i) Entities and Their Attributes in the Medical Appointments System

From the case study, the main entities and their attributes are:

1. **Patient**
   - Patient_ID (Primary Key)
   - Name
   - Date_of_Birth
   - Address
   - Contact_Number
   - Registered_Doctor_ID (Foreign Key referencing Doctor)
2. **Doctor**
   - Doctor_ID (Primary Key)
   - Name
   - Specialization
   - Contact_Number
3. **Receptionist**
   - Receptionist_ID (Primary Key)
   - Name
   - Contact_Number
4. **Appointment**
   - Appointment_ID (Primary Key)
   - Patient_ID (Foreign Key referencing Patient)
   - Doctor_ID (Foreign Key referencing Doctor)
   - Appointment_Date
   - Appointment_Time
   - Status (Booked, Cancelled, Completed)
5. **Prescription**
   - Prescription_ID (Primary Key)
   - Appointment_ID (Foreign Key referencing Appointment)

- Patient_ID (Foreign Key referencing Patient)
- Doctor_ID (Foreign Key referencing Doctor)
- Medication_Details

6. **Check-In**
    - CheckIn_ID (Primary Key)
    - Patient_ID (Foreign Key referencing Patient)
    - Appointment_ID (Foreign Key referencing Appointment)
    - CheckIn_Time

ii) Data Flow Diagrams (DFD) for the Medical Appointments System

DFD Level 0 (Context Diagram)

This level represents the system as a whole and how external entities interact with it.

External Entities:

- **Patient**
- **Doctor**
- **Receptionist**

Processes:

- **Manage Patient Records** (Register new patients, update details)
- **Manage Appointments** (Book, cancel, and check-in)
- **Record Consultation Outcomes** (Record appointment outcomes and prescriptions)

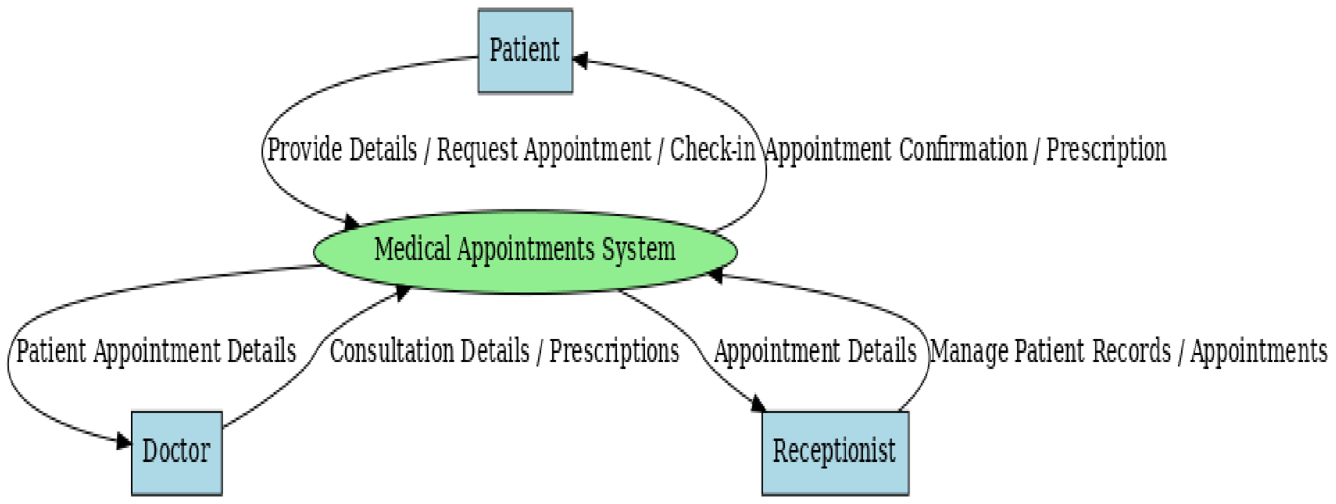
Data Stores:

- **Patient Database**
- **Appointment Database**
- **Doctor Database**
- **Prescription Database**

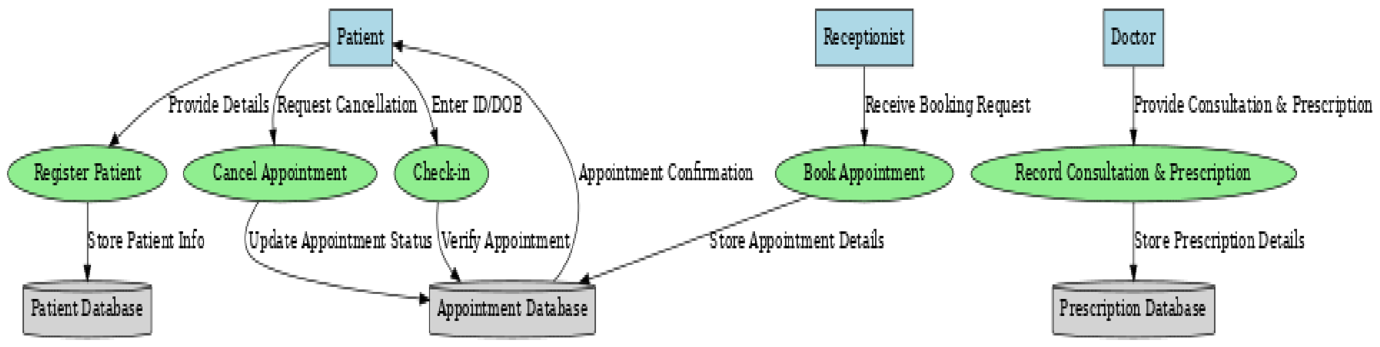
DFD Level 1 (Detailed View of Major Processes)

1. **Process: Patient Registration**
   - A new patient provides personal details to the receptionist.
   - The system generates a unique **Patient_ID** and stores the details.
2. **Process: Appointment Booking**
   - The patient contacts the receptionist with their **Patient_ID** or **Date_of_Birth**.
   - The receptionist checks the available doctors and time slots.
   - The appointment details are stored in the **Appointment Database**.
3. **Process: Appointment Cancellation**
   - A patient requests cancellation via a receptionist.
   - The system updates the **Appointment Database**.
4. **Process: Patient Check-in**
   - The patient enters their **Patient_ID** or **Date_of_Birth** at the terminal.
   - The system verifies the details and marks the check-in.
5. **Process: Doctor Consultation & Prescription Recording**
   - The doctor accesses the patient's appointment record.
   - The doctor records consultation details and prescriptions.
   - The system updates the **Prescription Database**.

Level 0 DFD


Patient
Provide Details / Request Appointment / Check-in
Appointment Confirmation / Prescription
Medical Appointments System
Patient Appointment Details
Consultation Details / Prescriptions
Appointment Details
Manage Patient Records / Appointments
Doctor
Receptionist

Level 1 DFD



2)

(i) Roles of Participants in the Computer-Based Information System Development Process.

1. Management (Project Managers & Company Executives)

- Define project goals and objectives.
- Assign teams to specific tasks based on expertise.
- Ensure that the right talent is matched with task difficulty.
- Oversee project execution and ensure quality standards are met.

2. Quality Assurance (QA) Teams

- Assess the performance of teams based on assigned tasks.
- Ensure that tasks meet company-defined quality standards.
- Provide feedback to improve efficiency and productivity.

3. System Analysts

- Identify the requirements of the system that will manage team assignments.
- Analyze how tasks are currently assigned and recommend improvements.
- Design workflows for the software solution.

4. Programmers (Software Developers)

- Develop the system based on specifications from system analysts.
- Implement features such as **team assignment automation**, **expertise matching**, and **task tracking**.
- Ensure system performance, security, and scalability.

5. Database Designers & Administrators

- Design and manage databases that store information on **employees, tasks, expertise levels, and project phases**.
- Ensure data integrity, security, and accessibility.
- Optimize data retrieval for efficient decision-making.

6. End Users (Teams Performing Tasks)

- Use the system to receive task assignments and track progress.
- Provide feedback on system usability and task allocation.

(ii) Suitable Software Development Process Model

A **Spiral Model** or **Iterative Model** is the best choice for this scenario because:

1. **Continuous Refinement & Improvement** – Since the company wants to match expertise with responsibility and ensure quality work, the system must be refined based on feedback from **QA teams** and **management**.

2. **Risk Management** – The **Spiral Model** incorporates risk analysis at every phase, ensuring the company minimizes failures in task assignment and expertise matching.

3. **Flexibility in Changes** – The company may want to adjust **task categorization, expertise ratings, and performance criteria**, which requires an iterative approach.

4. **User Involvement** – Since **QA teams, management, and workers** will all interact with the system, continuous feedback and refinement are necessary, making an **Iterative Model** a strong fit.

5. **Task Complexity Handling** – The difficulty rating and expertise level logic need testing and refinement over time, which works best with an **incremental** or **spiral** approach.

Thus, the **Spiral Model** is recommended for its ability to handle complexity, flexibility, and continuous feedback.

3)

Entities and Their Attributes for the Exotic Treat Company System

The system described involves **inventory management, sales tracking, and supplier orders**. Below are the key entities and their attributes:

1. Proprietor

- **Proprietor_ID** (Primary Key)
- **Name**
- **Contact Information**
- **Role**

2. Product (Cakes & Sweets)

- **Product_ID** (Primary Key)
- **Product_Name**
- **Category** (Homemade / Supplier)
- **Price**
- **Stock_Quantity**
- **Use_By_Date**
- **Supplier_ID** (Foreign Key, if applicable)

3. Sales

- **Sale_ID** (Primary Key)
- **Product_ID** (Foreign Key)
- **Quantity_Sold**
- **Sale_Date**
- **Total_Amount**

4. Supplier

- **Supplier_ID** (Primary Key)
- **Supplier_Name**
- **Contact_Details**
- **Payment_Method** (Cash on Delivery)

5. Orders (Stock Replenishment)

- **Order_ID** (Primary Key)
- **Supplier_ID** (Foreign Key)
- **Product_ID** (Foreign Key)
- **Quantity_Ordered**
- **Order_Date**
- **Delivery_Date**
- **Payment_Status** (Paid / Pending)

6. Raw Ingredients (For Homemade Products)

- **Ingredient_ID** (Primary Key)
- **Ingredient_Name**
- **Quantity_Available**
- **Use_By_Date**

To create a **Business Process Model and Notation (BPMN) diagram** for the **Supply Management Department of a Supermarket**, the process generally includes:

1. **Stock Monitoring** – Checking inventory levels.
2. **Supplier Communication** – Ordering supplies when stock is low.
3. **Order Approval** – Manager reviews and approves the order.
4. **Order Placement** – Sending the purchase order to the supplier.
5. **Delivery & Payment** – Receiving goods and making payments.
6. **Stock Update** – Updating inventory after receiving supplies.

5)

Question 5: Tools and Techniques Used in System Analysis

Correct Answers:
✅ **A. Business Process Modelling Notation (BPMN)** – Used to visually represent business processes.
✅ **C. Unified Modelling Language (UML)** – Used for designing and analyzing system structures and behaviors.

Question 6: Important Parts of Software Requirements Specification (SRS)

Correct Answers:
✅ **C. User Requirements** – Defines what the users need from the system.
✅ **D. System Requirements** – Specifies system functionalities, constraints, and performance criteria.
✅ **E. Software Specifications** – Provides detailed technical requirements for software development.

6) **Phases of the Spiral Model**

The **Spiral Model** is a risk-driven software development process that combines iterative and waterfall models. It consists of four main phases, repeated in each cycle:

1. **Planning Phase**
    - Gather and analyze requirements.
    - Identify objectives, constraints, and risks.
    - Estimate cost, schedule, and resources.

2. **Risk Analysis Phase**
    - Identify potential risks (technical, financial, user-related).
    - Develop strategies to mitigate risks.
    - Prototype solutions to address uncertainties.

3. **Engineering Phase** 🔧
    - Design, develop, and test the system in small iterations.
    - Implement feedback from the prototype phase.
    - Improve system functionality progressively.

4. **Evaluation & Review Phase**
    - Gather user and stakeholder feedback.
    - Verify if requirements are met.
    - Plan for the next iteration or finalize the project.

7) **Characteristics of a Good Requirement Specification Document**

□Consistency ✅

- **Definition:** A requirement specification should not have conflicting requirements. All statements should align without contradictions.
- **Example:**
    - **Inconsistent Requirement:**
        - *"The system should allow users to reset passwords via email."*

        - *"The system should not send emails for security reasons."*
    - **Consistent Requirement:**
        - *"The system should allow users to reset passwords via email with a one-time verification code."*

2️⃣Unambiguous 🔍

- **Definition:** Each requirement should have a clear, single interpretation to avoid confusion.
- **Example:**
    - **Ambiguous Requirement:**
        - *"The system should be fast."* (How fast?)
    - **Unambiguous Requirement:**
        - *"The system should process transactions within 2 seconds under normal load conditions."*

3️⃣Testable 🧪

- **Definition:** A requirement should be measurable and verifiable through testing.
- **Example:**
    - **Non-Testable Requirement:**
        - *"The system should be user-friendly."* (How do we measure "user-friendly"?)
    - **Testable Requirement:**
        - *"The system should allow a first-time user to complete registration within 3 minutes without external help."*

9)

Key Differences:

User Requirements	**Software Requirements**
Focus on **what** the system needs to do	Focus on **how** the system will implement these needs
Described from a **user perspective**	Described from a **technical/engineering perspective**
Often **abstract** and **non-technical**	More **detailed**, **technical**, and **specific**
Example: *"Allow user login with email"*.	Example: *"System validates email and password against the database."*

□User Requirements

- **Definition:** These describe the needs and expectations of the end users or stakeholders for the system.

Definition: These describe the detailed specifications of how the system will meet the user requirements.

10) Functional requirements

Non-Functional requirements

Domain Requirements

11) **a. Define Data Structure**

A **data structure** is a way of organizing, storing, and managing data so that it can be accessed and modified efficiently.

Examples of data structures include:

- Arrays
- Linked Lists
- Stacks
- Queues
- Trees
- Graphs

b. Simple and Structured Data Types

□Simple Data Types

- **Definition:** These are the basic data types that represent a single value and are not composed of other data types. They are the building blocks for more complex data structures.
- **Examples:**
  - **Integer (int):** Represents whole numbers, e.g., 5, -3, 100.
  - **Floating Point (float):** Represents numbers with decimal points, e.g., 3.14, -0.5.
  - **Character (char):** Represents a single character, e.g., 'A', '9', '%'.
  - **Boolean (bool):** Represents true or false values, e.g., True, False.

Example in C++:

cpp

CopyEdit

```
int age = 25;

float weight = 65.5;

char grade = 'A';
```

2️⃣Structured Data Types

- **Definition:** These are data types that allow you to store multiple values or elements together, often of different types. These data types are constructed from simple data types.
- **Examples:**
    - **Array:** A collection of elements of the same type, e.g., int[] arr = {1, 2, 3, 4, 5};.
    - **Structure (struct):** A collection of variables of different types grouped together, e.g., a person with name, age, and address.
    - **Class (in Object-Oriented Programming):** A blueprint for creating objects with attributes and methods.

Example in C++:

cpp

CopyEdit

```
// Array Example:

int numbers[] = {1, 2, 3, 4};


// Structure Example:

struct Person {

   string name;

   int age;

};
```

Person person1;

person1.name = "John";

person1.age = 30;

c. Static and Dynamic Allocation

1□Static Allocation

- **Definition:** In static memory allocation, the size of the memory required is determined at compile-time, and the memory is reserved for the entire program's lifetime. The memory is allocated in fixed blocks, and it cannot be resized at runtime.
- **Characteristics:**
  - Memory is allocated during the program's compilation.
  - Memory size must be known in advance.
  - More efficient in terms of memory management but less flexible.

Example in C++ (Static Allocation):

cpp

```
int arr[5];  // Array with a fixed size of 5
```

2□Dynamic Allocation

- **Definition:** In dynamic memory allocation, memory is allocated during runtime. The size of the memory block can change, depending on the needs of the program, and the memory is managed manually by the programmer (using pointers in languages like C/C++).

- **Characteristics:**
    - Memory is allocated at runtime.
    - The memory size can be adjusted during the program's execution.
    - More flexible but requires careful management to avoid memory leaks or corruption.

Example in C++ (Dynamic Allocation):

cpp

```
int* arr = new int[5];  // Dynamic allocation of an array of size 5
```

12) c) Prototype

13) b) External user interface

14) **b) Test case generation**
c) Backward traceability
e) Forward traceability

15)

i. Steps for Writing a Software Design Document (SDD)

A Software Design Document (SDD) serves as a blueprint for the development team, providing detailed information about the system's design. Here's a brief breakdown of the steps for writing an SDD:

1. **Introduction**
    - Describe the purpose of the document, the scope of the software, and its intended audience.
    - Define key terms and acronyms used in the document.
    - Provide an overview of the system.
2. **System Overview**
    - Provide a high-level description of the system architecture, including the components and their interactions.
    - Describe the major system features and their functionality.
3. **Design Considerations**
    - Identify design goals such as performance, security, scalability, and usability.
    - Discuss constraints (e.g., hardware, software, or regulatory requirements) that influence the design.
    - Mention any assumptions or dependencies.
4. **Architectural Design**
    - Provide a detailed architecture diagram and describe the system architecture.
    - Break down the system into major components and explain their responsibilities.
    - Discuss how the system’s modules communicate with each other.
5. **Data Design**
    - Define the data structures used within the system.
    - Discuss database schema, entities, relationships, and any data storage mechanisms.
6. **Interface Design**
    - Describe the user interface (UI) design, including screen layouts, user interactions, and visual elements.
    - Define how the system interfaces with external systems or services (e.g., APIs, third-party integrations).

7. **Component Design**
    - Break down the system into smaller modules or components and describe the functionality of each component.
    - Include class diagrams, sequence diagrams, and state diagrams to represent the behavior and structure of the components.
8. **Security Design**
    - Identify security requirements and how they will be addressed.
    - Describe mechanisms for authentication, authorization, data encryption, and any other security measures.
9. **Testing Strategy**
    - Provide an overview of the testing approach, including unit tests, integration tests, system tests, and user acceptance testing (UAT).
    - Mention any testing tools or frameworks that will be used.
10. **Glossary and Appendices**
    - Define technical terms used throughout the document.
    - Include any additional information, diagrams, or references that support the design.

ii. Advantages and Benefits of Creating a Work Breakdown Structure (WBS)

A **Work Breakdown Structure (WBS)** is crucial for organizing and managing a software project effectively. Here are the key advantages and benefits of creating a WBS:

1. **Improved Project Organization**
    - WBS breaks down the project into smaller, manageable tasks and work packages, ensuring all aspects of the project are accounted for. This helps organize the project into a clear, structured hierarchy, making it easier to understand and track progress.
2. **Enhanced Project Planning**
    - By breaking the work into smaller components, project managers can better estimate timelines, resources, and costs. This detailed planning enables more accurate scheduling and budgeting.
3. **Clear Assignment of Responsibilities**
    - WBS allows clear assignment of tasks to different team members. Each task or work package can be allocated to a specific person or team, improving accountability and focus.
4. **Risk Management**
    - By dividing the project into smaller tasks, it is easier to identify potential risks at each stage. Project managers can address and mitigate risks early in the process, reducing overall project uncertainty.
5. **Better Communication**
    - The WBS serves as a communication tool among stakeholders, as it provides a shared understanding of the project's scope and progress. It improves transparency and helps stakeholders track the project more effectively.
6. **Tracking and Monitoring Progress**
    - WBS allows for the easy tracking of task completion. Project managers can monitor the status of individual tasks, ensuring the project stays on schedule and within scope.
7. **Efficient Resource Allocation**
    - With a clear understanding of the tasks involved, resources can be allocated more efficiently. This ensures that team members are not overburdened and that resources are available when needed.
8. **Helps in Setting Milestones**

  - WBS enables the establishment of project milestones. These milestones represent the completion of key deliverables and provide clear indicators of progress.

9. **Quality Control**
   - By breaking down the project, WBS helps ensure that all necessary quality checks and controls are implemented at each level of the project. This ensures a high-quality end product.
10. **Facilitates Scope Management**
    - The WBS helps in defining the project scope clearly and in preventing scope creep by making sure all tasks are included and tracked. Any changes to the project scope can be easily identified and managed.

16) **1. Planning Phase**

- **Objective:** This is the initial phase where the team defines the project goals, user stories, and high-level requirements. It sets the stage for the iterations to follow.
- **Activities:**
  - The product owner works with stakeholders to define the product backlog, which consists of prioritized user stories (features or functionalities that the system must have).
  - Teams and stakeholders agree on the project's vision, goals, and how to prioritize features.
  - A high-level project roadmap may be developed, though it remains flexible and open to change.
  - Sprint planning is also a part of this phase, where the team selects tasks (user stories) for the upcoming sprint (typically a 2-4 week period).

2. Design Phase

- **Objective:** In this phase, the team designs the architecture and system components based on the requirements.

- **Activities:**
    - Developers design the system architecture, user interfaces, and any other necessary components.
    - The design should be flexible to adapt to changes and feedback later in the process.
    - In Agile, the design is often done incrementally, allowing for adaptations as development progresses and user feedback is received.
    - Collaboration with stakeholders helps ensure the design meets user expectations.

3. Development Phase

- **Objective:** This is the phase where the actual coding and building of the product occur.
- **Activities:**
    - Developers implement the features defined in the user stories.
    - Code is written in small, manageable pieces or increments.
    - Continuous integration is often used to frequently merge changes and ensure the product is always in a working state.
    - Testing is also conducted throughout this phase, with tests performed on each feature or component as it is developed.
    - Developers work closely with product owners and other stakeholders to clarify any misunderstandings or adjustments needed.

4. Testing Phase

- **Objective:** In Agile, testing happens continuously throughout the development cycle to ensure the product works as expected.
- **Activities:**
    - Testing is done in parallel with development, with developers performing unit tests, integration tests, and sometimes even user acceptance testing.
    - Testers ensure that each feature meets the specified requirements and works in the overall system context.
    - Agile encourages early and frequent testing to identify issues quickly and fix them before they become larger problems.
    - Automation testing tools are often used to streamline this process.

5. Review Phase

- **Objective:** This phase is about getting feedback from stakeholders and adjusting the product accordingly.
- **Activities:**
    - At the end of each sprint or iteration, the team conducts a sprint review meeting where they demonstrate the newly developed features to the stakeholders.
    - Stakeholders provide feedback, and if any changes or additions are needed, they are added to the product backlog for future sprints.
    - The review is collaborative, and the team works together with stakeholders to assess whether the project is on track or if any adjustments are necessary.

6. Release Phase

- **Objective:** Once the software is ready, it is deployed to the users for production.
- **Activities:**
    - After successful testing and feedback gathering, the product is released to the customers or users.
    - The release phase can involve additional tasks, such as preparing documentation, training end users, or setting up deployment processes.
    - In some cases, releases may happen multiple times throughout the project, especially in continuous delivery models.