

## Chapter Questions

### Chapter 1 Introduction

1. What are the effects of poor database design?
  - Unnecessary duplicated data
  - Inconsistent data
  - Data insertion problems
  - Data deletion problems
  - Use of meaningful identifies, which are subject to change

### Chapter 2 Entities and Data Relationships

1. Describe an entity, using the following terms: attributes, instance, identifier  
An *entity* is something about which we store data. The data that describes an entity are called its *attributes*. In a database, we actually store only the attributes of an entity. Each group of attributes that describes a single real-world occurrence of an entity acts to represent an *instance* of the entity. Each entity has some attribute values that distinguish it from every other entity in the database. These distinguishing attributes are called the *entity identifier* and are what allow the data to be retrieved from the database.
2. How should a multi-valued attribute be handled?  
An entity in a relational database cannot have multivalued attributes, so a new entity must be created to hold them. Create an entity of which you can store multiple instances, one for each value of the attribute.
3. What is a domain? What are the domain choices?  
The domain is an expression of the permissible values for a given attribute. The use of domain constraints within a Database Management System assures that we are getting data of the right type. Domains are usually stored in a document called a data dictionary.

The domain datatype choices are:

**CHAR:** a fixed-length string of text, usually up to 256 characters

**VARCHAR:** a variable-length string of text, usually up to 256 characters.

**INT:** An integer, the size of which varies depending on the operating system.

**DECIMAL** and **NUMERIC:** Real numbers, with fractional portions to the right of the decimal point. Must specify how many digits the number can contain (including the decimal point) and how many digits should be to the right of the decimal point (the precision).

**DATE:** a date

**TIME:** a time

**DATETIME:** the combination of a date and a time

**BOOLEAN:** a logical value (true or false)

4. Describe the following data relationships, with simple examples: one-to-one, one-to-many, many-to-many, mandatory

**One-to-one:** Given two entities (A and B), then a one-to-one relationship exists if at all times A is related to no instances of entity B or one instance of entity B, and B is related to no instances of entity A or one instance of entity A. Example: small town airport. The airport is located in one and only one town and the town contains one and only one airport.

**One-to-many:** Given instances of two entities (A and B), then a one-to-many relationship exists between two instances ( $A_i$  and  $B_j$ ) if  $A_i$  is related to zero, one, or more instance of entity B and  $B_j$  is related to zero or one instance of entity A. Example: a woman may have zero, one, or more biological daughters. A daughter has only one biological mother.

**Many-to-many:** a many-to-many relationship exists between entites A and B if for two instances of those entities ( $A_i$  and  $B_j$ ),  $A_i$  can be related to zero, one, or more instances of B and  $B_j$  can be related to zero, one, or more instances of entity A. Example: A customer order can contain multiple items; each item can appear on more than one order.

**Mandatory relationship:** the participation of a given instance of an entity in a relationship is not optional. Example: A customer can exist before an order; however without a customer an order can not exist.

5. On a separate page draw and label an example entity relationship diagram using the Chen Method. Repeat with the Information Engineering Method. Provide symbol keys.
6. How are the problems of many-to-many relationships solved?  
The many-to-many relationships must be replaced with a collection of one-to-many relationships. An entity known as a composite entity can be created to represent the relationship between two other entities.
7. Briefly describe a data model vs. a data flow.  
*A data flow* shows how data are handled within an organization, including who handles the data, where the data is stored, and what is done to the data.  
  
*A data model* depicts the internal, logical relationships between data, without regard to who is handling the data and what is being done with them. It contains information about the data being stored in a database (entities, attributes, and entity relationships).
8. What is a schema?  
A schema is the logical plan of a database. It can be represented by a completed entity-relationship diagram. The underlying physical storage, which is managed by the DBMS, is known as the physical schema.

### **Chapter 3 The Relational Data Model**

1. Describe the characteristics of columns and rows in a relation.  
The columns (attributes) of a relation have the following properties:
  - A name that is unique within the table
  - A domain: values are subject to a domain constraintThe rows (tuples) of a relation have the following properties:
  - Only one value at the intersection of a column and row: no multivalued attributes
  - Uniqueness: no duplicate rows
  - A primary key: a column or combination of columns that uniquely identifies each row
2. What are the characteristics of a good primary key?
  - It must be unique.

- It must not contain a null value.
- It should never change.
- It should avoid using meaningful data. Use arbitrary identifiers or concatenations of arbitrary identifiers.
- A concatenated primary key should be made up of the smallest number of columns necessary to ensure its uniqueness.

3. How do foreign keys and primary keys ensure referential integrity?

When a table contains a column that is the same as the primary key of a table, the column is called a foreign key. The matching of foreign keys to primary keys represents data relationships in a relational database. A relational DBMS uses the relationships indicated by matching data between primary and foreign keys. To ensure the consistency of the cross-references among tables the DBMS enforces the following constraint:

Every nonnull foreign key value must match an existing primary key value.

4. What are the advantages of using views?

- They provided a security mechanism by restricting users from viewing portions of a schema to which they should not have access.
- They can simplify the design of a database for technologically unsophisticated users.
- They can be used to store frequently used, complex queries which can then be executed by using the name of the view in a simple query.

5. What is found in the data dictionary? How is it used by a DBMS?

- Definitions of the columns that make up each table.
- Integrity constraints placed on relations.
- Security information (which user has the right to perform which operation of which table).
- Definitions of other database structural elements, such as views and user-defined domains.

Relational DBMS's are data dictionary driven. A DBMS uses the data dictionary to determine whether the database elements the user has requested are actually part of the schema and verifies that the user has the access rights to whatever he or she is requesting.

The DBMS also use the data dictionary to check integrity constraints before a data modification is permitted.

## Chapter 4 Normalization

1. List the steps for translating a properly constructed ER diagram into a set of relations:
  - Create one table for each entity.
  - For each entity that is only at the “one” end of one or more relationships, and not at the “many” end of any relationship, create a single-column primary key, using an arbitrary unique number if no natural primary key is available
  - For each entity that is at the “many” end of one or more relationships, include the primary key of each parent entity (those at the “one” end of the relationships) in the table as foreign keys.
  - If an entity at the “many” end of one or more relationships has a natural primary key (for example, an order or invoice number), use that single column as the primary key. Otherwise, concatenate the primary key of its parent or parents with any other column or columns need for uniqueness to form the table’s primary key.
  
2. a.) Describe First Normal Form (1NF). b.) What is a repeating group and how is it handled? c.) What are the problems with First Normal Form?
  - a) “In First Normal Form the data are stored in a two-dimensional table with no repeating groups.”
  
  - b) A repeating group is an attribute that has more than one value in each row (analogous to a multivalued attribute). A repeating group is handled by creating another table to handle multiple instance of the repeating group.
  
  - c) The problems with First Normal Form include insertion anomalies ( a situation that arises when you are prevented from inserting data into a relation because a complete primary key is not available), deletion

anomalies (loss of data that you would like to keep), and update anomalies (inconsistent data due to unnecessary duplicated data).

3. a.) Define Second Normal Form (2NF). b.) What is functional dependency? c.) What are the problems with Second Normal Form?
  - a) “The relation is in first normal form and all nonkey attributes are functionally dependent on the entire primary key.”
  - b) A functional dependency is a one-way relationship between two attributes such that at any given time, for each unique value of attribute A, only one value of attribute B is associated with it through the relation. The *determinant* is an attribute that determines the value of other attributes. Each determinant becomes the primary key of a relation. All attributes that are functionally dependent upon it become nonkey attributes in the relation.
  - c) Tables in Second Normal Form can still exhibit anomalies.
4. a.) Define Third Normal Form (3NF) and transitive dependency. b.) How is a transitive dependency removed? c.) What is a candidate key?
  - a) “The relation is in second normal form and there are no transitive dependencies.”

A transitive dependency exists when you have the following functional dependency pattern:

$A \rightarrow B$  and  $B \rightarrow C$  therefore  $A \rightarrow C$

- b) A transitive dependency exists only when the determinant that is not the primary key is not a candidate key for the relation. Break the relation into two smaller relations, each of which has one of the determinants in the transitive dependency as its primary key. The attributes determined by the determinants become the nonkey attributes in each relation, removing the transitive dependency and placing the relations in third normal form.
5. a.) Which normal form is a good design objective for most relations?
    - b.) Define the Boyce-Codd Normal Form and Fourth Normal Form.
    - c.) What is a multi-valued dependency?

a) Relations in 3NF are free of most anomalies. The BCNF and 4NF handle special cases that are not free of anomalies in 3NF.

b) Boyce-Codd Normal Form:

“The relation is in third normal form and all determinants are candidate keys.”

4NF:

“The relations is in Boyce-Codd normal form and there are no multivalued dependencies.”

c) A multivalued dependency exists when for each value of attribute A, there exists a finite set of values of attribute B that are associated with it and a finite set of values of attribute C that are also associated with it. Attributes B and C are independent of each other. To eliminate the multivalued dependency and bring the relation into 4NF, split the relation, placing each part of the dependency in its own relation.

6. Theoretically, what is a join?

An algebraic operation which looks for rows with matching values between two tables and creates a new row in a result table every time it finds a match.

## **Chapter 5 Database Structure and Performance Tuning**

1. What are the advantages and disadvantages of indexing?

Advantages: Indexing provides a fast access path to the values in a column or a concatenation of columns. The index contains an ordered list of keys, allowing the DBMS to go directly to the correct row or rows, without having to sequentially scan the base table's rows.

Using an index enforces uniqueness of primary keys and can speed data access.

Disadvantages: Indexes take up space in the database. The DBMS must update the index as well as the base table for data modification operations (insert, modify or delete), which may slow down performance.

2. What is clustering? What do you need to consider before clustering? Clustering is the process of getting data that are frequently accessed together stored on the same disk page (or on pages that are physically close together). A cluster is designed to keep rows related by matching primary and foreign keys together. To define the cluster, you specify a column or columns on which the DBMS should form the cluster and the tables that should be included. Then, all rows that share the same value of the column or columns on which the cluster is based are stored as physically close together as possible.

Considerations:

- Because clustering involves physical placement of data in a file, a table can be clustered on only one column or combination of columns.
  - Clustering can slow down performance of operations that require a scan of the entire table because clustering may mean that the rows of any given table are scattered throughout many disk pages.
  - Clustering can slow down inserting data.
  - Clustering can slow down modifying data in the columns on which clustering is based.
3. What is partitioning? Describe horizontal and vertical partitioning and their advantages.

Partitioning involves the splitting of large tables into smaller ones so that the DBMS does not need to retrieve as much data at any one time.

Horizontal partitioning involves creating two or more tables with exactly the same structure and splitting rows between those tables. Horizontal partitioning may be advantageous if there is a group of rows that are accessed together significantly more frequently than the rest of the rows in a table.

Vertical partitioning involves creating two or more tables with selected columns and all rows of a table. Vertical partitioning makes sense only when there is a highly skewed access pattern for the columns of a table. The more often a small, specific group of columns is accessed together, the more vertical partitioning will help.

## Chapter 6 Codd's Rules

1. List each of Codd's rules and discuss any implications for database design and performance:

### **Rule 1: The Information Rule:**

“All information in a relational database is represented explicitly at the logical level in exactly one way — by values in tables.”

In other words: “Relations (2 dimensional tables) should be the only data structure in a relational database. No hard-coded links between tables.”

Why this is a good idea:

- Logical relationships are very flexible.
- Relational database schemas are very flexible.

### **Rule 2: The Guaranteed Access Rule**

“Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.”

In other words: “You should need to know only three things to locate a specific piece of data: the name of the table, the name of the column, and the primary key of the row.”

What does this mean: Primary key values must be unique.

### **Rule 3: Systematic Treatment of Null Values**

“Null values (distinct from the empty character string or string of blank characters or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.”

In other words: “A relational DBMS must store the same value of null in all columns and rows where the user does not explicitly enter data values. The DBMS must have some consistent, known way of handling those nulls when performing queries.”

Why is this important: nulls provide a consistent way to distinguish between valid data such as a 0 and missing data.

### **Rule 4: Dynamic Online Catalog Based on the Relational Model**

“The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the

same relational language to its interrogation as they apply to regular data.”

In other words: “The data dictionary should be made up of nothing but tables and you need learn only one query language to access all elements of the database. ”

Select appropriate DBMS software.

### **Rule 5: The Comprehensive Data Sublanguage Rule**

“ A relational system may support several languages and various modes of terminal use (for example, fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

- Data definition
- View definition
- Data manipulation (interactive and by program)
- Integrity constraints
- Transaction boundaries (begin, commit and rollback)”

SQL-92 meets all of these rules. Some graphic languages might functionally meet these rules.

### **Rule 6: The View Updating Rule**

“All views that are theoretically updatable are also updatable by the system.”

In other words: “If a view meets criteria for updatability, a DBMS must be able to handle that update and propagate the updates back to the base tables.”

### **Rule 7: High-Level Insert, Update, and Delete**

“The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.”

In other words: “A DBMS’s data manipulation language must be able to insert, update, and delete more than one row with a single command.”

SQL has this capability.

**Rule 8: Physical Data Independence**

“Applications programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.”

In other words: “You should be able to move the database from one disk volume or directory to another without affecting application programs or interactive users.”

**Rule 9 Logical Data Independence**

“Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.”

In other words: “If you change the schema (adding or removing a table or column, for example) then other parts of the schema that should not be affected by the change remain unaffected.”

**Rule 10: Integrity Independence**

“Integrity constraints specific to a particular relational data base must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.”

“A minimum of the following two integrity constraints must be supported:

1. Entity integrity: No component of a primary key is allowed to have a null value.
2. Referential integrity: For each distinct nonnull foreign key value in a relational data base, there must exist a matching primary key value from the same domain.”

**Rule 11: Distribution Independence**

“A relational DBMS has distribution independence.”

In other words: “A distributed database must look to the user like a centralized database. Application programs and interactive users should not be required to know where data are stored, including the location of multiple copies of the same data.”

**Rule 12: Nonsubversion Rule**

“If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass

the integrity rules or constraints expressed in the higher level relational language (multiple-records-at-a-time).”

In other words: “There must be no way to use a direct-access language to get around the integrity constraints stored in the data dictionary. The integrity rules must be observed without exceptions.”

## **Chapter 7 Using SQL to Implement a Relational Design**

### **1. a) Illustrate the Database Object Hierarchy:**

#### **b) What are the Object Naming Conventions?**

- Column names must be unique within the table
- Table names must be unique within the schema
- Schema names must be unique within their catalog
- Catalog names must be unique within their cluster

For duplicate names, use this format:

catalog\_name.schema\_name.table\_name.column\_name

Names assigned to database elements can include the following:

- Letters
- Numbers
- Underscores

**Complete the following questions with SQL code where appropriate:**

### **2. Schema**

#### **a) To create a new schema:**

```
CREATE SCHEMA schema_name AUTHORIZATION  
owner_user_ID  
{  
other create statements go here  
}
```

#### **b) To specify the current schema for new database elements after a schema has been created initially:**

```
SET SCHEMA schema_name
```

#### **c) To create and add a table to the schema:**

```
CREATE TABLE schema_name.table_name
```

### 3. Domains

#### To create a domain use:

a CREATE SCHEMA statement (or SET SCHEMA if that is supported.)

```
CREATE DOMAIN domain_name data_type  
CHECK (expression_to_validate_values)
```

### 4. Tables

#### a) Three categories of tables:

- Permanent base tables: tables whose contents are stored in the database and remain permanently in the database unless they are explicitly deleted
- Global temporary tables: tables used for working storage that are destroyed at the end of SQL session. The definitions of the tables are stored in the data dictionary, but their data are not. Must be loaded with data each time they are going to be used. Can only be used by current user, but are visible to an entire SQL session (an application program or a user working with an interactive query facility)
- Local temporary tables: similar to global temporary tables, but are visible only to the specific program module in which they are created.

#### b) to create a table:

```
CREATE TABLE table_name  
(column1_name column1_data_type  
  column_constraints,  
column2_name column2_data_type  
column_constraints,  
...table_constraints)
```

#### c) Column Data Types:

- INTEGER (INT): a positive or negative whole number. Number of bits is implementation dependent.

- SMALLINT: A positive or negative whole number, usually half the size of a standard integer.
- NUMERIC: a fixed-point positive or negative number. Has a whole number portion and a fractional portion. You must specify the total length of the number (including the decimal point) and how many of those digits will be to the right of the decimal point (its precision).
- DECIMAL: A fixed-point positive or negative number. Similar to numeric value, but the DBMS may store more digits to the right of the decimal than you specify, which could give extra precision.
- REAL: A “single-precision” floating point value in the format: +-x.xxxxx \* 10YY where YY is the power to which 10 is raised.
- DOUBLE PRECISION (DOUBLE): A “double-precision” floating point number. Range and precision will be greater than with single-precision real numbers.
- FLOAT: A floating point number for which you can specify the precision.
- BIT: Storage for a fixed number of individual bits. Indicate the number of bits: BIT(n) where n is the number of bits.
- BIT VARYING: Storage for a varying number of bits. Up to a specified maximum: BIT VARYING (n)
- DATE: a date
- TIME: a time
- TIMESTAMP: the combination of a data and a time
- CHARACTER (CHAR): A fixed-length space to hold a string of characters. Indicate the width of the column CHAR (n) where n is the amount of space that will be allocated for the column in every row.
- CHARACTER VARYING (VARCHAR): a variable length space to hold a string of characters. Indicate the maximum width of the column VARCHAR (n).
- INTERVAL: a date or time interval

**d) Adding a default value:**

Add DEFAULT keyword to the column definition, followed by the default value

**e) NOT NULL constraints**

Primary key columns must be unique and not null. For columns where a value is required add NOT NULL after the column declaration.

#### **f) Adding Primary keys**

Add a PRIMARY KEY clause to a CREATE TABLE statement:

```
CREATE TABLE table_name  
(...  
PRIMARY KEY (column_name))
```

#### **g) Foreign keys**

**To specify a foreign key:**

```
FOREIGN KEY foreign_key_name (foreign_key_columns)  
REFERENCES primary_key_table (primary_key_columns)  
ON UPDATE update_action  
ON DELETE delete_action
```

#### **Foreign key Action options:**

SET NULL: Replace the foreign key value with null. This isn't possible when the foreign key is part of the primary key of its table.

SET DEFAULT: Replace the foreign key value with the column's default value

CASCADE: Delete or update all foreign key rows.

NO ACTION: On update, make no modification of foreign key values.

RESTRICT: Do not allow deletion of the primary key row.

#### **h) Additional column constraints**

UNIQUE keyword – verifies that all non null values are unique

CHECK clause – use after column declaration using keyword

VALUE in a predicate to indicate the value being checked.

### **5. Views**

#### **a) Typical views**

- One view for every base table that is exactly the same as the base table, but with a different name
- One view for each primary key-foreign key relationship over which you join frequently

- One view for each complex query that you issue frequently
- Views as need to restrict user access to specific columns and rows

## **b) Updatability Issues**

### **Restrictions:**

- A view must be created from no more than one base table or view.
- If the source of the view is another view, then the source view must also adhere to the rules for updatability.
- A view must be created from only one query. Two or more queries cannot be assembled into a single view table using operations such as union.
- The view must include the primary key columns of the base table.
- The view must include all columns specified as not null
- The view must not include any groups of data. It must include the original rows of data from the base table, rather than rows based on values common to groups of data.
- The view must not remove duplicate rows.

### **c) To create a view:**

```
CREATE VIEW view_name AS
SELECT...
```

## **6. Temporary Tables**

### **a) Differences between temporary tables and views:**

- A view exists only for a single query. Each time you use the name of a view, its table is recreated from existing data.
- A temporary table exists for the entire database session in which it was created.
- A view is automatically populated with the data retrieved by the query that defines it.
- You must add data to a temporary table with SQL INSERT commands.
- Only views that meet the criteria for view updatability can be used for data modification.
- Because temp. tables are base tables, all of them can be updated.

- Because the contents of a view are generated each time the view's name is used, a view's data are always current.
- The data in a temp. table reflect the state of the database at the time the table was loaded with data. The contents of the temp. table may be out of sync with other parts of the database.

**b) To create a temporary table:**

```
CREATE GLOBAL TEMPORARY TABLE table_name
(...)
```

or

```
CREATE LOCAL TEMPORARY TABLE table_name
(...)
```

**c) To load with data:**

```
INSERT INTO table_name
  SELECT...
  FROM...
  GROUP BY...
```

**d) Disposition:**

By default, the rows in a temporary table are purged whenever a transaction is committed. You can instruct the DBMS to retain the rows by including `ON COMMIT PRESERVE ROWS` to the end of the table creation statement

## 7. Indexes

**To create an index:**

```
CREATE INDEX index_name
ON table_name (index_key_columns)
```

or

```
CREATE UNIQUE INDEX index_name
ON table_name (index_key_columns)
```

## 8. Modifying Database Elements

**a) Adding new columns**

```
ALTER TABLE table_name
```

ADD column\_name column\_data\_type column\_constraints

**b) Adding table constraints:**

ALTER TABLE table\_name  
ADD table\_constraint

**c) Modifying columns:**

**To replace complete column definition:**

ALTER TABLE table\_name  
MODIFY column\_name column data type column constraints

**To add or change a default value only:**

ALTER TABLE table\_name  
MODIFY column\_name DEFAULT default\_value

**To switch between allowing nulls and not allowing nulls  
without changing any other column characteristics add NULL  
or NOT NULL:**

ALTER TABLE table\_name  
MODIFY column\_name NOT NULL

or

ALTER TABLE table\_name  
MODIFY column\_name NULL

**To modify a column constraint without changing any other  
column characteristics:**

ALTER TABLE table\_name  
MODIFY column\_name  
CHECK (...)

**d) Deleting elements:**

**To delete a column:**

ALTER TABLE table\_name  
DELETE column\_name

**To delete a CHECK table constraint:**

ALTER TABLE table\_name  
DELETE CHECK

**To remove the UNIQUE constraint:**

```
ALTER TABLE table_name  
DELETE UNIQUE (column_name)
```

**To remove a table's PRIMARY KEY:**

```
ALTER TABLE table_name  
DELETE PRIMARY KEY
```

**To delete a foreign key:**

```
ALTER TABLE table_name  
DELETE FOREIGN KEY foreignkey_name
```

**e) Renaming elements:**

**To rename a table:**

```
ALTER TABLE table_name  
RENAME newtable_name
```

**To rename a column:**

```
ALTER TABLE table_name  
RENAME column_name to newcolumn_name
```

## **9. Deleting Database Elements**

**To delete a structural element from a database:**

```
DROP TABLE table_name  
DROP VIEW view_name  
DROP INDEX index_name  
DROP DOMAIN domain_name
```

## **10. Granting and Revoking Access Rights**

**a) Types of access rights:**

SELECT: allows a user to retrieve data from a table or view

INSERT: Allows a user to insert new rows into a table or updatable view. Permission may be granted to specific columns.

UPDATE: Allows a user to modify rows in a table or updatable view. Permission may be granted to specific columns

DELETE: Allows a user to delete rows from a table or updatable view.

REFERENCES: Allows a user to reference a table as a foreign key in a table he or she creates. Permission may be granted to specific columns.

ALL PRIVILEGES: Gives a user all of the preceding rights to a table or view.

**b) Storing access rights:**

Stored in the data dictionary as Systableperm and Syscolperm.

Systableperm (table\_id, grantee, grantor, selectauth, insertauth, deleteauth, updateauth, updatecols, referenceauth)

**c) Granting rights**

GRANT type\_of\_rights

ON table\_or\_view\_name TO user\_ID

**d) Revoking rights**

REVOKE access\_rights

FROM table\_or\_view\_name FROM user\_ID

## **Chapter 8 Using CASE Tools for Database Design**

1. What documents are supported by most CASE (computer-aided software engineering) tools?
  - Data dictionary: the backbone of the project, providing a single repository for all processes, entities, attributes, and domains used anywhere throughout the project
  - Requirements: usually text specifications of what individual parts of the system must do
  - Data flow diagrams: document the way in which data flow throughout an organization, indicating who handles the data. Useful in documenting the relationships between multiple organization units and the data they handle.
  - Structure charts: used to model the structure of application programs that will be developed using structured programming techniques.
  - Data models: the ER diagrams
  - Screen prototypes: drawings of sample screen layouts, useful for documenting the user interface of application

programs. Can act as a cross-check to ensure that a database design is complete by allowing you to verify that everything needed to generate the sample screen designs is present in the database.

- State models: indicate the ways in which data change
- Task diagrams: used to help plan application programs in which multiple operations must occur at the same time.
- Class diagrams: used when performing object-oriented rather than structured analysis
- Object diagrams: used during object-oriented analysis to indicate how objects communicate with one another by passing messages

2. How are CASE tools used for ER Diagrams?

CASE tools simplify creation of ER diagrams and can generate reports that document the contents of an ERD. They can translate relationships into reports such as a relation specification report. They are also useful for pinpointing errors in the graphical version.

3. How are Data Flow Diagrams useful in the database design process?

They help a designer to determine whether an organization needs a single, integrated database or a collection of independent databases.

4. How is the data dictionary used by a CASE tool?

It provides central repository for documenting entities, attributes and domains. By linking with the ER diagram you can provide enough information for the CASE tool to generate the SQL CREATE statements needed to define the structure of the database. Other benefits: ensures that a given database element is defined consistently. If linked to an ER diagram, then the CASE tool can automatically identify foreign keys. Overall, it enforces consistency.

5. What is necessary for the CASE tool to generate usable SQL code?

The CASE software can generate the SQL CREATE TABLE commands for you if it contains a complete data dictionary. The data dictionary must contain:

- Domains for every attribute

- Primary key definitions (created as attributes are added to entities in an ER diagram)
- Foreign key definitions (created by CASE tool after ER diagram is complete)
- Any additional constraints that are to be placed on individual attributes (created when adding attributes to entities in the ER diagram)

6. Why do CASE tools include sample input and output designs? CASE tools provide a way to draw and label sample screen and report layouts so that db designer can double-check that the database can provide all the data needed by application programs.