Abhinav Yalamaddi - YXA210040

Database Design - CS 6360.002 - Fall 2022 - HW1A

1.1. Define the following terms: data, database, DBMS, database system, database catalog, program-data independence, user view, DBA, end user, canned transaction, deductive database system, persistent object, meta-data, and transaction-processing application.

Data: Identified facts with implicit meaning that can be recorded, For instance, a customer's delivery address, or a bank balance.
Database: A collection of connected data that is logically consistent and has some inherent significance. It represents some part of reality and is created, built, and data-populated for a specific purpose, for a certain group of intended users. - ex. A customer database containing personal information about consumers (name, address, phone number) would be used by a delivery app.
DBMS: A database management system (DBMS) is a general-purpose software system that allows users to define, manipulate, query, retrieve, share, and manage data stored in a database.
Database System: A database system is made up of a database and DBMS software.
Database Catalog: This contains information on the definition and description of the database structure and constraints, as well as information on the structure of each file, the type and storage format of each data item, and multiple data constraints.
Program Data Independence: The ability to modify data attributes without having to change programs that must access the data.
User View: A subset of the database or virtual data that is derived from but not explicitly contained in the database files.
DBA: Database Administrators are in charge of keeping a database environment running well, as well as keeping data secure and intact.
End User: The person who eventually uses a product - in the case of a database, the end user may be a web application or a mobile app.
Canned Transaction: Standard queries and updates that have been meticulously programmed and tested. Naive end-users frequently utilize this to constantly query and update the database.
Deductive Database System: A database system that can derive conclusions from the rules and information stored in its database.
Persistent Object: An item that survives program termination and can be recovered when the program is restarted.
Meta-data: Descriptive information regarding the core database's structure.
Transaction Processing application: Applications used concurrently by multiple users attempt to update or add to the database - for example, a web application for purchasing flight tickets.

Abhinav Yalamaddi
YXA210040

1.3. Discuss the main characteristics of the database approach and how it differs from traditional file systems.

Each user defines and implements the files required for a specific software application in traditional file systems. While several users may be interested in the same data set, each group keeps distinct files because each wants data that is not available in the other's files. A single repository stores data that is defined once and then accessed by several users in the database approach. In contrast to a file system, where each application can separately name data components, the names and labels of data in a database system are defined once and used repeatedly by queries, transactions, and applications. Moreover, the database method incorporates a database system's self-describing nature, supports data abstraction as well as different views of data, and allows for data sharing and multi-user transaction processing. A typical file system does not support the above features.

1.6. Discuss the capabilities that should be provided by a DBMS.

Storage, update, and retrieval - permits data to be read, updated, deleted, and added to a database.
Metadata is a catalog that contains all of the information that describes the database, the data, and other attributes.
Controlling redundancy is important in order to reduce duplication of effort, conserve storage space, and increase efficiency.
Transaction support - ensures that every piece of a database update transaction is completed properly, or provides version control to roll back any partial update to guarantee the database remains in a consistent state.
Allowing sophisticated objects in object-oriented languages to survive program termination and be accessed later by providing persistent storage for program objects.
Query processing and optimization involve selecting an optimal query execution plan for each query based on existing storage structures and performing queries and changes efficiently.
Security – system protection against unwanted access or attacks, as well as protection against software and hardware crashes (through backups and recoveries).
Concurrency - gives assistance when numerous users are accessing and modifying the database at the same time to ensure that the data remains consistent.
Multiple user interfaces - offering distinct user interfaces to different user groups based on varying degrees of technical skill that are appropriate for the authority provided to that user group.
Representing complicated relationships with data - in addition to establishing new relationships as they arise, and conveniently retrieving and updating relevant data
Integrity constraints – give the ability to create and enforce data quality limitations.
Allow for rule-based inference and actions - give tools for defining deduction rules for inferring new information from stored database facts.

Abhinav Yalamaddi
YXA210040

1.7. Discuss the differences between database systems and information retrieval systems.

Database Systems: This applies to structured and prepared data encountered in everyday applications. Using keywords, data is cataloged, annotated, and indexed in a database system. In the case of a database system, the database is used to generate information.

Information retrieval system: This is an alternative method for gathering information. In this scenario, the material searching is based on keywords, which solves many issues with free-form text processing of documents. The internet or network is employed to obtain information in this case.

1.8. Identify some informal queries and update operations that you would expect to apply to the database shown in Figure 1.2.

Update Instructor for CS1310 section 119 with “Abhinav”
Add course_name “Introduction to Machine Learning” course_number CS 5822 credit_hours 3 Department CS
Find instructor for a CS3380 for Fall
Find grades for all students in section 135
Find all courses that have MATH2410 as a prerequisite

1.14. Consider Figure 1.2.
a. If the name of the ‘CS’ (Computer Science) Department changes to the ‘CSSE’ (Computer Science and Software Engineering) Department and the corresponding prefix for the course number also changes, identify the columns in the database that would need to be updated.

In the Student table - the Major column
In the Course table - the Course_number and Department columns
In the Section table - the Course_number column
In the Prerequisite table - the Course_number and Prerequisite_number

Abhinav Yalamaddi
YXA210040

2.1. Define the following terms: data model, database schema, database state, internal schema, conceptual schema, external schema, data independence, DDL, DML, SDL, VDL, query language, host language, data sublanguage, database utility, catalog, client/server architecture, three-tier architecture, and n-tier architecture.

Data Model: A data model is a set of concepts used to describe the structure of a database. It provides the necessary tools for data abstraction.
Database schema: A database's description.
Database state: The data in a database at a specific moment in time.
Internal schema: Specifies a database's physical storage structure.
Conceptual schema: Specifies the overall structure of the database for a group of users.
External schema: Defines the portion of a database that a specific user group is interested in while concealing the remainder of the database from that user group.
Data independence: This refers to the ability to change the schema at one level of the database without affecting the schema at the next higher level.
DDL: Data Definition Language - is a programming language used to define conceptual and internal schemas.
SDL: (Storage Definition Language) This is used to design the internal structure.
DML: (Data Manipulation Language) This is used for data retrieval, insertion, deletion, and modification.
VDL: View Definition Language - this is a language used to define user views and their mappings to the conceptual schema.
Query language: A high-level DML that is utilized in a standalone, interactive environment.
Host language: A general-purpose programming language with DML commands embedded in it.
Data Sublanguages: DML instructions incorporated in a general-purpose programming language are referred to as data sublanguages.
Database utilities: Programs that assist in the management of the database system (loading, backup, storage reorganization, performance monitoring).
Catalog: A catalog is a collection of metadata about a database.
Client/server architecture: A computing approach in which the server hosts, provides, and controls the majority of the client's resources and services. This architecture consists of one or more clients that are linked to a server via a network connection and share computational resources.
Three-tier Architecture: Client (first tier): A client that communicates with the user. Application/Web Server (second tier): Contains the application's business logic. Database server (Tier 3): Layer of database services
N-tier architecture - Extensions to the three-tier design in which each layer, such as the client or application layers, is further subdivided into many levels. This has the advantage of allowing each layer to run on its own processor or operating system and be handled independently, in addition to distributing programming and data throughout the network.

Abhinav Yalamaddi
YXA210040

2.3. What is the difference between a database schema and a database state?

A database schema is a database's description or blueprint. It outlines a database's structure and organization.
A database state refers to the data in a database at a specific point in time.
When a database is first defined, it is in an empty state. The database is converted from one state to another whenever it is modified.

2.4. Describe the three-schema architecture. Why do we need mappings between schema levels? How do different schema definition languages support this architecture?

The three-schema architecture allows user applications to be separated from the physical database. It is made up of three parts: an internal schema, a conceptual schema, and an external schema.
To change requests and query results across the external, conceptual, and internal levels, mappings are required. In DBMSs when there is no rigorous separation of conceptual and internal data. To define both schemas, DDL is used. If the separation is strict, DDL is used exclusively for the conceptual schema while SDL is used for the internal schema. For systems that have a distinct
VDL is used to specify user views and their mappings to the conceptual schema in a three-schema design.
A comprehensive language like SQL, which combines DDL and VDL, is utilized in the present DBMS. The internal schema is given by a combination of functions, parameters, and storage-related specifications; there is no particular language that plays the role of SDL.

2.5. What is the difference between logical data independence and physical data independence? Which one is harder to achieve? Why?

The ability to update the conceptual schema without having to change the external schemas or application programs is referred to as logical data independence.
Physical data independence refers to the ability to change the internal schema without affecting the conceptual or external schema.
Most databases and file environments provide physical data independence, where physical facts are hidden from the user and applications are oblivious of these features.
Logical data independence is far more difficult to establish since it permits structural and constraint changes to occur without affecting application programs, which is a much more stringent requirement.

Abhinav Yalamaddi
YXA210040

2.6. What is the difference between procedural and nonprocedural DMLs?

Procedural DML - A low-level DML must be incorporated in a general-purpose programming language since it is used to get individual records or objects from the database and process them individually, which necessitates programming language constraints such as looping.

Non-procedural DML - A high-level DML that can be used on its own to concisely express complex database operations. These DML statements can be typed into a terminal interactively or included in a general-purpose programming language.

2.14. If you were designing a Web-based system to make airline reservations and sell airline tickets, which DBMS architecture would you choose from Section 2.5? Why? Why would the other architectures not be a good choice?

A three/n-tier architecture would be used. The user interface will be on the client tier, the web server will have any reservation-related business logic, and the database layer will have the DBMS that users will read from/update while looking up and selling airline tickets.
The two-tier client/server architecture would be a poor choice since any business logic would have to reside in either the database or the client tier. This would place undue strain on the database server or load the client tier with extra functionality, potentially resulting in a slower application and a worse user experience. Because a web application requires the user interface and database server to be on separate platforms, the centralized DBMS design would not be a good solution.

7.1. Discuss the role of a high-level data model in the database design process.

High-level data models aid in conceptual design by allowing users to articulate their data requirements, and they include thorough descriptions of entity kinds, relationships, and constraints. The high-level data model is also used as a reference to guarantee that all users' data requirements are met and that no conflicts exist.

7.3. Define the following terms: entity, attribute, attribute value, relationship instance, composite attribute, multivalued attribute, derived attribute, complex attribute, key attribute, and value set (domain).

Entity: It is a real-world object that we want to represent in the Database along with its attributes.

Attribute: In the actual world, an Attribute represents an Entity's properties. For example, the Student entity has attributes such as Id, Name, Address, and their respective values. Attributes are recorded in the database as table columns with values.

Attribute value: This value is used to describe an entity. It is connected to the attributes of an entity. For example, a Student has an attribute name, thus the value will be a student's name.

Relationship instance: A database tuple is simply a single row with a single row of data for that relation. A relation instance is a collection of such tuples.

Composite attribute: A composite attribute is a grouping of one or more attributes. A customer's name, for example, is divided into the first and last name extra characteristics, and then a collection of them is termed a composite attribute.

Multivalued attribute: When an attribute contains more than one value, it is referred to as a multivalued attribute. For example, an employee may have many contact numbers, hence the phone property in the database may have multiple values for a single person.

Derived attribute: These attributes include the value that has been derived or taken from other values. For example, we can calculate a value for the age attribute in the table based on the birthdate attribute.

Complex attribute: It is an attribute that covers many values from other multiple attributes. For example, if a person has many contact numbers, emails, and addresses, then a contact detail attribute to represent these values is a complicated attribute.

Key attribute: This attribute is used to uniquely identify any data by combining many attributes or by using a single attribute.

Value set: This is a set of values assigned to each entry's attribute. For example, the Name property can accept string values from a-z, A-Z, blank letters, and so forth.

7.4. What is an entity type? What is an entity set? Explain the differences between an entity, an entity type, and an entity set.

Entity Type: The Entity Data Model's Entity Type is the essential building block for describing the structure of data (EDM). An entity type in a conceptual model represents the structure of top-level concepts such as customers or orders.

An entity key is defined by one or more properties.

Entity Set: A set of entities of the same kind is referred to as an entity set (e.g., all persons having an account at a bank). Entity sets do not have to be disjoint. For example, the entity set employee (all bank employees) and the entity set customer (all bank customers) may share members.

Difference between an entity, entity type, and entity set:-

Abhinav Yalamaddi
YXA210040

An entity is an existing object that can be distinguished from other objects. An entity can be concrete (such as a person or a book) or abstract (like a holiday or a concept). An entity set is a collection of entities of the same type (for example, all people who have a bank account).

7.7. What is a participation role? When is it necessary to use role names in the description of relationship types?

The part of a relationship in which each entity participates is referred to as the participation role. When the same entity type participates in a relationship type many times in distinct roles, role names must be used in the description. In recursive relationships, role names are required.

7.11. What is meant by a recursive relationship type? Give some examples of recursive relationship types.

A recursive relationship exists when an entity can be associated to itself or when the same entity type participates in a relationship type multiple times in different roles. A supervisory relationship between an employee and their supervisor is an example of a recursive relationship. Both entities belong to the Employee entity set. The employee entity type engages in the supervision relationship twice in this scenario, once as a supervisor and once as an employee. Another case in point is a course prerequisite relationship, in which one course is a requirement for another, and both entities are members of the Course entity set. The course entity type appears twice, first as a requirement and again as a course, resulting in a recursive relationship.

7.12. When is the concept of a weak entity used in data modeling? Define the terms owner entity type, weak entity type, identifying relationship type, and partial key.

When there are many attributes or when entity types do not have key attributes of their own, the idea of a weak entity is utilized in data modeling.

Owner Entity Type: Owner entity types are entities that are related to weak entity types by combining some of their attribute values.

Weak Entity Type: A weak entity type is one that lacks significant characteristics of its own.

Identifying Relationship Type: The identifying relationship type is the sort of relationship that connects a weak entity type to its owner.

Partial Key: A partial key is a collection of qualities that can be used to distinguish weak entities that are tied to the same owner entity.

Abhinav Yalamaddi
YXA210040

7.13. Can an identifying relationship of a weak entity type be of a degree greater than two? Give examples to illustrate your answer.

Yes, a weak entity type can have an identifying relationship with a degree greater than two. The weak entity type has numerous owner entity types for a ternary identifying relationship type. A Supply relationship, for example, associates three entities - supplier, part, and project - with each relationship instance. This must be expressed as a weak entity type with no partial key.
The owner entity categories are comprised of the three participating entities.

7.17. Composite and multivalued attributes can be nested to any number of levels. Suppose we want to design an attribute for a STUDENT entity type to keep track of previous college education. Such an attribute will have one entry for each college previously attended, and each such entry will be composed of college name, start and end dates, degree entries (degrees awarded at that college, if any), and transcript entries (courses completed at that college, if any). Each degree entry contains the degree name and the month and year the degree was awarded, and each transcript entry contains a course name, semester, year, and grade. Design an attribute to hold this information. Use the conventions in Figure 7.5.

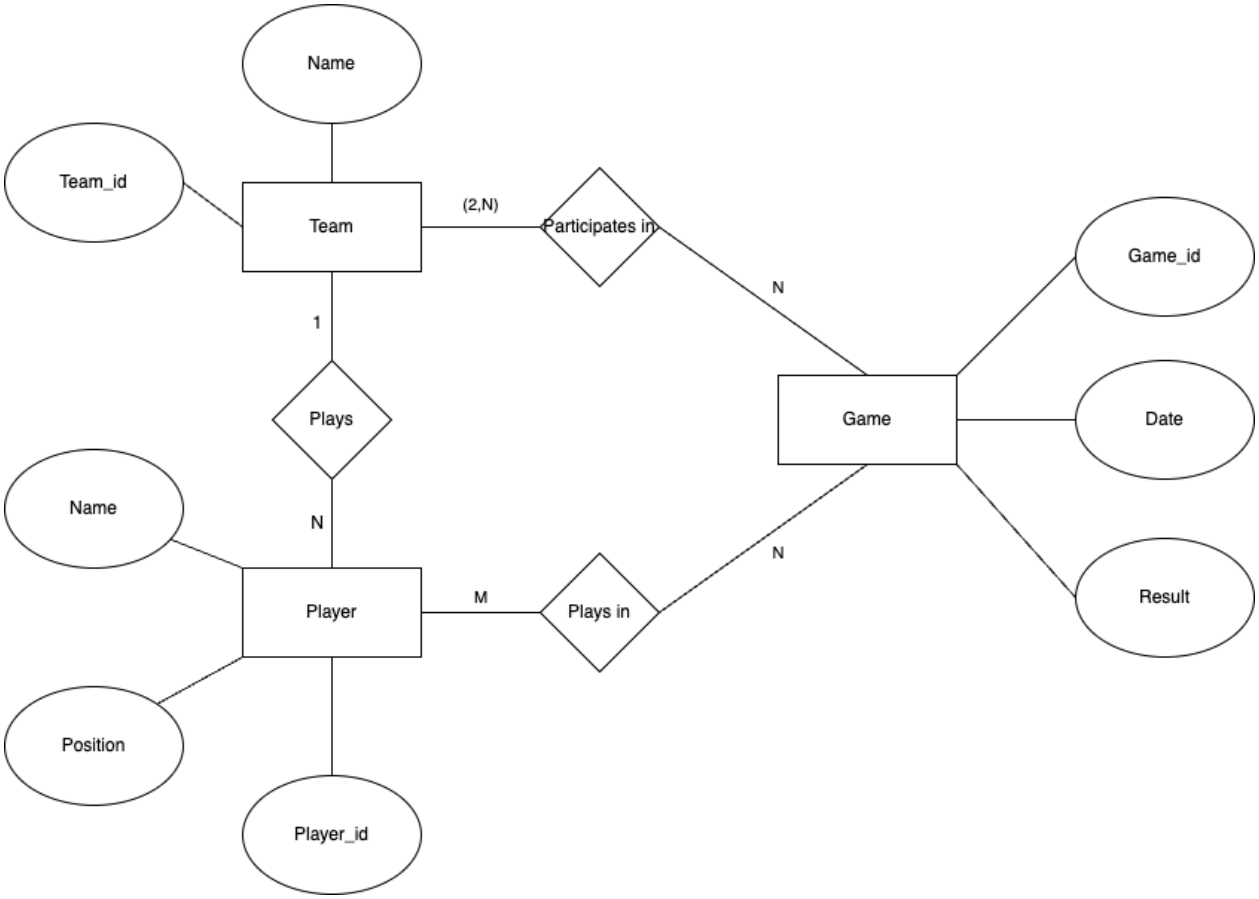
{Previous_education(College_name, Start_date, End_date,
{Degree_awarded (Degree_name, Degree_month, Degree_year)},
{Transcript (Course_name, Semester, Year, Grade)})}

7.19. Consider the ER diagram in Figure 7.20, which shows a simplified schema for an airline reservations system. Extract from the ER diagram the requirements and constraints that produced this schema. Try to be as precise as possible in your requirements and constraints specification.

Each airport is assigned an Airport code, and we must keep track of the airport's name as well as the city and state in which it is located. For landing, several instances of Airports are accessible for multiple instances of Airplane types. Each Airport instance can have numerous leg instances leaving from it. Each Airport instance can have numerous leg instances arrive at it. Multiple instances of Flight Leg can be scheduled to arrive and leave from each Airport instance.
Each Airplane type is identified by a Type Name, and we must keep track of the company and the maximum number of seats. Airplane-type objects can land at numerous Airports. Every instance airplane type is made up of many instances of Airplane.
Each Airplane is identified by an Airplane id, and the total number of seats must be kept track of. Airplane instances must be of a specified Airplane type instance. Each Airplane instance can be allocated to various Leg Instance instances.
Each Flight is assigned a number and we need to keep track of the airline and weekdays.
Each instance of Flight can offer multiple instances of Fare. Flight instances can contain multiple Flight Leg instances.

A Leg number is assigned to each Flight Leg. Flight Leg instances can be made up of Leg instance instances. Flight Leg instances must be scheduled for arrival and departure at a certain Airport instance.
Each fare is assigned a code, and we must keep track of the amount and restrictions.
Fare instances must be linked to a single instance of Flight.
Each Leg Instance is identifiable by date, and we need to keep track of how many seats are available. Each Leg Instance must be associated with at least one Airplane and be a Flight Leg instance. Multiple Seat instances can be reserved for each Leg instance.
Leg Instance instances must depart from a specific Airport instance. Leg Instance instances must arrive at a certain Airport instance.
A seat number is assigned to each seat. Seat instances must be reserved for a certain Leg instance.

7.22. A database is being constructed to keep track of the teams and games of a sports league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Design an ER schema diagram for this application, stating any assumptions you make. Choose your favorite sport (e.g., soccer, baseball, football).



Abhinav Yalamaddi
YXA210040

7.23. Consider the ER diagram shown in Figure 7.21 for part of a BANK database. Each bank can have multiple branches, and each branch can have multiple accounts and loans.

a. List the strong (nonweak) entity types in the ER diagram.
Bank, Loan, Account, Customer

b. Is there a weak entity type? If so, give its name, partial key, and identifying relationship.
Bank_Branch - Branch_no, Branches

c. What constraints do the partial key and the identifying relationship of the weak entity type specify in this diagram?
Branch_no is a partial key that indicates that a Branch_no is not globally unique but is unique inside a specific Bank. By combining Branch_no and Bank_code, a branch can be uniquely recognized.
Bank entities are constrained by the partial key in such a way that each branch of a single bank must have a unique Branch_number.
A cardinality ratio limitation is defined by the identifying connection Branches, which states that each Bank_Branch must belong to a single Bank and that a Bank can have many Bank_Branches. Furthermore, both the Bank and the Bank_Branch entities are subject to a 100% participation constraint, which means that a bank cannot exist in the absence of branches, and bank branches cannot exist in the absence of a bank.

d. List the names of all relationship types, and specify the (min, max) constraint on each participant of an entity type in a relationship type. Justify your choices.
Branches - Bank (1,N) - One Bank can have many associated branches. Bank_Branch (1,1) - Bank_Branch has total participation in the Branches relationship so has a min constraint of 1 and can only belong to a max of 1 Bank.

Account (1,1) - An account must belong to a Branch and can only be associated with one branch. Accts - Bank_Branch (0,N) - A Bank_Branch can hold any number of accounts with no participation constraints.

A_C - Accounts (1,N) - An account must belong to at least one customer, and can belong to more than one customer. Customer (0,N) - A customer can have 0 to any number of accounts.

Customer (0,N) - A customer can have 0 to N loans. L_C - Loans (1,N) - A loan must be associated with at least one customer and can belong to multiple customers.

Loans - Bank_Branch (0,N) - A Bank_Branch can hold any number of loans with no participation constraints. Loans (1,1) - A loan must belong to a Branch and can only be associated with one branch.

Abhinav Yalamaddi
YXA210040

e. List concisely the user requirements that led to this ER schema design.
- Each bank has its own code, name, and address.
- Each account is identified by a unique account number, balance, and type.
- Each loan has its own loan number, loan amount, and loan category.
- Every customer has a distinct Social Security Number, name, address, and phone number.
- A bank has one or more branches, each with its own branch number and address. Each branch is owned by a particular bank.
- A bank branch opens accounts for consumers, and each account must be assigned to a certain branch.
- A bank branch makes loans to consumers, and each loan must be assigned to a certain branch.
- Accounts and loans can belong to more than one customer, and accounts loans can belong to more than one consumer.

f. Suppose that every customer must have at least one account but is restricted to at most two loans at a time and that a bank branch cannot have more than 1,000 loans. How does this show up on the (min, max) constraints?
The customer to A_C relationship would have a constraint of (1,1).
The customer to L_C relationship would have a constraint of (0,2).
The Bank_Branch relationship with Loan would have a constraint of (0,1000)

7.27 Cardinality ratios often dictate the detailed design of the database. The cardinality ratio depends on the real-world meaning of the entity types involved and is defined by the specific application. For the following binary relationships, suggest cardinality ratios based on the common-sense meaning of the entity types. Clearly state any assumptions you make.
1. 1 to Many - A student may have more than one social security card (legally with the same unique social security number), and every social security number belongs to a unique student.
2. Many to Many - Multiple teachers can be assigned to students, and multiple teachers can be assigned to students.
3. Many to Many - A classroom has several walls that can be shared by several classrooms.
4. 1 to 1 - Every country can only have one president at any given time, and each president can only rule over one country.
5. Many to Many - A textbook can be allocated to numerous courses, and a textbook can be assigned to multiple courses.
6. Many to Many - An item can be part of numerous orders, and an item can be part of multiple orders.
7. Many to Many - Multiple students can attend multiple courses, and multiple students can be offered courses.
8. Many to 1 - Assuming that every class has a unique instructor. In case instructors were allowed to team teach, this will be many-many.
9. 1 to 1 - An instructor can only have one office, and each office can only have one instructor (assuming the office is not shared by numerous instructors).
10. 1 to many - An item can be included in many bids, but each bid is unique to a single item.

Abhinav Yalamaddi
YXA210040

7.28 Consider the ER schema for the MOVIES database in Figure 7.24.

Assume that MOVIES is a populated database. ACTOR is used as a generic term and includes actresses. Given the constraints shown in the ER schema, respond to the following statements with True, False, or Maybe. Assign a response of Maybe to statements that, while not explicitly shown to be True, cannot be proven False based on the schema as shown. Justify each answer.

A. There are no actors in this database that have been in no movies.
True - Total participation of Actor in Performs_In relationship.

B. There are some actors who have acted in more than ten movies.
Maybe - there is no upper bound for Actors in the Performs_In association, but it is uncertain whether any actors have been in more than ten films.

C. Some actors have done a lead role in multiple movies.
True - The lead_Role relationship with the Actor entity has an upper bound constraint, although it is unknown if there are certain actors that have played lead parts in several movies.

D. A movie can have only a maximum of two lead actors.
True - Actor - Lead_role relationship type has a constraint of 2.

E. Every director has been an actor in some movie.
False - ALSO_A_DIRECTOR contains a single line that states that some may not.

F. No producer has ever been an actor.
False - Some producers may be actors

G. A producer cannot be an actor in some other movie.
False - There are no restrictions that prevent producers from appearing in other films.

H. There are movies with more than a dozen actors.
False - A film must include at least one actor, although it is uncertain whether there are any films with more than a dozen actors.

I. Some producers have been a director as well.
Maybe - There are no rules against producers from also being directors, although it is unknown whether there are particular examples of producers who have also been directors.

J. Most movies have one director and one producer.
Maybe - A movie can have only one director but numerous producers; it is unknown whether most movies have only one producer.

Abhinav Yalamaddi
YXA210040

K. Some movies have one director but several producers.
True - All movies must have a single director but may have several producers. It is unknown whether some films have only one producer.

L. There are some actors who have done a lead role, directed a movie, and produced some movies.
Maybe - There are no restrictions on performers taking on starring roles, directing, and producing films. It is unknown whether any actors have done this.

M. No movie has a director who also acted in that movie.
Maybe - it is possible for a director to have also acted in a movie, but it is unclear if there is a specific instance of this.