Abubakr Mamajonov 52493

1. Describe Requirements Gathering and Design Steps.

Requirements Gathering:

Identify stakeholders: Determine the individuals or groups who will be using or affected by the database system.

Conduct interviews and meetings: Gather information by conducting interviews and meetings with stakeholders to understand their needs, expectations, and challenges.

Analyze existing systems and data: Evaluate any existing systems, data sources, or processes to identify potential requirements and improvements.

Use case analysis: Identify and document the various use cases or scenarios that the database system should support.

Define functional and non-functional requirements: Identify the specific functionalities the system should provide and any non-functional requirements such as performance, scalability, security, and usability.

Conceptual Design:

Entity-Relationship (ER) modeling: Create an ER model to represent the conceptual view of the database system, identifying entities, attributes, and relationships between them.

Normalize the data: Apply normalization techniques to eliminate redundancy and ensure data integrity.

Define primary and foreign keys: Identify the primary keys for each entity and establish the relationships using foreign keys.

Logical Design:

Convert ER model to a relational schema: Transform the ER model into a relational schema consisting of tables, columns, and relationships.

Define table structures: Determine the attributes and data types for each table and establish appropriate constraints, such as unique and not null constraints.

Refine relationships: Establish referential integrity by defining foreign keys to enforce relationships between tables.

Normalize the schema: Further normalize the schema to eliminate anomalies and optimize data storage and retrieval.

Physical Design:

Determine storage requirements: Estimate the amount of storage space needed for the database and plan for scalability.

Indexing strategy: Identify the key columns that require indexing to optimize query performance.

Partitioning and clustering: Determine if data partitioning or clustering techniques should be employed to improve performance.

Security and access control: Define appropriate security measures, access controls, and user roles to protect the data.

Backup and recovery strategy: Develop a backup and recovery plan to ensure data integrity and availability in case of failures or disasters.

Implementation and Testing:

Create the database schema: Implement the designed schema by creating tables, relationships, constraints, and indexes in the database management system (DBMS).

Load data: Populate the database with sample or real-world data.

Test the database: Perform various tests to verify the functionality, performance, and security of the database system.

Iterate and refine: Gather feedback from stakeholders and make any necessary adjustments to the design and implementation.

Documentation and Maintenance:

Document the database design: Prepare comprehensive documentation describing the database schema, relationships, constraints, and other relevant details.

Provide user documentation: Create user manuals or guides to help users interact with the database system effectively.

Maintenance and monitoring: Establish processes for monitoring and maintaining the database system, including performance tuning, backups, and regular updates as needed.

2. Describe Crow's Foot Notation – Relationship Symbols

Crow's Foot notation is a popular graphical representation used to depict relationships between entities in entity-relationship (ER) diagrams. In Crow's Foot notation, relationship symbols are used to visually represent the associations between entities. Here are the commonly used relationship symbols in Crow's Foot notation:

One-to-One Relationship:

In Crow's Foot notation, a one-to-one relationship is represented by drawing a straight line between the related entities. Each end of the line is marked with a short vertical line ("crow's foot") and a "1" to indicate that only one instance of each entity can be associated with the other.

One-to-Many Relationship:

A one-to-many relationship is depicted by drawing a straight line from the "one" entity to the "many" entity. The "one" end is marked with a crow's foot symbol and a "1," while the "many" end is marked with a short horizontal line and an "M" to indicate that multiple instances of the "many" entity can be associated with a single instance of the "one" entity.

Many-to-Many Relationship:

A many-to-many relationship is represented by drawing a connector line between the related entities. Each end of the line is marked with a crow's foot symbol and an "M" to indicate that multiple instances of each entity can be associated with each other.

Identifying Relationship:

An identifying relationship is used when the primary key of the child entity includes the primary key of the parent entity as a foreign key. It is denoted by drawing a solid line between the entities and placing a crow's foot symbol on the side of the child entity. This indicates that the child entity's primary key is composed, at least in part, of the primary key of the parent entity.

Non-identifying Relationship:

A non-identifying relationship is used when the primary key of the child entity does not include the primary key of the parent entity. It is represented by drawing a dashed line between the entities and placing a crow's foot symbol on the side of the child entity.

These relationship symbols in Crow's Foot notation provide a clear visual representation of the cardinality and connectivity between entities in an ER diagram, making it easier to understand and communicate the structure of a database system.

3. Describe Systems Development Life Cycle (SDLC).

The Systems Development Life Cycle (SDLC) is a structured approach used to plan, develop, implement, and maintain information systems. It consists of a series of phases or stages that guide the development process from inception to

deployment and beyond. The typical SDLC model encompasses the following stages:

Requirements Gathering:

In this initial phase, the project team identifies the needs and objectives of the system by gathering requirements from stakeholders, users, and domain experts. The requirements are documented and analyzed to establish the scope of the project.

System Analysis:

During this stage, the gathered requirements are analyzed in detail to understand the existing system, identify potential improvements, and determine the functional and non-functional specifications of the new system. Feasibility studies, cost-benefit analysis, and risk assessment may also be performed.

System Design:

In this phase, the system design is created based on the analysis. The architecture, components, and interfaces of the system are defined. The database structure, user interfaces, algorithms, and security mechanisms are designed. This stage ensures that the system's blueprint is comprehensive and aligned with the requirements.

System Development:

The actual development of the system takes place in this phase. The software code is written, modules are integrated, and testing is performed to ensure that the system functions as intended. Developers follow coding standards, best practices, and quality assurance processes during development.

System Testing:

The developed system undergoes various testing processes, including unit testing, integration testing, system testing, and acceptance testing. These tests verify the system's functionality, performance, usability, and security. Bugs and issues are identified, reported, and resolved to achieve a stable and reliable system.

System Implementation:

Once the system has passed testing and received approval, it is deployed in the production environment. This involves installing and configuring the system, migrating data, training users, and transitioning from the old system to the new one. User support and documentation are provided to facilitate a smooth transition.

System Maintenance:

After deployment, the system enters the maintenance phase, where it is continuously monitored, updated, and enhanced to meet changing user needs, fix bugs, and improve performance. Regular maintenance activities include bug fixing, security updates, performance optimization, and user support.

The SDLC is typically followed in a sequential manner, but it can also be iterative or incremental, depending on the project's requirements and the chosen development methodology (e.g., Waterfall, Agile). It provides a systematic framework for managing the development process, ensuring that information systems are developed efficiently, effectively, and with quality throughout their lifecycle.

4. How normal forms can be transformed from lower normal forms to higher normal forms?

Normalization is a process in database design that involves transforming a database schema from lower normal forms (e.g., 1NF or 2NF) to higher normal

forms (e.g., 3NF or Boyce-Codd Normal Form). The transformation is achieved by applying specific rules and guidelines. Here is a general overview of how normal forms can be transformed:

From 1NF to 2NF:

Identify the functional dependencies: Determine the dependencies between attributes in a relation.

Remove partial dependencies: If any attributes depend on only a part of the candidate key, create separate relations for those attributes.

Establish relationships between new and existing relations: Use foreign keys to establish relationships between the new relations and the original relation.

From 2NF to 3NF:

Identify the transitive dependencies: Find dependencies where non-key attributes depend on other non-key attributes.

Create new relations: Move the attributes involved in transitive dependencies to separate relations, including the determinant and dependent attributes.

Establish relationships between new and existing relations: Use foreign keys to establish relationships between the new relations and the original relation.

From 3NF to Boyce-Codd Normal Form (BCNF):

Identify the candidate keys: Determine the minimal set of candidate keys for each relation.

Identify the functional dependencies: Determine all functional dependencies within the relation.

Remove partial dependencies: If any attributes depend on only a part of a candidate key, create separate relations for those attributes.

Analyze for violations of BCNF: Check if there are any functional dependencies where the determinant is not a candidate key.

Create new relations: Move the attributes involved in the BCNF violations to separate relations, including the determinant and dependent attributes.

Establish relationships between new and existing relations: Use foreign keys to establish relationships between the new relations and the original relation.

It's important to note that achieving higher normal forms (such as 4NF or 5NF) may involve further analysis and decomposition of the relations based on additional dependencies and considerations.

Normalization helps eliminate data redundancy, ensures data integrity, and improves database efficiency. However, it's essential to strike a balance between normalization and performance requirements, as highly normalized schemas can sometimes lead to complex queries and reduced performance.

5. What is difference between Strong and Weak Entity?

In database design, there are two types of entities: strong entities and weak entities. The main difference between them lies in their ability to exist independently or dependently within a database schema. Here's an explanation of each:

Strong Entity:

A strong entity is an entity that can exist independently and has its own unique identifier (primary key). It has sufficient attributes to be uniquely identified in the database schema. Strong entities have their own meaning and can participate in relationships with other entities. Examples of strong entities could be "Customer," "Employee," or "Product." They have their own existence and do not rely on other entities to be uniquely identified.

Weak Entity:

A weak entity, on the other hand, is an entity that cannot exist independently and relies on a relationship with a strong entity for its existence. It does not have its own unique identifier but rather uses a partial key or a combination of attributes as its identifier. The primary key of a weak entity consists of the primary key of the related strong entity, along with its own attributes. Weak entities represent concepts or entities that have a weak or dependent relationship with other entities. A typical example of a weak entity is "Order Item," which depends on the "Order" entity for its existence.

Key Differences:

Independence: Strong entities can exist independently and have their own unique identifier, while weak entities rely on a relationship with a strong entity for their existence and use a partial key.

Uniqueness: Strong entities have a unique identifier (primary key) that distinguishes them from other entities, whereas weak entities derive their uniqueness from the combination of the related strong entity's primary key and their own attributes.

Relationships: Strong entities can participate in relationships with other entities, whereas weak entities have a dependent relationship with a strong entity and cannot exist without it.

Attributes: Strong entities have sufficient attributes to be uniquely identified, while weak entities may have additional attributes but rely on the related strong entity for their identification.

It's important to properly identify and model the relationships between strong and weak entities in a database schema to maintain data integrity and accurately represent the real-world entities and their dependencies.

6. For data: Surname, first name 1, first name 2, passport number, street, street number, zip code, city, country, email, telephone, design the database (conceptual and physical model). Tables should be in third normal form or higher.

To design a database schema for the given data while ensuring it meets at least third normal form (3NF), we'll break down the information into logical entities and establish relationships between them. Here's an example of a conceptual and physical model:

Conceptual Model:

Person:

person_id (Primary Key)

surname

first_name_1

first_name_2

passport_number

Address:

address_id (Primary Key)

street

street_number

zip_code

city

country

Contact:

contact_id (Primary Key)

person_id (Foreign Key referencing Person)

email

telephone

Physical Model:

Table: Person

Columns:

person_id (Primary Key)

surname

first_name_1

first_name_2

passport_number

Table: Address

Columns:

address_id (Primary Key)

street

street_number

zip_code

city

country

Table: Contact

Columns:

contact_id (Primary Key)

person_id (Foreign Key referencing Person)

email

telephone

In the above model, we have three tables: Person, Address, and Contact. Each table represents a logical entity with its own set of attributes. Here's a breakdown of the tables and their relationships:

The Person table stores information about individuals, including their unique person_id, surname, first names (first_name_1 and first_name_2), and passport number.

The Address table represents the address details, including address_id (primary key), street, street_number, zip_code, city, and country. The address_id serves as a unique identifier for each address entry.

The Contact table holds contact information, including contact_id (primary key), person_id (foreign key referencing Person table), email, and telephone. The

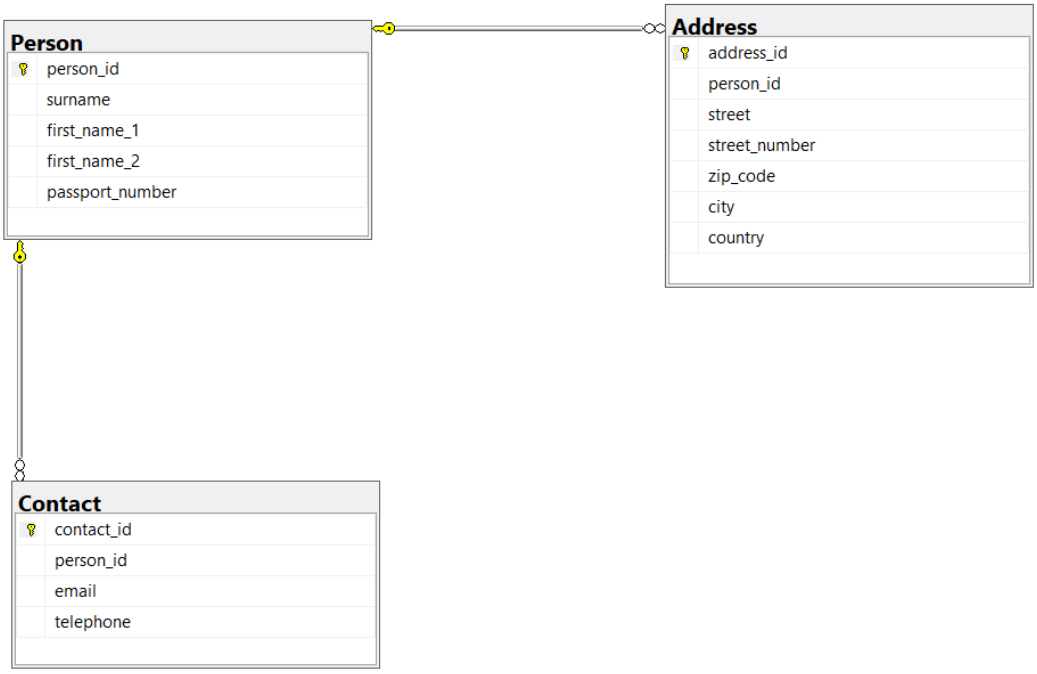
person_id establishes a relationship between the Contact and Person tables, allowing multiple contacts to be associated with a single person.

This design adheres to the third normal form (3NF) as there is no transitive dependency or data redundancy. Each table represents a distinct entity, and the relationships between tables are properly defined using primary key and foreign key constraints.

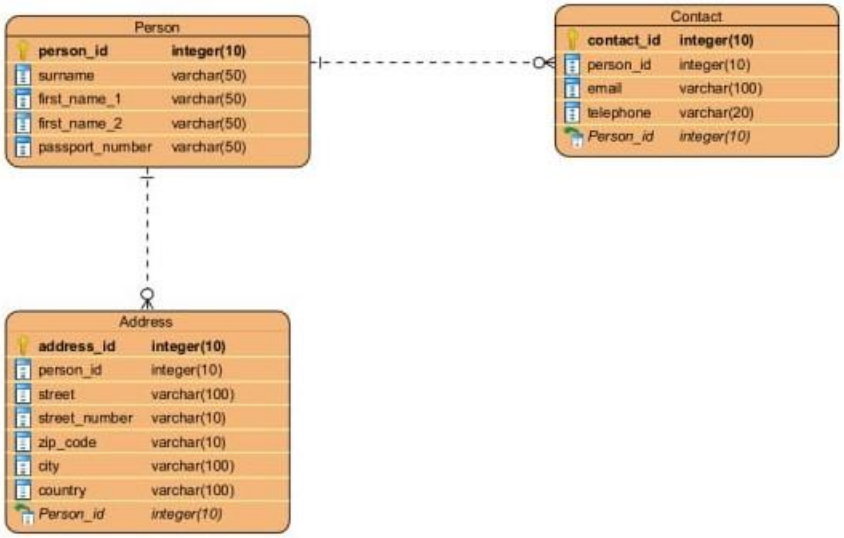
```
-- Create the Person table
CREATE TABLE Person (
  person_id INT PRIMARY KEY,
  surname VARCHAR(50),
  first_name_1 VARCHAR(50),
  first_name_2 VARCHAR(50),
  passport_number VARCHAR(50)
);

-- Create the Address table
CREATE TABLE Address (
  address_id INT PRIMARY KEY,
  street VARCHAR(100),
  street_number VARCHAR(10),
  zip_code VARCHAR(10),
  city VARCHAR(100),
  country VARCHAR(100)
);
```

```
-- Create the Contact table
CREATE TABLE Contact (
  contact_id INT PRIMARY KEY,
  person_id INT,
  email VARCHAR(100),
  telephone VARCHAR(20),
  FOREIGN KEY (person_id) REFERENCES Person(person_id)
);
```

ERD DIAGRAM using SQL Server Management Studio:



ERD DIAGRAM using Visual Paradigm:


Person
person_id integer(10)
surname varchar(50)
first_name_1 varchar(50)
first_name_2 varchar(50)
passport_number varchar(50)
Contact
contact_id integer(10)
person_id integer(10)
email varchar(100)
telephone varchar(20)
Person_id integer(10)
Address
address_id integer(10)
person_id integer(10)
street varchar(100)
street_number varchar(10)
zip_code varchar(10)
city varchar(100)
country varchar(100)
Person_id integer(10)