TABLE OF CONTENTS

Chapter 3 : Relation Database Model

1.1 A Logical View Of Data

A relation model enables you to view data *logically* rather than *physically*. Therefore tables play a prominent role in the relational model. A table has the advantages of structural and data independence, although it still resembles a file from a conceptual point of view. Because you can think of related records as being stored in independent tables, the relational database model is much easier to understand that it's hierarchical and network database predecessors.

1.1.1 Characteristics of a Relational Table

- A table is perceived as a two-dimensional structure composed of rows and columns.
- Each table row (*tuple*) represents a single entity occurrence within the entity set.
- Each table column represents an attribute, and each column has a distinct name.
- All values in a column must conform to the same data format.
- Each column has a specific range of values know as the *attribute domain.*
- The order of the rows and columns is immaterial to the DBMS.
- Each table must have an attribute or a combination of attributes that uniquely identifies each row.

1.1.2 Keys

Keys consist of one or more attributes that determine other attributes, and its role is therefore based on determination. If you know the value of attribute A, you can look up or determine the value of attribute B. Any attribute that is part of a key is known as a *key attribute*.

Within a table, each primary key value must be unique to ensure that each row is uniquely identified by the primary key. In that case, the table is said to exhibit *entity integrity*. To maintain entity integrity, a *null* value is not permitted in the primary key.

Nulls, if used improperly, can create problems because they have many different meanings. A null for example can represent an unknown attribute value, or a know, but missing, attribute value, or a not applicable condition. Depending on the sophistication of the application development software, nulls can create problems when functions such as COUNT, AVERAGE, and SUM are used. In addition, nulls can create logical problems when relational tables are linked. To avoid nulls, some designers use special codes, known as *flags* to indicate the absence of some value.

Controlled redundancy makes the relation database work. Tables within the database share common attributes that enable the tables to be linked together. Multiple occurrences of values in a table are not redundant when they are required to make the

relationship work. Data redundancy exists only when there is *unnecessary* duplication of attribute values.

Relational Database Keys can be defined as follows:

- **Superkey** is an attribute or combination of attributes that uniquely identifies each row in a table.
- **Candidate key** is a minimal superkey. A superkey that does not contain a subset of attributes that is itself a superkey
- **Primary key** is a candidate key selected to uniquely identify all other attribute values in any given row. Cannot contain null entries.
- **Secondary key** is an attribute or combination of attributes used to strictly for data retrieval purposes.
- **Foreign key** is an attribute or combination of attributes in one table whose values must either match the primary key in another table or be null.

1.2 Integrity Rules

1.2.1 Entity Integrity

The requirements for entity integrity are that all primary key entries are unique, and no part of a primary key may be null.

The purpose is that each row will have a unique identity, and foreign key values can properly reference primary key values.

For example no invoice can have a duplicate number, nor can it be null. In short, all invoices are uniquely identified by their invoice number.

1.2.2 Referential Integrity

The requirements are that a foreign key may have either a null entry, as long as it is not a part of its table's primary key, or an entry that matches the primary key value in a table to which it is related. Every non-null foreign key value *must* reference an *existing* primary key value.

The purpose is that it is possible for an attribute NOT to have a corresponding value, but it will be impossible to have an invalid entry. The enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table.

For example a customer might not yet have an assigned sales representative or number, but it will be impossible to have an invalid sales representative or number.

1.3 Relational Set Operators

The degree of relational completeness can be defined by the extent to which relational algebra is supported. *Relation algebra* defines the theoretical way of manipulating table contents using the eight relation operators: SELECT, PROJECT, JOIN, INTERSECT, UNION,

DIFFERENCE, PRODUCT and DIVIDE. The use of relation operators have the property of *closure*, that is the use of relational algebra operators on existing tables or relations produces new relations.

- **UNION** combines all rows from two tables, excluding duplicate rows. The tables must have the same attribute characteristics. When two or more tables share the same number of columns, when the columns have the same names, and when they share the same domains, they are said to be *union compatible*.
- **INTERSECT** yields only the rows that appear in both tables. The tables must be union compatible to yield valid results.
- **DIFFERENCE** yields all rows in one table that are not found in the other table, that is, it subtracts one table from the other. The tables must be union compatible to yield valid results.
- **PRODUCT** yields all possible pairs of rows from two tables also known as the Cartesian product. Therefore if one table has six rows and the other table has three rows, the PRODUCT yields a list composed of 6 times 3 = 18 rows.
- **SELECT** also known as *RESTRICT*, yields values for all rows found in a table. SELECT can be used to list all of the row values, or it can yield those row values that match a specified criterion. SELECT therefore yields a horizontal subset of a table.
- **PROJECT** yields all values for selected attributes. PROJECT therefore yields a vertical subset of a table.
- **DIVIDE** requires the use of one single-column table and one two-column table. U
- **JOIN** allows information to be combined from two or more tables. It allows the use of independent tables linked by common attributes.
    - **Equijoin** links tables on the basis of an equality condition that compares specified columns of each table. The outcome of the equijoin does not eliminate duplicate columns and the condition or criterion used to join the tables must be explicitly defined. It gets its name from the use of the = or comparison operator.
    - **Theta Join** is an equijoin that uses one of the other comparison operators.
    - **Outer Join** Matched pairs are retained and any unmatched values in other table are left null.
        - **Left outer join** yields all the rows in the “left” table, including those that do not have a matching value in the “right” table.
        - **Right outer join** yields all of the rows in the “right” table, including those that do not have a matching value in the “left” table.
    - **Natural Join** links tables by selecting only the rows with common values in their common attributes. A natural join is the result of a three stage process:

- First a PRODUCT of the tables is created
- Second a SELECT is performed on the output of the first step to yield only the rows for which the common attributes values are equal. The common columns are referred to as the *join columns*.
- Third a PROJECT is performed on the results of the second step to yield a single copy of each attribute, thereby eliminating duplicate columns.

The final outcome of a natural join yields a table that does not include unmatched pairs and provides only the copies of the matches. If no match is made between the table rows, the new table does not include the unmatched row. Also the column on which the join was made, occurs only once in the new table.

1.4 Data Dictionary and The System Catalog

1.4.1 Data dictionary

The *data dictionary* provides a detailed accounting of all tables found within the user or designer created database. Thus the data dictionary contains at least all of the attribute names and characteristics for each table in the system. It therefore contains metadata, or data about data. It is sometimes described as "the database designer's database" because it records the design decisions about tables and their structures.

1.4.2 System Catalog

Like the data dictionary the system catalog contains metadata. It can be described as a detailed system data dictionary that describes all objects within the database. Because the system catalog contains all required data dictionary information, the terms *system catalog* and *data dictionary* are often interchangeable. The system catalog is actually a system-created database whose tables store the user or designer created database characteristics and contents and can therefore be queried like any other user or designer created database.

1.5 Relationships Within The Relational Database

1.5.1 One-To-Many Relationship

The one to many relationships is the relational modeling ideal and should be the norm for any relational database design. This relationship can be found in any database environment. For example one course can generate many classes but each class refers to only one course.

1.5.2 One-To-One Relationship

The one to one relationship should be rare in any relational database design. In this relationship one entity can be related to only one other entity, and vice versa. For example a professor can only chair one department and one department can only have one department chair. The existence of the one to one relationship sometimes means that the entity components were not defined properly. It could also indicate that the two entities actually belong in the same table. As rare as one to one relationships are certain conditions absolutely require their use.

1.5.3 Many-To-Many Relationship

Many to many relationships cannot be implemented as such in the relational model but will be broken to produce a set of one to many relationships. For example a student can have many classes and each class has many students. Problems with many to many relationships can be overcome by creating a *composite entity* or a *bridge entity*.

1.6 Data Redundancy Revisited

Data redundancy leads to data anomalies. Those anomalies can destroy the effectiveness of the database. It is however possible to control data redundancies by using common attributes also known as foreign keys, that are shared by tables. The proper use of foreign keys is crucial to data redundancy control; they however do not eliminate data redundancies because they can be repeated many times. There are also instances where data redundancy is necessary.

1.7 Indexes

An *Index* is an orderly arrangement used to logically access rows in a table. From a conceptual point of view, an index is composed of an index key and a set of pointers. The *index key* is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key. The index key can have multiple attributes.

A *unique index* is an index in which the index key can have only one pointer value or row associated with it.

A table can have many indexes, but each index is associated with only one table.

1.8 CODDs Relational Database Rules

- **Information**. All information in a relational database must be logically represented as column values in rows within tables
- **Guaranteed Access**. Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
- **Systematic Treatment of Nulls**. Nulls must be represented and treated in a systematic way, independent of data type.
- **Dynamic On-Line Catalog Based on the Relational Model.** The metadata must be stored and managed as ordinary data, that is, in tables within the database. Such data must be available to authorized users using the standard database relational language.
- **Comprehensive Data Sublanguage.** The relational database may support many languages. However it must support one well defined declarative language with support for data definition, view definition, data manipulation, integrity constraints, authorization, and transaction management.
- **View Updating.** Any view that is theoretically updatable must be updatable through the system.

- **High-Level Insert, Update and Delete**. The database must support set-level inserts, updates, and deletes.
- **Physical Data Independence**. Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.
- **Logical Data Independence**. Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values.
- **Integrity Independence**. All relation integrity constraints must be definable in the relational language and stored in the system catalog, not at application level.
- **Distribution Independence**. The end users and application programs are unaware and unaffected by the data location
- **Nonsubversion.** If the system supports low-level access to the data, there must not be a way to bypass the integrity rules of the database.
- **Rule Zero.** All preceding rules are based on the notion that in order for a database to be considered relational, it must use its relational facilities exclusively to manage the database.

Chapter 4 : Entity Relationship Modeling

1.9 Entity Relationship Model

The entity relationship model forms the basis of an entity relationship diagram. The diagram represents the conceptual database as viewed by the end user. The diagrams depict the database's main components, entities, attributes and relationships.

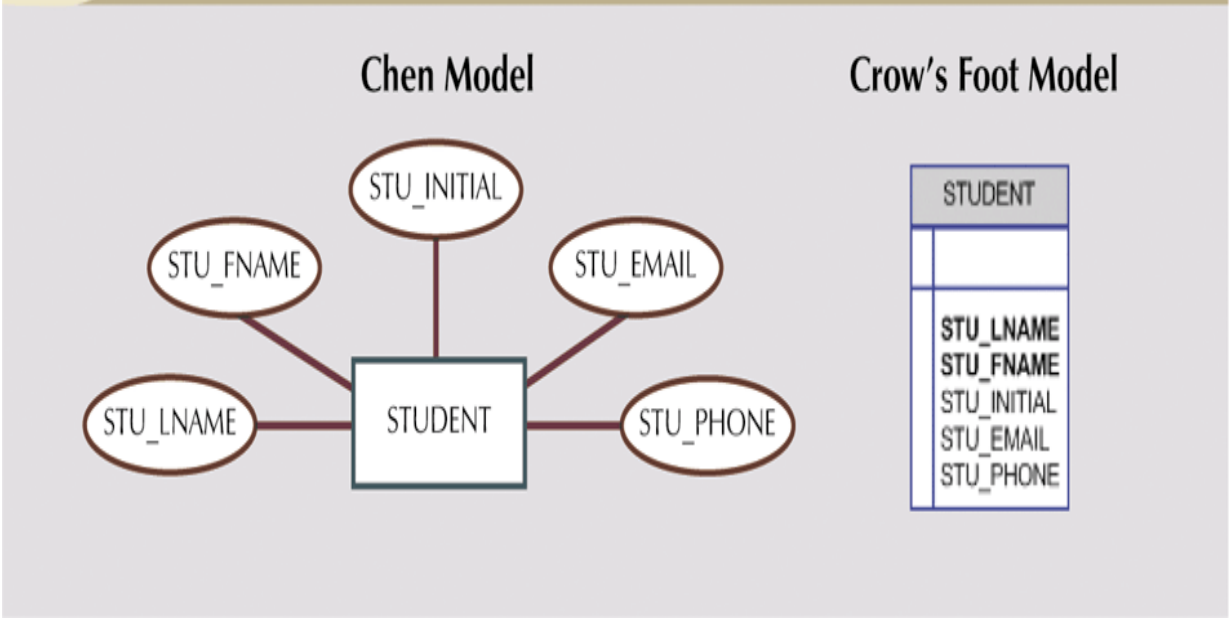
1.9.1 Entities

An entity is an object of interest to the end user. An entity actually refers to the *entity set* and not to a single entity occurrence. In other words the word *entity* in the entity relationship model corresponds to a table and not to a row in the relational environment. In model refers to a specific table row as an *entity instance* or *entity occurrence*. In both the Chen and Crow's Foot models, an entity is represented by a rectangle containing the entity's name. The entity name, a noun, is usually written in all capital letters.

1.9.2 Attributes

Attributes are characteristics of entities. In the Chen model attributes are represented by ovals and are connected to the entity rectangle with a line. Each oval contains the name of the attribute it represents. In the Crow's Foot model the attributes are written in the attribute box below the entity rectangle.

FIGURE 4.1 The attributes of the STUDENT entity



Attributes have a domain. A domain is the attribute's set of possible values. The domain for a persons SEX would be F or M. Attributes may share a domain.

The entity relationship model uses *identifiers* to uniquely identify each entity instance. In the relational model, such identifiers are mapped to primary keys in tables. Identifiers are underlined in the entity relational diagram. Key attributes are also underlined in a frequently used table structure shorthand notation.

Ideally, a primary key is composed of only a single attribute. It is possible however to use a *composite key*. Both key attributes are then underlined in the entity notation.

Attributes are classified as simple or composite. A *composite* attribute is an attribute that can be further subdivided to yield additional attributes. For example an ADDRESS can be subdivided into street, city and zip code. A *simple* attribute is an attribute that cannot be subdivided. For example AGE. To facilitate detailed queries, it is usually appropriate to change composite attributes into a series of simple attributes.

A *single valued attribute* is an attribute that can have only a single value. For example a person can only have one identification number.

Multivalued attributes are attributes that can have many values. For example a person may have several college degrees. If multivalued attributes exists, the designer must decide on one of the following courses of action:

- Create several new attributes within the original entity, one for each of the original multivalued attributes. Although this solution seems to work it can lead to major structural problems in the table.

- Create a new entity composed of the original multivalued attribute's components. The new independent entity is then related to the original entity in a one to many relationship.

A *derived attribute* is an attribute whose value is calculated from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. The advantages of storing derived attributes is that it saves CPU processing cycles, data value is readily available and can be used to keep track of historical data. The disadvantages of storing are that it requires constant maintenance to ensure the derived value is current, especially if any values used in the calculation changes. The advantages of not storing derived attributes are that it saves on storage space and computation always yields correct values. The disadvantages of not storing derived attributes are that it uses CPU processing cycles and adds coding complexity to queries.

1.9.3 Relationships

A relationship is an association between entities. The entities that participate in a relationship are also knows as *participants*. The relationship name is an active or passive verb for example a STUDENT *takes* a CLASS. Relationships between entities always operate in both directions and a relationship classification is difficult to establish if you know only one side of the relationship.

1.9.4 Connectivity and Cardinality

The term *connectivity* is used to describe the relationship classification, in other words if it is a one to one, one to many or many to many relationship.

Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity.

Connectivities and cardinalities are established by very concise statements known as *business rules*. Such rules, derived from precise and detailed description of an organization's data environment, also establish the entity relationship model's entities, attributes, relationships, connectivities, cardinalities and constraints.

1.9.5 Existence Dependence

An entity is said to be *existence dependant* if it can exist in the database only when it is associated with another related entity occurrence. In implementation terms, an entity is existence dependant if it as has mandatory foreign key of which the attribute cannot be null.

If an entity can exist apart from one or more related entities, it is said to be *existence independent*. Such entities are sometimes referred to as *strong* or *regular* entities.

1.9.6 Relationship Strength

The concept of relationships strength is based on how the primary key of a related entity is defined. To implement a relationship, the primary key of one entity appears as a foreign key in the related entity.

A *weak relationship*, also known as *non-identifying relationship*, exists if the primary key of the related entity does not contain a primary key component of the parent entity. By default, relationships are established by having the primary key in the parent entity appear as a foreign key on the related entity.

A *strong relationship*, also know as an *identifying relationship*, exists when the parent key of the related entity contains a parent key component of the parent entity.

1.9.7 Weak entities

A *weak entity* is one that meets two conditions:

1. It is existence dependent
2. It has a primary key that is partially or totally derived from the parent entity in the relationship.

The database designer usually determines whether an entity can be described as weak based on the business rules.

1.9.8 Relationship Participation

Participation in an entity relationship is either optional or mandatory.

Optional participation means that one entity occurrence doe not *require* a corresponding entity occurrence in a particular relationship. The existence of *optionality* indicates that the minimum cardinality is 0 for the optional entity.

Mandatory participation means that one entity occurrence *requires* a corresponding entity occurrence in a particular relationship. If no optionality symbol is depicted with the entity, the entity exists in a mandatory relationship with the related entity. The existence of a mandatory relationship indicates that the minimum cardinality is 1 for the mandatory entity.

TABLE 4.3 Crow's Foot Symbols

CROW'S FOOT SYMBOL	CARDINALITY	COMMENT
	(0,N)	"Many" side is optional.
	(1,N)	"Many" side is mandatory.
	(1,1)	"1" side is mandatory.
	(0,1)	"1" side is optional.

1.9.9 Relationship Degree

A *relationship degree* indicates the number of entities or participants associated with a relationship.

A *unary relationship* exists when an association is maintained within a single entity.

A *binary relationship* exists when two entities are associated.

A *ternary relationship* exists when three entities are associated.

1.9.10 Recursive relationships

A *recursive relationship* is one in which a relationship can exist between occurrences of the same entity set. It is naturally found within unary relationships

1.9.11 Composite Entities

Relationships do not contain attributes; the relational model generally requires the use of one to many relationships. If a may to many relationship is encounter you have create a bridge entity between the entities displaying such relationships. The *bridge entity* also knows as a *composite entity* is composed of the primary keys of each of the entities to be connected. A bridge entity can also contain additional attributes that play no role in the connective process.

1.10 Developing An Entity Relationship Diagram

The process of database design is an iterative rather than linear or sequential process. An *iterative process* is one based on repetition of processes and procedures.

Building and entity relationship diagram usually involves the following activities:

- Create a detailed narrative of the organization's *description of operations*
- Identify the *business rules* based on the description of operations
- Identify the main *entities* and *relationships* from the business rules
- Develop the initial entity relationship diagram
- Identify the *attributes and primary keys* that adequately describe the entities.
- Revise and review the entity relationship diagram.

1.11 Database Design Challenges: Conflicting Goals

Database designers often must make design compromises that are triggered by conflicting goals such as:

- The database design must conform to design standards. Such standards have guided you in developing logical structures that minimize data redundancies, thereby minimizing the likelihood that destructive data anomalies will occur.
- In may organizations, particularly those generating large numbers of transactions, high processing speeds are often a top priority in database design. High processing speed means minimal access time, which may be achieved by minimizing the number and complexity of logically desirable relationships.

- The quest for timely information might be the focus of database design. Complex information requirements may dictate data transformations, and they may expand the number of entities and attributes within the design.

Chapter 5 : Normalization of Database Tables

1.12 Database Tables And Normalization

Normalization is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies. The normalization process involves assigning attributes to tables based on the concept of determination. Normalization works through a series of stages called normal forms. The first three stages are described as first normal form (1NF), second normal form (2NF), and third normal form (3NF). From a structural point of view second normal form is better than first normal form and third normal form is better than second normal form. The highest form of normalization is always the most desirable. *Denormalization* produces a lower normal form, that is, a third normal form will be converted to a second normal form through denormalization.

1.13 The Normalization Process

In normalization the objective is to create tables that have the following characteristics:

- Each table represents a single subject.
- No data item will be *unnecessarily* stored in more than one table. The reason for this requirement is to ensure that the data are updated in only one place
- All attributes in a table are dependent on the primary key, the entire primary key and nothing but the primary key.

1.13.1 Conversion to First Normal Form

A *repeating group* derives its name from the fact that a group of multiple entries of the same type can exist for any *single* key attribute occurrence. A relational table must not contain repeating groups as it is the cause of data redundancies. Normalizing the table structure will reduce the data redundancies.

The normalization process starts with a simple three-step procedure.

- **Step 1 : Eliminate the repeating groups**

  Start by presenting the data in a tabular format, where each cell has a single value and there are no repeating groups. To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value.

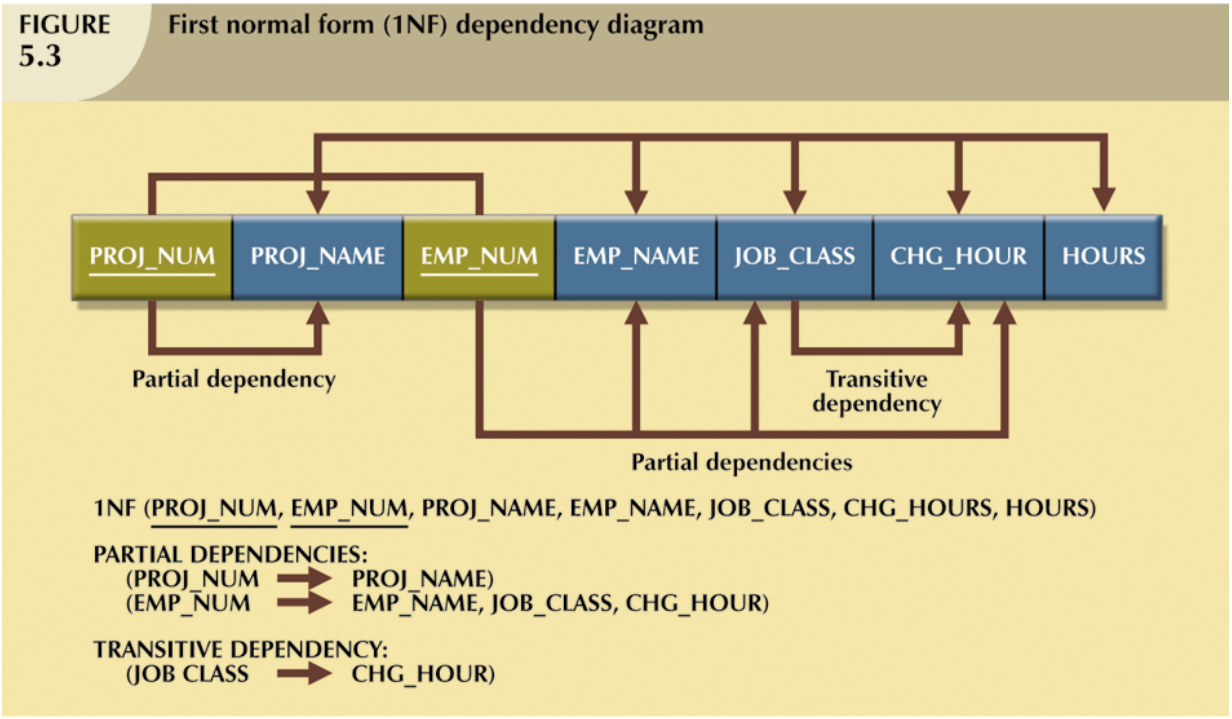
- **Step 2 : Identify the primary key**

The primary key must uniquely identify all of the remaining entity occurrences' attributes. The new primary key will most likely be composed out of several attributes.

- **Step 3 : Identify all dependencies**

  Dependencies can be depicted by the help of a diagram known as a *dependency diagram*. It can depict the all dependencies found within a given table structure and give you the bird's eye view of all of the relationships among a table's attributes and their use, making it less likely that you will overlook an important dependency.

FIGURE 5.3 First normal form (1NF) dependency diagram



Partial dependency is a dependency based on only a part of a composite primary key. While partial dependencies are sometimes used for performance reasons, they should be used with caution as they can still cause data redundancies. *Transitive dependency* is a dependency of one non prime attribute on another non prime attribute. Transitive dependencies can still yield data anomalies.

All relation tables satisfy the first normal form requirements

A table is in first normal form when:

- All of the key attributes are defined.
- There are no repeating groups in the table.
- All attributes are dependent on the primary key.

1.13.2 Conversion To Second Normal Form

The relational database design can be improved by converting the database into second normal form and is done in two steps:

- **Step 1 : Write each key component on a separate line**

  Write each key component on a separate line with the original key on the last line. Each component will become a key in a new table. In other words the initial composite key is now divided into tables.

- **Step 2 : Assign corresponding dependent attributes**

  Determine those attributes that are dependent on other attributes to build the new tables.

At this point most of the anomalies have been eliminated.

A table is in second normal form when:

- It is in first normal form
- It includes no partial dependencies, that is no attribute is dependant on only part of the primary key. It is possible that transitive dependencies still exist.

1.13.3 Conversion To Third Normal Form

Data anomalies created by the database organization are easily eliminated and the tables converted to third normal form by completing the following steps:

- **Step 1 : Identify each new determinant**

  For every transitive dependency, write the determinant as a parent key for a new table. If there are three different transitive dependencies, there are three different determinants.

- **Step 2 : Identify the dependant attributes**

  Identify the attributes that are dependent on each determinant identified in Step 1 and identify the dependency. Name the table to reflect its contents and function.

- **Step 3 : Remove the dependent attributes from transitive dependencies**

  Eliminate all dependent attributes in the transitive relationships from each o the tables that have such transitive relationship. Make sure each table has a determinant and that no table contains inappropriate dependencies.

A table is in third normal form when:

- It is in second normal form.
- It contains no transitive dependencies.

1.14 Improving The Design

The table structures are now cleared of all transitive and partial dependencies. Normalization by itself cannot be relied on to make good designs. Instead normalization is valuable because it eliminates data redundancies.

Issues to address in order to produce a good normalized set of tables:

- Evaluate primary key Assignments
- Evaluate Naming Conventions
- Refine Attribute Atomicity
- Identify New Attributes
- Identify New Relationships
- Refine Primary Keys as Required for Data Granularity
- Maintain Historical Accuracy
- Evaluate Using Derived Attributes

1.15 Surrogate Key Considerations

When a primary key is considered to be unsuitable, designers must use surrogate keys. At the implementation level, a surrogate key is a system defined attribute generally created and managed via the database management system. Usually a system defined surrogate key is numeric and its value is automatically incremented for each new row.

1.16 Higher Level Normal Forms

1.16.1 The Boyce-Codd Normal Form

A table is in *Boyce-Codd normal form* when every determinant in the table is a candidate key. When a table contains only one candidate key, the third normal form and Boyce-Codd normal form is equivalent. Boyce-Codd normal form can only be violated when the table contains more than one candidate key.

1.16.2 Fourth Normal Form

A table is in fourth normal form when it is in third normal form and has no multiple sets of multivalued dependencies.

The fourth normal form is largely academic if you make sure that your tables to the following two rules:

- All attributes must be dependent on the primary key, but they must be independent of each other
- No row may contain two or more multivalued facts about and entity.

1.17 Normalization and Database Design

Normalization should be part of the design process. Therefore, make sure that proposed entities meet the required normal form before the table structures are created. It is difficult to separate the normalization process from the entity relationship modeling process; the two techniques are use in an iterative and incremental process.

1.18 Denormalization

Although the creation of normalized relations is an important database design goal, it is only one of many such goals. Good database design also considers processing requirements. As tables are decomposed to conform to normalization requirements, the number of database tables expands. Joining the larger number of tables takes additional input output operations and processing logic, thereby reducing system speed.

The conflicts between design efficiency, in formation requirements and processing speed are often resolved through compromises that may include denormalization. Unnormalized tables in a production database tend to suffer from these defects:

- Data updates are less efficient because programs that read and update tables must deal with larger tables
- Indexing is more unmanageable. It simply is not practical to build all of the indexes required for the many attributes that may be located in a single unnormalized table.
- Unnormalized tables yield no simple strategies for creating virtual tables know as *views.*

Chapter 6 : Advanced Data Modeling

1.19 The Extended Entity Relationship Model

As the complexity of the data structure being modeled has increased and as application software requirements have become more stringent, there has been an increasing need to capture more information in the data model. The *extended entity relationship model* is the result of adding more semantic constructs to the original entity relationship model. A diagram using this model is called an extended entity relationship diagram.

1.19.1 Entity Supertypes and Subtypes

In modeling terms, an *entity supertype* is a generic entity type that is related to one or more *entity subtype*, where the entity supertype contains the common characteristics and the entity subtypes contain the unique characteristics of each entity subtype.

1.19.2 Specialization Hierarchy

Entity supertypes and subtypes are organized in a specialization hierarchy. The *specialization hierarchy* depicts the arrangement of higher-level entity supertypes and lower level entity subtypes.

The relationships depicted within the specialization hierarchy are sometimes described in terms of "IS A" relationships. A subtype can only exist within the context of a supertype and every subtype can have only one supertype to which it is directly related in the specialization hierarchy. However, a specialization hierarchy can have many levels of supertype and subtype relationships.

A specialization hierarchy provides the means to:

- Support attribute inheritance

- Define a special supertype attribute known as the subtype discriminator
- Define disjoint or overlapping constraints and complete or partial constraints.

1.19.3 Inheritance

The property of *inheritance* enables an entity subtype to inherit the attributes and relationships of the supertype. All entity subtypes inherit their primary key attribute from their supertype.

At implementation level, the supertype and its subtypes depicted in the specialization hierarchy maintain a one to one relationship.

1.19.4 Subtype Discriminator

A *subtype discriminator* is the attribute in the supertype entity that determines to which entity subtype each supertype occurrence is related. The default comparison condition for the subtype discriminator attribute is the equality comparison. However, there may be situations in which the subtype discriminator is not necessarily based on an equality comparison.

1.19.5 Disjoint and Overlapping Constraints

An entity supertype can have disjoint or overlapping entity subtypes. *Disjoint subtypes* also known as *non-overlapping subtypes*, are subtypes that contain a *unique* subset of the supertype entity set. In other words, each entity instance of the supertype can appear in only one of the subtypes.

Overlapping subtypes are subtypes that contain non unique subsets of the supertype entity set, that is, each entity instance of the supertype may appear in more than one subtype.

1.19.6 Completeness Constraint

The *completeness constraint* specifies whether each entity supertype occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total. *Partial completeness* means that not every supertype occurrence is a member of a subtype. *Total completeness* means that every supertype occurrence must be a member of at least one subtype.

1.19.7 Specialization and Generalization

Specialization is the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping unique characteristics and relationships of the subtypes.

Generalization is the bottom-up process of identifying a higher-level, more generic entity supertype from lower-level entity subtypes. Generalization is based on grouping common characteristics and relationships of the subtypes.

1.20 Entity Clustering

An *entity cluster* is a "virtual" entity type used to represent multiple entities and relationships in the entity relationship diagram. An entity cluster is formed by combining multiple interrelated entities into a single abstract entity object. An entity cluster is considered "virtual" or "abstract" in the sense that it is not actually an entity in the final

entity relationship diagram. The entity cluster is a temporary entity used to represent multiple entities and relationship with the purpose to simply the entity relationship diagram and thus enhancing its readability. Avoid the display of attributes when entity clusters are used.

1.21 Entity Integrity: Selecting Primary Keys

1.21.1 Natural Keys and Primary Keys

A *natural key* or *natural identifier* is a real-world generally accepted identifier used to distinguish real world objects. A data modeler uses a natural identifier as the primary key of the entity being modeled, assuming that the entity has a natural identifier.

1.21.2 Primary Key Guidelines

Desirable primary key characteristics are:

- **Unique values**. The primary key must uniquely identify each entity instance. A primary key must be able to guarantee unique values. It cannot contain nulls.
- **Nonintelligent**. The primary key should not have embedded semantic meaning.
- **No change over time**. If an attribute has semantic meaning, it may be subject to updates. Changing a primary key value means that you are basically changing the identity of an entity.
- **Preferably single attribute**. A primary key should have the minimum number of attributes possible. Single attribute primary keys are desirable but not required.
- **Preferably numeric**. Unique values can be better managed when they are numeric because the database can use internal routines to implement a "counter style" attribute that automatically increments values with the addition of each new row.
- **Security complaint.** The selected primary key must not be composed of any attributes that might be considered a security risk or violation.

1.21.3 When To Use Composite Primary Keys

Composite primary keys are particularly useful in two cases:

- As identifiers of composite entities, where each primary key combination is allowed only once in the many to many relationship
- As identifiers of weak entities, where the weak entity has a strong identifying relationship with the parent entity.

1.21.4 When To Use Surrogate Primary Keys

Surrogate primary keys are accepted practice in today's complex data environments. Surrogate primary keys are especially helpful when there is no natural key, when the selected candidate key has embedded semantic contents, or when the selected candidate key is too long or unmanageable. If you use a surrogate key, you must ensure that the candidate key of the entity in question performs properly through the use of "unique index" and "not null" constraints.

1.22 Design Cases: Learning Flexible Database Design

1.22.1 Implementing One To One Relationships

Foreign keys work with primary keys to properly implement relationships in the relation model. With one to one relationships there are two options for selecting and placing the foreign key. Either place a foreign key in both entities or place a foreign key in one of the entities.

1.22.2 Maintaining History of Time Variant Data

Time variant data refer to data whose values change over time and for which you *must* keep history of the data changes. To model time variant data, you must create a new entity in a one to many relationship with the original entity. The new entity will contain the new value, the date of the change, and whatever other attribute is pertinent to the event being modeled.

1.22.3 Fan Traps

A *design trap* occurs when a relationship is improperly or incompletely identified and, therefore, is represented in a way that is not consistent with real world. The most common design trap is a *fan trap*.

A *fan trap* occurs when you have one entity in two one to many relationships to other entities, thus producing association amount the other entities that is not expressed in the model.

1.22.4 Redundant Relationships

Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model.

1.23 Data Modeling Checklist

BUSINESS RULES

- Properly document and verify all business rules with the end users.
- Ensure that all business rules are written precisely, clearly, and simply. The business rules must help identify entities, attributes, relationships and constraints.
- Identify the source of all business rules and ensure that each business rule is accompanied by the reason for its existence and by the date and person responsible for the business rule verification and approval.

DATA MODELING

Naming Conventions

- Entity names:
  - Should be nouns that are familiar to business and should be short and meaningful
  - Should include abbreviations, synonyms, and aliases for each entity

    - Should be unique within the model
    - For composite entities, may include combination of abbreviated names of the entities linked through the composite entity.
- Attribute names:
    - Should be unique within the entity
    - Should use the entity abbreviation or prefix
    - Should be descriptive of the characteristic
    - Should use suffixes such as _ID for the primary key attribute
    - Should not be a reserved word
    - Should not contain spaces or special characters such as @,! or &
- Relationship names:
    - Should be active or passive verbs that clearly indicate the nature of the relationship

Entities

- All entities should represent a single subject
- All entities should be in third normal form or higher
- The granularity of the entity instance is clearly defined
- The primary key is clearly defined and supports the selected data granularity

Attributes

- Should be simple and single valued
- Should include default values, constraints, synonyms and aliases
- Derived attributes should be clearly identified and included source
- Should not be redundant unless they are required for transaction accuracy or for maintaining a history or are used as a foreign key

Relationships

- Should clearly identify relationship participants
- Should clearly define participation and cardinality rules

Entity Relationship Diagram

- Should be validated against expected processes: inserts, updates, and deletes

- Should evaluate where, when and how to maintain a history
- Should not contain redundant relationships except as required
- Should minimize data redundancy to ensure single place updates

Chapter 10 : Transaction Management and Concurrency Control

1.24 What Is A Transaction

In database terms, a *transaction* is any action that reads from and or writes to a database. A transaction may consist of a simple SELECT statement to generate a list of table contents; It may consist of a series of related UPDATE statements to change the values of attributes in various tables; it may consist of a series of INSERT statements to add rows to one or more tables; or it may consist of a combination of SELECT, UPDATE and INSERT statements.

A transaction is a logical unit of work that must be entirely and completed or entirely aborted; no intermediate states are acceptable. A successful transaction changes the database from one consistent state to another. A *consistent database state* is one in which all data integrity constraints are satisfied.

Most real world database transactions are formed by two or more database requests. A *database request* is the equivalent of a single SQL statement in an application program or transaction. Therefore, if a transaction is composed of two UPDATE statements and one INSERT statement, the transaction uses three database requests.

1.24.1 Evaluating Transaction Results

Not all transactions update the database. SQL code represents a transaction because it *accesses* the database. Improper or incomplete transaction can have a devastating effect on database integrity. Some database management systems provide means by which the user can define enforceable constraints based on business rules. Other integrity rules, such as those governing referential and entity integrity, are enforced automatically by the database management system.

1.24.2 Transaction Properties

All transactions must display *atomicity, consistency, isolation, durability,* and *serializabitlity.* These properties are sometimes referred to as the ACIDS test

- **Atomicity** requires that all operations of a transaction be completed, if not the transaction is aborted.
- **Consistency** indicates the permanence of the database's consistent state. When a transaction is completed, the database reaches a consistent state.
- **Isolation** means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed.
- **Durability** ensures that once transaction changes are done, it cannot be undone or lost, even in the event of a system failure.

- **Serializability** ensures that the concurrent execution of several transaction yields consistent results.

1.24.3 Transaction Management With SQL

The American National Standards Institute has defined standards that govern SQL database transactions. Transactions support is provided by two SQL statements: COMMIT and ROLLBACK. The standard requires that when a transaction sequence is initiated by a user or an application program, the sequence must continue through all succeeding SQL statements until on of the following events occur:

- A COMMIT statement is reached and all changes are permanently recorded within the database.
- A ROLLBACK statement is reached and all changes are aborted and the database is rolled back to its previous consistent state.
- The end of a program is successfully reached and all changes are permanently recorded within the database. It is equivalent to a COMMIT.
- The program is abnormally terminated and all changes are aborted and the database is rolled back to its previous consistent state. It is equivalent to a ROLLBACK.

1.24.4 The Transaction Log

A database management system uses a *transaction log* to keep track of all transaction that update the database. The information stored in this log is used by the database management system for a recovery requirement triggered by a ROLLBACK statement, a programs abnormal termination, or a system failure.

The transaction log stores:

- A record for the beginning of the transaction
- For each transaction component or SQL statement
  - The type of operation being performed, UPDATE, DELETE or INSERT
  - The names of the objects or tables affected by the transaction
  - The “before” and “after” values for the fields being updated
  - Pointers to the previous and next transaction log entries for the same transaction
- The ending or COMMIT of the transaction

1.25 Concurrency Control

The coordination of the simultaneous execution of transaction in a multiuser database system is known as *concurrency control*. The objective of concurrency control is to ensure the serializability of transactions in a multiuser database environment. The simultaneous execution of transactions over a shared database can create several data integrity and consistency problems of which the main three are lost updates, uncommitted data and inconsistent retrievals.

1.25.1 The Scheduler

The *scheduler* is a special database management system program that establishes the order in which the operation within concurrent transactions are executed. The scheduler interleaves the execution of database operations to ensure serializability and isolation of transaction. The scheduler basis its action on concurrency control algorithms, such as locking or time stamping methods. It makes sure that the computers central processing unit is used efficiently and facilitates data isolation to ensure that two transactions do not update the same data element at the same time.

1.26 Concurrency Control With Locking Methods

A *lock* guarantees exclusive use of a data item to a current transaction. A transaction acquires a lock prior to data access and the lock is released when the transaction is completed. All lock information is managed by a *lock manager*, which is responsible for assigning and policing the locks used by the transactions.

1.26.1 Lock Granularity

Lock granularity indicates the level of lock use. Locking can take place at the following levels:

- **Database Level.** In a *database level lock* the entire database is locked, thus preventing the use of any tables in the database by a transaction.
- **Table Level.** In a *table level lock* the entire table is locked, preventing access to any row by a transaction.
- **Page Level.** In a *page level lock* the database management system will lock an entire diskpage.
- **Row Level.** A *row level lock* allows concurrent transactions to access different rows of the same table even when the rows are located on the same page.
- **Field Level.** The *field level lock* allows concurrent transaction to access the same row as long as they require the use of a different field or attribute within that row.

1.26.2 Lock Types

Regardless of the level of locking different lock types can be used:

- **Binary Locks**. A *binary lock* has only two states locked (1) or unlocked (0)
- **Exclusive Locks**. An *exclusive lock* exists when access is reserved specifically for the transaction that locked the object.
- **Shared Lock.** A *shared lock* exists when concurrent transactions are granted read access on the basis of a common lock.

1.26.3 Two Phase Locking To Ensure Serializability

Two phase locking defines how transactions acquire and relinquish locks. Two phase locking guarantees serializability but does not prevent deadlocks. The two phases are:

- A growing phase, in which a transaction acquires all required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
- A shrinking phase, in which the transaction releases all locks and cannot obtain any new locks.

The two phase locking protocol is governed by the following rules:

- Two transaction cannot have conflicting locks
- No unlock operation can precede a lock operation in the same transaction
- No data are affected until all locks are obtained, that is, until the transaction is in its locked point.

1.26.4 Deadlocks

A database *deadlock* is caused when two transactions wait for each other to unlock data. Deadlocks are possible only when one of the transactions wants to obtain an exclusive lock on a data item, no deadlocks can exist among shared locks.

The three basic techniques to control deadlocks are:

- **Prevention.** A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur.
- **Detection.** The database management system periodically tests the database for deadlocks.
- **Avoidance.** The transaction must obtain all of the locks it needs before it can be executed.

1.27 Concurrency Control With Time Stamping Methods

The *time stamping* approach to scheduling concurrent transactions assigns a global, unique time stamp to each transaction. The time stamp value produces an explicit order in which transactions are submitted to the database management system. Time stamps must have two properties. **Uniqueness** to ensure that no equal time stamp values can exist and **Monotonicity** to ensure that time stamp values always increase.

1.27.1 Wait Die and Wound Wait Schemes

In a *wait die* scheme the older transaction waits and the younger is rolled back and rescheduled. In a *wound wait* scheme the older transaction rolls back the younger transaction and reschedules it.

1.28 Concurrency Control With Optimistic Methods

The *optimistic method* is based on the assumption that the majority of the database operations do not conflict. The optimistic approach does not require locking or time stamping techniques. Instead, a transaction is executed without restrictions until it is

committed. Using the optimistic approach each transaction moves through two or three phases:

- **Read phase.** The transaction reads the database, executes the needed computations and makes the updates to a private copy of the database values. All update operations of the transaction are recorded in a temporary update file, which is not accessed by the remaining transactions.
- **Validation phase.** The transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database.
- **Write phase.** The changes are permanently applied to the database.

1.29 Database Recovery Management

Database recovery restores a database from a given state, usually inconsistent, to a previously consistent state. Recovery techniques are based on the *atomic transaction property*, all portions of transaction must be treated as single logical unit of work, so all operations must be applied and completed to produce consistent database. If a transaction operation cannot be completed, the transaction must be aborted, and any changes to database must be rolled back.

1.29.1 Transaction Recovery

Four important concepts that affect the recovery process:

- **Write-ahead-log protocol**. This protocol ensures that transaction logs are always written before any database data are actually updated.
- **Redundant transaction logs.** Most database management systems keep several copies of the transaction log to ensure that a physical disk failure will not impair the database management systems ability to recover data.
- **Buffers.** A buffer is a temporary storage area in primary memory used to speed up disk operations.
- **Checkpoints.** A database checkpoint is an operation in which the database management system writes all of its updated buffers to disk.

When the recovery procedures uses *deferred write* or *deferred update*, the transaction operations do not immediately update the physical database. The database is physically updated only after the transaction reaches its commit point, using the transaction log information.

When the recovery procedure uses *write-through* or *immediate update* the database is immediately updated by transaction operations during the transactions execution, even before the transaction reaches its commit point.

Chapter 12 : Distributed Database Management Systems

1.30 The Evolution of Distributed Database Management Systems

A *distributed database management system* governs the storage and processing of logically related data over interconnected computer systems in which both data and processing functions are distributed among sever sites.

The dynamic business environment and the centralized database's shortcomings spawned a demand for applications based on accessing data from different sources at multiple locations. These database environments are then managed by distributed database management systems.

1.31 Advantages And Disadvantages

Advantages of a distributed database management system:

- Data are located near the greatest demand site.
- Faster data access.
- Faster data processing speed.
- Growth facilitation.
- Improved communications.
- Reduced operating costs.
- User-friendly interface.
- Less danger of a single-point failure.
- Processor independence.

Disadvantages of a distributed database management system:

- Complexity of management and control.
- Security.
- Lack of standards.
- Increased storage requirements.
- Increased training cost.

1.32 Distributed Processing and Distributed Databases

In *distributed processing* a databases logical processing is shared among two or more physically independent sites that are connected through a network.

A *distributed database* stores a logically related database over two or more physically independent sites. The sites are connected via a computer network.

1.33 Characteristics of Distributed Database Management Systems

A distributed database management system must at least have the following functions:

- *Application interface* to interact with the end user or application programs and with other database management systems within the distributed database.
- *Validation* to analyze data requests.
- *Transformation* to determine which data request components are distributed and which are local.
- *Query optimization* to find the best access strategy.
- *Mapping* to determine the data location of local and remote fragments
- *Input output interface* to read or write data from or to permanent local storage.
- *Formatting* to prepare the data for presentation to the end user or to an application program.
- *Security* to provide data privacy at both local and remote databases.
- *Backup and recovery* to ensure the availability and recoverability of the database in case of failure.
- *Database administration* features for the database administrator.
- *Concurrency control* to manage simultaneous data access and to ensure data consistency across database fragments in the distributed database management system.
- *Transaction management* to ensure that the data mover from one consistent state to another. This activity includes the synchronization of local and remote transactions as well as transactions across multiple distributed segments.

A distributed database management system must perform all of the functions of a centralized database management system. In addition it must handle all necessary function imposed by the distribution of data and processing. And perform those additional functions transparently to the end user.

1.34 Distributed Database Management System Components

A distributed database management system must include at least the following components:

- *Computer workstations* that form the network system. The distributed database system must be independent of the computer system hardware.
- *Network hardware and software* components that reside in each workstation. The network components allow all sites to interact and exchange data.
- *Communications media* that carry the data from one workstation to another. The distributed database management system must be communications media independent.

- The *transaction processor*, which is the software component found in each computer that requests data. The transaction processor receives and processes the applications data request. Also known as the *application processor* or the *transaction manager*.

- The *data processor* which is the software component residing in each computer that stores and retrieves data located at the site. Also known as the *data manager.*

1.35 Levels of Data and Process Distribution

1.35.1 Single-Site Processing, Single-Site Data

In the *single- site processing, single-site data* scenario, all processing is done on a single CPU or host computer and all data are stored on the host computers local disk. Processing cannot be done on the end users side of the system. This scenario is typical of most mainframe and midrange computer database management systems. The database is located on a host computer which is accessed by dumb terminals connected to it. The scenario is also typical of the first generation single-user microcomputer databases.

1.35.2 Multiple-Site Processing, Singe-Site Data

Under the *multiple-site processing, single-site data* scenario, multiple processes run on different computers sharing a single data repository. This scenario requires a network file server running conventional applications that are accessed through a LAN.

1.35.3 Multiple-Site Processing, Multiple-Site Data

The *multiple-site processing, multiple-site data* scenario describes a fully distributed database management system with support for multiple data processors and transactions processors at multiple sites.

Distributed Database management systems are classified as either:

- **Homogeneous.** Which integrates only one type of centralized database management system over a network.

- **Heterogeneous.** Which integrates different types of centralized database management systems over a network.

- **Fully heterogeneous.** Which supports different database management systems that may even support different data models running under different computer systems.

1.36 Distributed Database Transparency Features

Transparency features have the common property of allowing the end user to feel like the databases only user.

Transparency features include:

- **Distribution transparency**, which allows a distributed database to be treated as a single logical database.

- **Transaction transparency**, which allows a transaction to update data at several network sites.

- **Failure integrity**, which ensures that the system will continue to operate in the event of a node failure.
- **Performance transparency**, which allows the system to perform as if it were a centralized database management system.
- *Heterogeneity transparency,* which allows the integration of several different local database management systems under a common or global schema.

1.37 Distribution Transparency

Allows a physically dispersed database to be managed as though it were a centralized database.

Three levels o distribution transparency are recognized:

- **Fragmentation transparency** is the highest level of transparency. The end user or programmer does not need to know that the database is partitioned. Neither fragment names, nor locations are specified.
- **Location transparency** exists when the end user or programmer must specify the database fragment names but does not need to specify where those fragments are located.
- **Local mapping transparency** exists when the end user or programmer must specify both the fragment names and their locations.

1.38 Transaction Transparency

Is a distributed database management system property that ensures that the database transactions will maintain the distributed databases integrity and consistency.

1.38.1 Distributed Requests and Distributed Transactions

The basic difference between a nondistributed transactions and distributed transactions is that the latter can update or request data from several different remote sites on a network.

A *remote request* lets a single SQL statement access the data that is to be processed by a single remote database processor.

A *remote transaction* composed of several requests, accesses data at a single remote site.

A *distributed transaction* allows a transaction to reference several different local or remote database processor sites.

A *distributed request* lets a single SQL statement reference data located at several different local or remote database processor sites.

1.38.2 Distributed Concurrency Control

Concurrency control becomes especially important in the distributed database environment because multisite, multiple-process operations are more likely to create data inconsistencies and deadlocked transactions than single-site systems are.

1.38.3 Two-Phase Commit Protocol

Distributed databases make it possible for a transaction to access data at sever sites. A final COMMIT must not be issued until all sites have committed their parts of the transaction.

The two-phase commit protocol requires that the transaction entry log for each database processor be written before the database fragment is actually updated.

1.39 Performance Transparency and Query Optimization

The objective of a query optimization routine is to minimize the total cost associated with the execution of a request.

The costs associated with a request are a function of the:

- Access time cost involved in accessing the physical data stored on disk
- Communication cost associated with the transmission of data among nodes in distributed database systems.
- CPU time cost associated with the processing overhead of managing distributed transactions.

Query optimization must provide distribution transparency as well as *replica transparency*. Replica transparency refers to the distributed database management systems ability to hide the existence of multiple copies of data from the user.

Most of the algorithms proposed for query optimization are based on two principles:

- The selection of the optimum execution order
- The selection of sites to be accessed to minimize communication costs

Within those two principles, a query optimization algorithm can be evaluated on the basis of its *operation mode* or the *timing of its optimizations*.

Operation modes can be classified as manual or automatic. *Automatic query optimization* means that the distributed database management system finds the most cost-effective access path without user intervention. *Manual query optimization* requires that the optimization be selected and scheduled by the end user or programmer.

Within the timing classification, query optimization can be static or dynamic. *Static query optimization* takes place at compilation time. *Dynamic query optimization* takes place at execution time.

Query optimization techniques can be classified according to the type of information that is used to optimize the query, it can be either statistically based or rule-based. *Statistically based query optimization algorithm* uses statistical information about the database to determine the best access strategy. *Rule-based query optimization algorithm* is based on a set of user-defined rules to determine the best query access strategy.

1.40 Distributed Database Design

1.40.1 Data Fragmentation

Data fragmentation allows you to break a single object into two or more segments or fragments. Each fragment can be stored at any site over a computer network. Information about the fragmentation is stored in the distributed data catalog, from which it is accessed by the transaction processor to process user requests.

Data fragmentation strategies are based at the table level and consist of dividing a table into logical fragments. Three types of such data fragmentation are:

- **Horizontal fragmentation** refers to the division of a relation into subsets of rows.
- **Vertical fragmentation** refers to the division of a relation into attribute subsets.
- **Mixed fragmentation** refers to a combination of horizontal and vertical strategies.

1.40.2 Data Replication

Data replication refers to the storage of data copies at multiple sites served by a computer network. Fragment copies can be stored at several sites to serve specific information requirements. Because the existence of fragmentation copies can enhance data availability and response time, data copies can help to reduce communication and total query costs.

Replicated data are subject to the mutual consistency rule. The *mutual consistency rule* requires that all copies of data fragments be identical.

Three replication scenarios exist:

- A **fully replicated database** stores multiple copies of each database fragment at multiple sites. It can be impractical due to the amount of overhead it imposes.
- A **partially replicated database** stores multiple copies of some database fragments at multiple sites. It is handled well by most databases.
- An **unreplicated database** stores each database fragment at a single site.

1.40.3 Data Allocation

Data allocation describes the process of deciding where to locate data.

Data allocation strategies are as follows:

- With **centralized data allocation**, the entire database is stored at one site
- With **partitioned data allocation**, the database is divided into several disjointed parts and stored at several sites.
- With **replicated allocation**, copies of one or more database fragments are stored at several sites.

Data distribution over a computer network is achieved through data partition, through data replication, or through a combination of both.

1.41 Client Server versus Distributed Database Management System

Client server architecture refers to the way in which computers interact to form a system. It features users of resources, or a client, and a provider of resources, or a server. It can be used to implement the database management system in which the client is the transaction processor and the server is the database processor.

Client server application advantages:

- Client server solutions tend to be less expensive than alternate minicomputer or mainframe solutions
- Client server solutions allow the end user to use the microcomputers interface thereby improving functionality and simplicity
- More people in the job market have computer skills than mainframe skills
- The computer is well established in the workplace
- Numerous data analysis and query tools exist to facilitate interaction with many of the database management systems that are available in the computer market
- There is a considerable cost advantage to offloading applications development from the mainframe to powerful computers.

Client server application disadvantages:

- The client server architecture creates a more complex environment in which different platforms are often difficult to manage.
- An increase in the number of users and processing sites often paves the way for security problems.
- The client server environment makes it possible to spread data access to a much wider circle of users, which increases the demand for people with a broad knowledge of computers and software applications and the burden of training.

1.42 C.J Dates Twelve Commandments For Distributed Databases

1) Local site independence
2) Central site independence
3) Failure independence
4) Location transparency
5) Fragmentation transparency
6) Replication transparency
7) Distributed query processing
8) Distributed transaction processing

9) Hardware independence

10) Operating system independence

11) Network independence

12) Database independence

Chapter 14 : Database Connectivity and Web Development

1.43 Database Connectivity

Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. Database connectivity software is also knows as *database middleware* because it interfaces between the application program and the database. The data repository, also known as the *data source*, represents the data management application that will be used to store the data generated by the application program.

1.43.1 Native SQL Connectivity

Native SQL connectivity refers to the connection interface that is provided by the database vendor and is unique to that vendor. The best example of that type of native interface is the Oracle RDBMS. To connect a client application to an Oracle database, you must install and configure the Oracle's SQL Net interface in the client computer.

1.43.2 Open Database Connectivity and Data or Remote Access Objects

Open Database Connectivity is Microsoft's implementation of a superset of the SQL Access Group Call Level Interface standard for database access. Open Database Connectivity is probably the most widely supported database connectivity interface. It allows any Windows application to access relational data sources, using SQL via a standard application programming interface.

Data Access Objects is an object-oriented application programming interface used to access MS Access, MS FoxPro and dBase databases from Visual Basic programs. It provided an optimized interface that exposed the functionality of the Jet data engine to programmers. The interface can also be used to access other relation style data sources.

Remote Data Objects is a higher-level object-oriented application interface used to access remote database servers. It uses the lower-level database access object and open database connectivity for direct access to databases. The interface was optimized to deal with server based databases, such as MS SQL Server and Oracle.

The basis open database connectivity architecture has three main components:

- A high-level open database connectivity application programming interface through which application programs access open database connectivity functionality.
- A driver manager that is in charge of managing all database connections.

- An open database connectivity driver that communicates directly to the database management system.

1.43.3 Object Linking and Embedding for Database

Object Linking and Embedding for Database is middleware that adds object-oriented functionality for access to relational and nonrelational data. It is composed of series of component objects models that provide low-level database connectivity for applications.

The Object Linking and Embedding for Database model is better understood when you divide its functionality into following types of objects:

- *Consumers* are objects that request and use data.
- *Providers* are objects that manage the connection with a data source an provide data to the consumers.

Object Linking and Embedding for Database did not provide support for scripting languages.

1.43.4 ADO.NET

ADO.NET is data access component of Microsoft's .NET application development framework. It introduced two new features critical for development of distributed applications, DataSets and XML support. A DataSet is disconnected memory-resident representation of database. DataSet is internally stored in XML format and data in DataSet can be made persistent as XML documents.

ADO.NET comes with two standard data providers:

- Data provider for OLE-DB data sources.
- Data Provider for SQL Server.

No matter data provider, it must support set of specific objects in order to manipulate data in data source :

- Connection
- Command
- DataReader
- DataAdapter
- DataSet
- DataTable

1.44 Internet Databases

Web database connectivity allows new innovative services that:

- Permit rapid response to competitive pressures by bringing new services and products to market quickly

- Increase customer satisfaction through creation of Web-based support services
- Yield fast and effective information dissemination through universal access from across street or across globe

1.44.1 Web-To-Database Middleware: Server-Side Extensions

Also known as Web-to-database middleware. It is a program that interacts directly with Web server to handle specific types of requests. It provides its services to Web server in a way that is totally transparent to client browser.

1.44.2 Web Server Interfaces

Two well-defined Web server interfaces:

- Common Gateway Interface (CGI)
- Application programming interface (API)

1.44.3 The Web Browser

Web browser is software that lets users navigate or browse the Web. It is located in a client computer and the end-user uses it to interface to World Wide Web. It interprets HTML code received from Web server and presents different page components in a standard way.

1.44.4 Client-Side Extensions

Client side extensions adds functionality to Web browser.

Although client side extensions are available in various fomrs, the most commonly encountered extentions are:

- Plug-ins
- Java and JavaScript
- ActiveX and VBScript

1.45 Using A Web-To-Database Production Tool : Coldfusion

A Web application server is a middleware application that expands Web server functionality by linking it to wide range of services. It provides a consistent run-time environment for Web applications.

ColdFusion application middleware can be used to:

- Connect to and query database from Web page
- Present database data in Web page, using various formats
- Create dynamic Web search pages
- Create Web pages to insert, update, and delete database data
- Define required and optional relationships

- Define required and optional form fields
- Enforce referential integrity in form fields
- Use simple and nested queries and form select fields to represent business rules

1.45.1 The Web As a Stateless System

Stateless system indicates that at any given time, Web server does not know status of any of clients communicating with it. Client and server computers interact in very short "conversations" that follow request-reply model.