, viz :(i) Entity_Relationship integration : during this phase an entity of one view which is similar to a relationship of another view are merged into a single concept nested entity in the partial integrated view; (2) entity integration. In this phrase entity of the pair of views are integrated; and (3). Relationship integration; in this phase relationship of both the views are integrated. We describe below each of those phases, for each phase we describe the rules for merging, and for each rule examples are provided for clarification.
Phase I  Entity_Relationship Integration

Description: If an entity E of one view (Say V1) is same as a relationship R of the other view (say V2) then R will be added to the integrated view VI and declare as a nested entity and its attributes are defined as follows:

\[ \text{Att("VI",R)} = \left[ \text{att("V1",E)} - \right. \]
\[ \left. \left\{ \text{U (att("V2",E_i) U key("V2",E_i))} \right\} \right. \]
\[ \left. \left. \text{U att("V2",R)} \right] \right. \]

Where \( E_i, \ i = 1,2,\ldots,n \) are the set of entities over which the relationship R is defined.

Definition:
If : \( \text{ent (''V1'', E) = rel (''V2'',R)} \)
and \( \text{ent_set (''V2'',R) (E_1,E_2,\ldots,E_n)}; \)
Then :
1. Add R to the integrated view "VI".
2. Define the relationship attributes of R as defined above.
3. Do not add the entity E to the integrated view "V2".

Example : Consider the relationship OWNER of view "V1" and the entity OWNER of view "V1". The result of integration of these two concept is show in Figure 4.22.
Phase II: Entity Integration

The following rules are used for integrating entities of the two views:

Rule 1. Identical Domains

Description: If the entity E belong to both the view V1 and V2, then it is added to the integrated view V1 and its attributes are defined as follows.

\[
\text{att} ("V1", E) = \text{att} ("V1", E) \cup \text{att} ("V2", E)
\]

\[
\text{Key} ("V1", E) = \text{Key} ("V1", E).
\]

Definition:

If : ent ("V1", E) and ent ("V2", E)

Then :

1. Add entity E to "V1".
2. Defined the key of E in "V1" as the key of E in "V1".
3. Define the other attributes of E in "V1" as the union of the other attributes of the entity E in both the view "V1" and "V2".

Example: Consider the entity EMPLOYEE of the two views. The result of integrating it are shown in Figure 4.23.

Rule 2. Subset Domains

Description: If an entity E1 of one view V1 is a subset of the entity E2 of view V2, then the two entities E1 and E2 are added to the integrated view and the subset relationship "E1
FIGURE 4.23 ENTITY INTEGRATION (IDENTICAL DOMAINS)

FIGURE 4.24 ENTITY INTEGRATION (SUBSET DOMAINS)
ISA E₂", is added to it. the keys and other attributes of E₁ and E₂ in the integrated view "VI" are defined as follows:

\[
\text{att}(\text{"VI"}, E₂) = \text{att}(\text{"V2"}, E₂) \\
\text{att}(\text{"VI"}, E₁) = \text{att}(\text{"V1"}, E₁) - \text{att}(\text{"V2"}, E₂) \\
\text{key ("VI"}, E₂) = \text{key ("V2"}, E₂)
\]

The key of E₁ in "VI" is not defined as it can be obtained from the ISA relationship.

**Definition:**

If 

\[
: \text{ent ("VI"}, E₁) \text{ and ent ("V2"}, E₂) \\
\text{and } \text{Sub}(E₁, E₂)
\]

then : 1. Add E₁ and E₂ to the integrated view "VI".

2. Add the edge Edge("VI"}, E₁, E₂, "ISA", "") to "VI".

3. Define the key and attributes of E₂ in "VI" same as the key and attributes of E₂ in V₂ respectively.

4. Define attributes of E₁ view "VI" as the difference of the attributes of entity E₁ in "V1" and the set of attributes of E₂ in "V2"; and do not define the key of E₁ in "VI".

**Example:**

Consider the following two entities of the two views:

Manager (EMP_ID, type, salary, name)

Employee(EMP_ID, name, add, salary)
Obviously Manager is subset of the entity Employee, the result of integrating these two entities are shown in Figure 4.24.

**Rule 3. Common domains**

**Description:** If pair of entities $E_1$ and $E_2$ of the two views $V_1$ and $V_2$ have a common domain then three entities $E_1$, $E_2$ and $E$ are created in the integrated view $V_I$. The entity $E$ is generalization of the two entities $E_1$ and $E_2$ and its attributes of $E$ are defined as the intersection of the set of attributes of entities $E_1$ and $E_2$.

**Definition:**

If $\text{ent}("V_1", E_1)$ and $\text{ent}("V_2", E_2)$ and $\{\text{sub}(E_1, E) \text{ and } \text{sub}(E_2, E)\}$

then:

1. add entities $E_1$, $E_2$, and $E$ to the integrated view "$V_I$".
2. add the isa-hierarchies "$E_1 \text{ ISA } E"$ and "$E_2 \text{ ISA } E$" to the integrated view.
3. define the attributes of $E$, $E_1$ and $E_2$ as follows:
   
   $\text{att("V_I", E)} = \text{att("V_1", E_1)} \text{ att("V_2", E_2)}$
   
   $\text{att("V_I", E_1)} = \text{att("V_1", E_1)} \text{ att("V_I", E_2)}$
   
   $\text{att("V_I", E_2)} = \text{att("V_2", E_2)} \text{ att("V_1", E)}$

4. The key of $E$ in is set to that of the key of $E_1$.
5. The key of $E_1$ and $E_2$ in "$V_I$" are not defined as they can be obtained from the ISA relationships.
Examples: Consider the following two entities:
Grade_student(Stud_ID,name,Add,GPA,Type_of_support,Grade_course)
Science_student(Stud_ID,Name,Science_course,Add)

Assume that some but not all graduate students are science students. The result of integrating these two entities are shown in Figure 4.25.

Rule 4: Unrelated entities
Description: Unrelated entities of both the views are simply added to the integrated view together with their attributes and keys.

Phase III: Relationship Integration

In this phase the relationships of the pair of views are integrated. Relationships are integrated asking rules for edge merging. The remaining relationships which are not common to both the views are simply copied into the integrated view. The edge merging technique can merge identical relationships with same degree or different degree. For relationship of different degrees, it merges the related edges and the remaining unrelated edges are added to the relationships in the integrated view. Following rules are used for merging pair of edges of identical relationships of the two views:
Figure A.25 Entity Interaction with Common Domains
Rule 1. Edges with identical entities.

Definition:

If : edge("V1",R,E,L,C) and edge("V2",R,E,L,C)

Then : add edge("VI",R,E,L,C) to the integrated view VI.

Example:

Consider the relationship "Registration" of the two views "V1" and "V2". Following Rule 1 an edge edge("VI","Registration","Course","","N") will be created in the integrated view "VI".

Rule 2. Edges with entities with subset domains.

Definition:

If : edge("V1",R,E_1,L,C) and edge("V2",R,E_2,L,C)

and subset(E_1,E_2);

then : create edge("VI",R,E_2,L,C) in the integrated view.

Example : Consider the relationship "Registration" of the two views "V1" and "V2". In this case, the edge("VI","Registration","Student","","N") will be created in the integrated view "VI" as shown in the Fig. (Note that the subset relationship "Pg_student, ISA student" is already added during entity integration.) See Fig. 4.26.

Rule 3 : Edge with entities with common domain

Definition:

If : edge("V1",R,E_1,L,C) and edge("V2",R,E_2,L,C)

and sub(E_1,E) and sub(E_2,E);
Figure 4.26 Relationship Integration
View 1

View 2

Figure 4.27 Relationship Integration (Edges with entities having common domain)
Then: create edge("VI",R,E,L,C) in the integrated view "VI".

**Example**: Consider the relationship "Registration" of the two views "V1" and "V2" shown in Fig. 4.27. In this case both the entities Science_student and Grad_student are subsets of the entity Student. The integration of relationships are shown in Figure 4.27.

4.4. **Summary**

This section has presented a step_by_step methodology for designing user views of a database application. The methodology is based on an extended entity relationship model. It has shown how the EER model corresponding to the user view of the database can be constructed interactively. Motivation for this work is an increasing need for effective and easy to use design support tools in any system development environment and in particular in database applications which becomes more and more lounge and semantically complex and where the development cost should be limited. The methodology is described in a step_by_step manner and provided with several rules and heuristics to facilitate the design of an interactive tool for designing user views of a database, and it is implemented in our expert system VMITS.