**Q1.1**

**Data**

The word data is derived from the Latin word **datum** which means something given. **Data** are known facts that can be recorded and which have implicit meaning.

**Database**

A database is an organized collection of previously defined data.

**DBMS (DataBase Management System)**

DBMS (database management system) is a computerized system that enables users to create and maintain a database. It helps define, construct, manipulate, and share databases between different users and applications.

**Database catalog**

Database catalog stores a complete definition or description of the database structure (e.g., structure of each file, the type, and storage format) and constraints on the data.

**Program-data independence**

Program-data independence is a property of DBMS access programs that allows a change in the structure of some files independently from access programs by storing the structure of data files in the DBMS catalog separately from the access programs.

**User view**

User view is a perspective of the database specific for a certain user (or group of users), which can be a subset of the database, or it may contain virtual data that is derived from the database files but is not explicitly stored.

**DBA (Database Administrator)**

\*DBA \*or database administrator administers primary (database) and secondary resources (DBMS and related software). DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. DBA also handles security breaches and poor system response time.

**End user**

End user is the people accessing the database for querying, updating, and generating reports. The end user is the primary reason for the creation of a certain database.

**Canned transactions**

\textbf{Canned transactions} are the standard types of queries and updates that have been carefully programmed and tested, used by naive or parametric end users when constantly querying and updating the database.

**Deductive database systems**

Deductive database systems are database systems that provide capabilities for defining deduction rules for inferencing new information from the stored database facts.

**Persistent object**

Persistent object survives the termination of program execution and can later be directly retrieved by another program.

**Transaction-processing application**

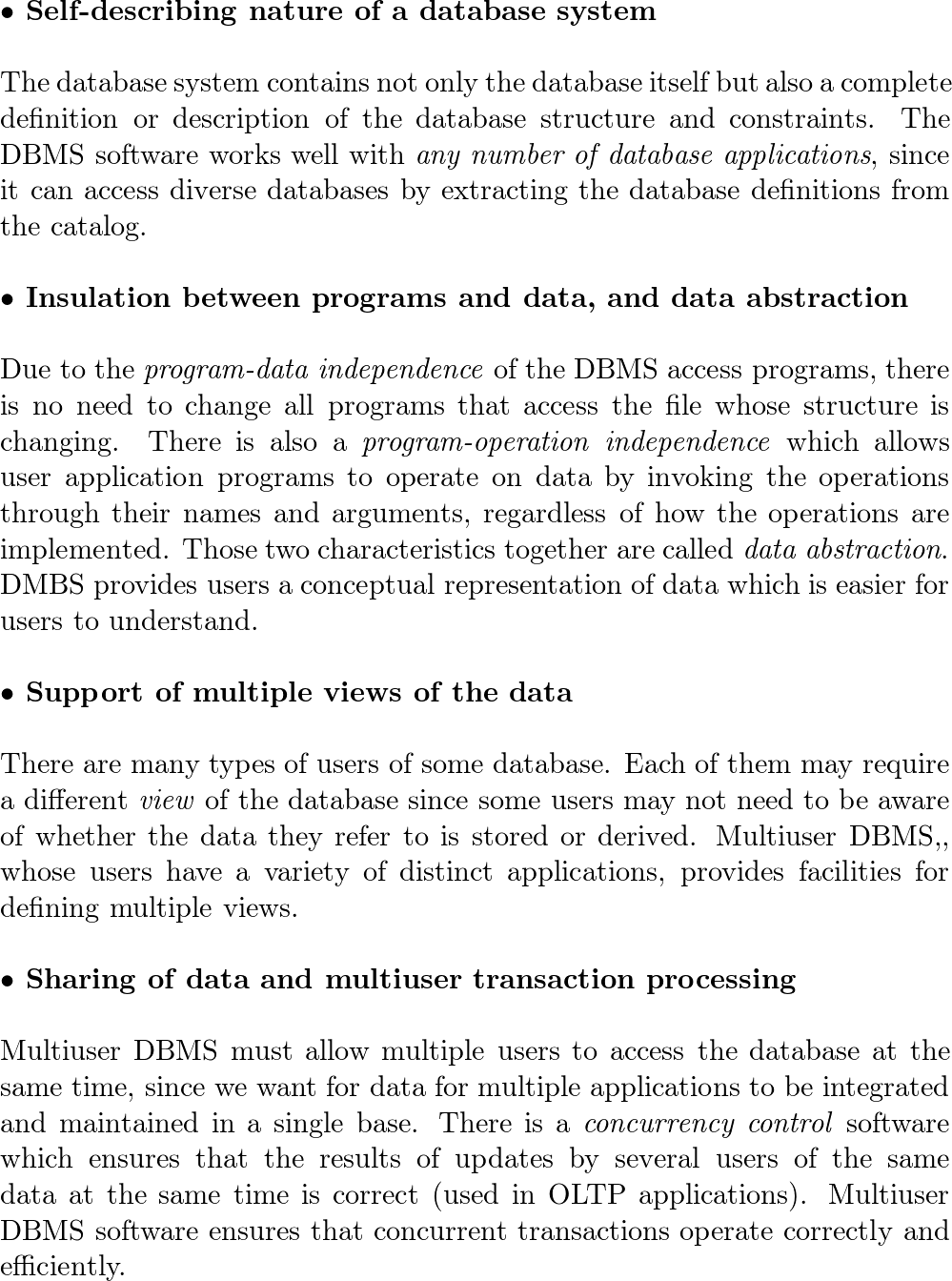
Transaction-processing application is a type of application that allows several users at the same time to update the same data in a controlled manner so that the result of the updates is correct.

**Q1.2**

Databases involve four main types of actions and they are: % for bullet points

* DEFINING – involves specifying the data types, structures, and constraints of the data to be stored in the database
* CONSTRUCTING – the process of storing the data on some storage medium that is controlled by the DBMS
* MANIPULATING – The database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini-world, and generating reports from the data
* SHARING – allows multiple users and programs to access the database simultaneously

**Q1.3**



**The difference with traditional file systems**:

In traditional file processing traditional**file processing**, data definition is part of the application programs so they are constrained to work with only one specific database, whose structure is declared in the application programs. In traditional file processing does not have a property of program-data independence does*not have a property of program-data independence*, since the structure of data files is embedded in the application programs, hence any changes to the structure of a file may require changing all programs that access that file

Self-describing nature of the database system

Insulation between programs and data, and data abstraction

Support of multiple views of the data

Sharing of data and multiuser transaction processing

**Q1.4**

**DBA** (database administrator) is responsible for authorizing access to the database, coordinating and monitoring its use and acquiring software and hardware resources as needed. DBA also handles security breaches and poor system response time.

**Database designers** are responsible for finding the data to be stored in the database and for choosing the appropriate structures that will represent and store this data. Database designers have to communicate with all possible database users to understand all requirements so that they can create a design that will meet all these requirements. After the database design is complete, they can work on the staff of the DBA. They also interact with each potential group of users to better understand and develop all the necessary views of the database.

**DBA** (database administrator) is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. ……

**Database designers** are responsible for finding the data to be stored in the database and for choosing the appropriate structures that will represent and store this data. ……

**Q1.5**

**Casual end users** while accessing the database may acquire different information each time. To specify their requests, they use a sophisticated database query interface. They are usually middle- or high-level managers or other occasional browsers.

**Naive** or parametric end uses parametric**end users** are the most numerous type of database end users. They are constantly querying and updating the database, using canned transactions.

**Sophisticated end users** are engineers, scientists, business analysts, and others who are familiar with the facilities of the DBMS so that they can implement their own applications that will meet their complex requirements.

**Standalone users** have personal databases. They create it by using program packages that provide user-friendly interfaces.

Casual, naive, or parametric, sophisticated, and standalone end users.

**Casual end users** while accessing the database may acquire different information each time. To specify their requests, they use a sophisticated database query interface. They are usually middle- or high-level managers or other occasional browsers.

**Q1.6**

**Controlling redundancy**

Redundancy in storing the same data multiple times could lead to several problems. There is a duplication of effort, storage space is wasted duplication*of effort, storage space is wasted* and files that represent the same data may become inconsistent. Data normalization in the database approach ensures consistency and saves storage space. However, in practice, it is sometimes necessary to use controlled redundancy controlled*redundancy* to improve the performance of queries (denormalization). DBMS should have the capability to control any redundancy to avoid inconsistencies among the files.

∙∙ Restricting unauthorized access restricting**unauthorized access**

DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions. The DBMS should enforce these restrictions automatically.

∙∙ Providing persistent storage for program objects**Providing persistent storage for program objects**

A complex object in C++++ or ����*Java* can be stored permanently in an object-oriented DBMS. That object is *persistent* since it exists after the termination of program execution and it can be retrieved later by another program. Object-oriented DBMS typically offer data structure compatibility with one or more object-oriented programming languages.

∙∙ Providing storage structures and search techniques for efficient query processing**Providing storage structures and search techniques for efficient query processing**

To speed up disk search for the desired records, DBMS often uses indexes. They are based on tree data structures or hash data that are modified for disk search. To obtain database records that we are querying, those records must be copied from the disk to the main memory. DBMS often has a *buffering* or *caching* module that maintains parts of the database in main memory buffers. DBMS also has query processing and optimization query*processing and optimization* module to choose an efficient query execution plan for each query based on the existing storage structures.

∙∙ Providing backup and recovery providing**backup and recovery**

DBMS's backup and recovery subsystem*backup and recovery subsystem* is responsible for recovery.

∙ Providing multiple user interfaces**Providing multiple user interfaces**

DBMS should provide a variety of user interfaces. For example, apps for mobile users, query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users and menu-driven interfaces and natural language interfaces for standalone users. DBMS often has a capability to provide Web GUI (graphical user interfaces) to a database.

∙∙ Representing complex relationships among data**Representing complex relationships among data**

DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships and to retrieve and update related data easily and efficient.

∙∙ Enforcing integrity constraints**Enforcing integrity constraints**

DBMS should provide capabilities for defining and enforcing integrity constraints for the data. Database designers should identify integrity constraints during the database design. DBMS should act differently for different types of constraints. Some of them can be specified to the DBMS and automatically enforced. Some must be checked by update programs. It can also happen that a data item is entered erroneously and still satisfies integrity constrains.

∙∙ Permitting inferencing and actions using rules and triggers**Permitting inferencing and actions using rules and triggers**

In deductive database systems *deductive database systems* DBMS compiles and maintains *rules* that can be specified declaratively. In today's relation database systems, it is possible to associate *triggers* with tables. Trigger is a form of rule activated by updates to the table which leads to performing some additional operations to some other tables. More used procedures to enforce rules are called stored procedures stored*procedures*. They are invoked appropriately when certain conditions are met. Even better functionality is provided by active database systems*active database systems* which provide active rules that can automatically initiate actions when certain events and conditions occur.

**Controlling redundancy**

Redundancy in storing the same data multiple times could lead to several problems. There is a duplication of effort, storage space is wasted duplication*of effort, storage space is wasted* and files that represent the same data may become *inconsistent*. Data normalization in the database approach ensures consistency and saves storage space. However, in practice, it is sometimes necessary to use controlled redundancy controlled*redundancy* to improve the performance of queries (denormalization). DBMS should have the capability to control any redundancy in order to avoid inconsistencies among the files.

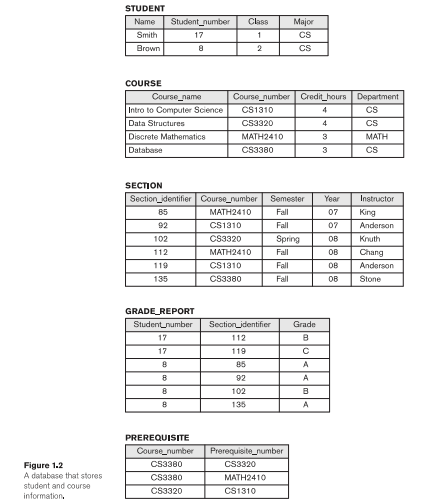
**Q1.7**

**Database system** deals with structured data*structured data*, while Informational**Informational** Retrieval (IR)**Retrieval (IR)** system deals with unstructured data*unstructured data*. There is no fixed schema*no fixed schema* in IR systems (there are various data models) and query models are free-form, while Database system is schema driven*schema driven*, relational model is predominant and query model is structured*relational model is predominant and query model is structured*. At Database systems there are rich metadata operations*rich metadata operations* and query returns data*query returns data*, while at IR systems search request returns list or pointers to documents*search request returns list or pointers to documents*. Maybe the most significant difference between those two systems is that the Database system gives results that are based on exact matching exact*matching and*, hence are always *correct*, while IR systems give results that are based on approximate matching and measures of effectiveness*that are based on approximate matching and measures of effectiveness*, so they may be imprecise and ranked *imprecise and ranked*.

To sum it up, there is no need for structured data in IR systems, but then the search results may be imprecise, while Database systems require structured data, but each well-structured query will give you correct results.

There is no need for structured data in IR systems, but then the search results may be imprecise, while Database systems require structured data, but each well-structured query will give you correct results.

**Q1.8**



First, we will give examples of some informal **queries** that could apply to the given database

All grades in Discrete Mathematics.

All grades from a student with the Name Brown.

All courses a certain student has a grade in.

All prerequisites of course Database.

Next, we will give examples of some informal **updates** that could apply to the given database:

∙∙ Change Credit hours of Intro to Computer Science to 55.

∙∙ Change the Grade for student Smith in Intro to Computer Science to B.

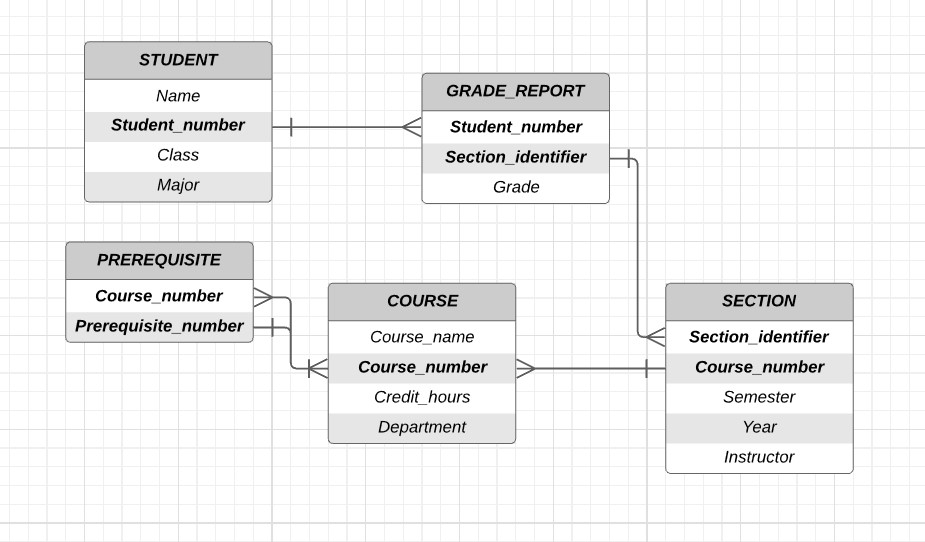
∙∙ Change the Major of a student whose name is Brown to MATH.

∙∙ Change the Class of student whose name is Smith to 22.

Queries: All grades in Discrete Mathematics; All grades from student with Name Brown ……

Updates: Change Credit hours of Intro to Computer Science to 55; Change Grade for student Smith in Intro to Computer Science to B ……

**Q1.10**



Above scheme is there to notice all the connections between different tables.

Each GRADE\_REPORT record is related to one STUDENT record and one SECTION record.

Each SECTION record is related to one COURSE record.

Each PREREQUISITE record is related to two COURSE records - one of the course and other of its prerequisite (since prerequisite number is also some course number)

Each GRADE\_REPORT record is related to one STUDENT record and one SECTION record.

Each SECTION record is related to one COURSE record.

Each PREREQUISITE record is related to two COURSE records - one of the course and the other of its prerequisite (since prerequisite number is also some course number)

**Q1.11**

For example, some users can be interested only in printing the transcript for certain students (that is an often required document). You can see the view in Figure 1.51.5 (a) is derived from the database in Figure 1.21.2 (a). That view is the TRANSCRIPT view. Also, some users may want to check whether the students have taken all the prerequisites of each course they are currently willing to register for. For that, you have the COURSE\_PREREQUISITE view in Figure 1.51.5 (b). Another example would be if some user is interested in calculating the average grade of some course. Then that user wants to see all grades for some course. That view (COURSE\_GRADES) would have a list of grades for each course (just like each course in the COURSE\_PREREQUISITE view has a list of prerequisites)

**Q1.12**

A few examples of integrity constraints would be:

Student numbers should be unique for each STUDENT record so that every student is unique and there is no inconsistency in data.

Course number should be unique for each COURSE record, since two courses with the same Course number would, for example, cause we don't know from which course a certain student has a certain grade (it could be in both with certain Course number we're looking at).

Next, the Course number in SECTION record must exist in COURSE record, otherwise that course does not exist and hence, students cannot have grades in it.

Student numbers should be unique for each STUDENT record so that every student is unique and there is no inconsistency in data

**Q1.13**

GIS implementations

GIS implementations often implement their data organization schemes for functions related to processing maps, physical contours, lines, polygons, etc.

Communication and switching systems

Designed by companies like ��&�*AT*&*T* - early manifestations of database software that was made to run very fast with hierarchically organized data for quick access and routing of calls.

Computer-aided design tools

CAD tools used by engineers have their own data and file management software for internal manipulations of drawings

**Q1.14(a)**

All columns that refer to CS as Computer Science (department) would have to be updated to CSSE as Computer Science and Software Engineering. Hence, columns **Course\_number** and **Department** in COURSE, **Course\_number** in SECTION, as well as columns **Course\_number** and **Prerequisite\_number** in PREREQUISITE need to be updated in that case

**Q1.14(b)**

Since every course is in some department, as we can see in table COURSE, then it is enough that Course\_number is some unique number, without a department prefix. We can always find out the department by looking at the COURSE table. For example, let the Course\_number for Intro to Computer Science be 13101310, etc. If Course\_number was structured that way, then, with the change of CS to CSSE, we wouldn't have to change anything in the SECTION table or PREREQUISITE table, just one column in the COURSE table (Department).

Review Questions

**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*.**

**Ans**

**Data model** is an abstract model that organizes and describes data, and how each element of data relates to another.

**Database schema** is a description of a database that is made during database design to best fit the needs of future users. It is not expected to change frequently.

**Database state** or **snapshot** is the data in the database at a particular moment in time (since the data in the database may change frequently). Another name for it is the current set of occurrences or instances in the database.

**Internal schema** is defined on the *internal level* of three-schema architecture for database systems. It describes the physical storage structure of the database. It uses a physical data model and gives details of data storage.

**Conceptual schema** is defined on the *conceptual level* of three-schema architecture for database systems. It describes the structure of the database for a community of users. Conceptual schema doesn't give details of physical storage structures, but it rather describes entities, data types, relationships, user operations, and constraints.

**External schema** is defined on the *conceptual* or *view level* of three-schema architecture for database systems. It describes just the part of the database certain user groups want to see and hides the rest.

**Data independence** is the capacity to change the schema at one level of a database system without having to change the schema at a higher level.

**DDL** or **data definition language** is used by the database administrator and by the database designers to define conceptual and internal schemas. In DBMS where there is a separation between the conceptual and internal level, DDL is used to specify the conceptual schema only. In most DBMSs, DDL is used to define both conceptual and external schemas. DDL compiler, within DBMS, processes DDL statements which identify descriptions of the schema constructs and store the schema description in the DBMS catalog.

**DML** or **data manipulation language** is a set of operations used to manipulate the database (retrieval, insertion, deletion, and modification of data).

**SDL** or **storage definition language** is used to specify the internal schema in DBMS where there is a clear separation between the conceptual and internal levels.

**VDL** or **view definition language** would be used for three-schema architecture to specify user views and their mappings to the conceptual schema.

**Query language** is a high-level data manipulation language (DML) used in a standalone interactive manner.

**Host language** is a general-purpose programming language that has DML commands. The belonging DML is called the data**sublanguage**.

**Database utilities** help the database administrator in managing the database system. Types of functions in utilities: loading, backup, database storage reorganization, and performance monitoring.

**Catalog** is used for the storage of schema descriptions by the DBMS.

**Client/server architecture** is used when a large number of PCs, workstations, file servers, printers, and other software and equipment are connected via a network. Each server in that network has its specific functionalities and becomes the specialized server. The resources provided by specialized servers can be accessed by client machines which provide different users with interfaces to use specialized servers and to use the machine to run local applications.

**Three-tier architecture** is a type of basic DBMS architecture created on a client/server framework. This architecture adds an intermediate layer (middle tier called the application server of the Web server) between the client and the database server. This intermediate layer runs application programs and stores business rules that are used to access data from the database server.

**N**-**tier architecture**, where *n* may be four or five, gives an architecture that divides the layers between the user and the stored data into finer components. Usually, the business logic layer is divided into multiple layers. This kind of architecture gives the advantage that any one tier can run on an appropriate processor or operating system platform and can be handled independently.

**2.2. Discuss the main categories of data models. What are the basic differences**

**among the relational model, the object model, and the XML model?**

**Ans**

The main categories of data models are **high-level** **representational model**, and **low-level**. *High-level* also called *conceptual data models* use concepts such as entities, their attributes, and relations between them. Those are data that users can see and understand. The second one, *representational* or *implementation* (not a standard term) data models hide many details of data storage on disk but can be implemented on a computer system directly. So users can see/understand data from this model, but not all of them. In the end, there are low-level or physical*data models* that provide concepts that describe details of how data is stored on the computer storage. Those concepts are meant for computer specialists, not for users.

The basic differences between the relational model, the object model, and the XML model are that in the relational model data and relation, data are shown in tables, the object model is inspired by object-oriented programming so there are objects and there are classes. XML model is the tree-structured data model.

Data model categories are: **high-level**, **representational**, and **low-level**

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

**Ans**

**Database schema** is a description of a database and it is specified during database design and is not expected to change frequently. Changes are usually needed as the requirements of the database applications change. It is the skeleton structure of the entire database. On the other side, the **database state** is data, the content of the database at a particular moment in time.

For example, when manipulating the values of data items in the database we change the current state (database state), but the main structure (database schema) stays the same.

**Database schema** is a description of a database and it is specified during database design and is not expected to change frequently. Changes are usually needed as the requirements of the database applications change. It is the skeleton structure of the entire database. On the other side, the database**state** is the data and content of the database at a particular moment in time.

**2.4. Describe the three-schema architecture. Why do we need mappings among**

**schema levels? How do different schema definition languages support this**

**architecture?**

**Ans**

**Three-schema architecture** separates the user applications from the physical database by defining schemas at three different levels: **internal level**, **conceptual level**, and **external level**. The internal level has an internal schema that describes the physical storage structure of the database. The conceptual level has a conceptual schema that concentrates on describing entities, data types, relationships, user operations, and constraints. It does not deal with the details of physical storage. The external level includes external schemas. The particular user group has a particular view of the database which a particular external schema describes.

Each user group has its view, that is, it refers to its external schema. To reach the actual data stored on a physical level, DBMS must transform the request from the certain user group specified on its external schema into the request familiar to the conceptual schema (lower level) and then into a request on the internal schema which is just above the physical level, that is, the stored database. That process of transforming the request and returning the correct external view of the result is called **mapping**.

For three-schema architecture, there is the **view definition language** (VDL). It specifies user views and their mapping to the conceptual schema. In most DBMSs, DDL is still used to define conceptual and external schemas. In relational DBMSs, the VDL is substituted by SQL to define user or application views.

**Three-schema architecture** separates the user applications from the physical database by defining schemas at three different levels: **internal level**, **conceptual level** and **external level**.

**2.5. What is the difference between logical data independence and physical data**

**independence? Which one is harder to achieve? Why?**

**Ans**

**Logical data independence** means that the conceptual schema can be changed without changing the external schemas or application programs. It may be needed to expand the database, to change constraints, or to reduce the database. **Physical data independence** means that the internal schema can be changed without changing the conceptual schema (therefore, the external schemas don't have to change too). Changes to the internal schema may be needed for some physical file reorganization - to improve the performance of retrieval or update.

Logical data independence is **harder** to achieve because it allows structural and constraint changes without affecting application programs, which is a strict requirement

Logical data independence - change of conceptual schema, without change of the external schemas or application programs.

Physical data independence - change of internal schema, without change of the conceptual schema.

Logical data independence is harder to achieve.

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

**Ans**

**High-level** or **non-procedural** DML can be used on their own to specify complex database operations. This DML can specify and retrieve many records in a single DML statement (set-at-a-time or set-oriented DMLs). It is usually used by casual end users.

On the other hand, **Low-level** or **procedural** DML must be embedded in a general-purpose programming language. This type of DML retrieves individual records or objects from the database and processes each separately (record-at-time DMLs). It is typically used by programmers

**Non-procedural** DML: can be used on their own, set-at-a-time or set-oriented DML, usually used by casual end users.

**Procedural** DML: must be embedded in a general-purpose programming language, record-at-time DML, typically used by programmers.

**2.7. Discuss the different types of user-friendly interfaces and the types of users**

**who typically use each.**

**Ans**

∙ **Menu-based interfaces for Web clients or browsing** have a list of options (menus) that lead the user through the formulation of the request. Pull-down menus are popular in Web-based user interfaces or browsing interfaces.

∙**Apps for mobile devices** are for example apps for banking, reservations, insurance companies, etc. They are made so that mobile users have access to their data. This kind of interface usually requests user login and it usually provides a limited menu of options for mobile access.

∙**Forms-based interfaces** display a form to each user which can fill out the form to insert new data. This kind of interface is usually designed for naive users.

∙**Graphical user interfaces** (GUI) display a schema to the user in a diagrammatic form. By manipulating the diagram, the user can specify a query. Both menus and forms are used in GUIs.

∙**Natural language interfaces** try to understand the given requests in some language. It has its schema and dictionary of important words which it uses to figure out the meaning of request. If the interpretation is successful, then the request is forwarded to the DBMS for processing (otherwise, a dialogue with the user is started to clarify the request).

∙**Keyword-based database searches** are similar to Web search engines. They use predefined indexes on words and ranking functions to retrieve resulting documents.

∙**Speech input and output** provides limited use of speech as an input query and answer to a question or result also in a speech form using a library of predefined words and conversion from text or numbers into speech. Examples of applications are inquiries for telephone directory, flight arrival/departure, credit card account information, etc.

∙ **Interfaces for parametric users** (such as bank tellers) is used when user performs small set of operations repeatedly. System analysts and programmers design and implement interface for each class of naive users.

∙**Interfaces for the DBA** have privileged commands that can be used only by the DBA staff (such as creating accounts, setting system parameters, granting account authorization, changing a schema or reorganizing the structure of the database).

∙**Menu-based interfaces for Web clients or browsing** have list of options (menus) which lead the user through the formulation of the request. Pull-down menus are popular in Web-based user interfaces or in browsing interfaces.

**2.8. With what other computer system software does a DBMS interact?**

**Ans**

DBMS interacts with the operating system while accessing a database or the catalog. If the computer system is shared by many users, the OS can schedule DBMS disk access requests. If the computer system is dedicated to running the database server, the DBMS will control the main memory buffering of disk pages. DBMS also interacts with compilers for general-purpose host programming languages and with application servers and client programs

OS, compiler, application servers, client programs

**2.9. What is the difference between the two-tier and three-tier client/server**

**architectures?**

**Ans**

**The two-tier architecture** is called that way because the software components are distributed over **two systems - client and server**. Due to the emergence of the Web, the roles of clients and servers have changed, which leads to a three-tier architecture.

**Three-tier architecture** added an **intermediate layer** (middle layer) between the client and database server which is called the application server or Web server. Therefore, in a three-tier architecture, there are three layers - client, application (or Web) server, and database server-**three layers - client, application (or Web) server, and database server**.

In two-tier architecture there are two layers (client and server) and in three-tier architecture added another intermediate layer between client and server called the application server or Web server.

**2.10. Discuss some types of database utilities and tools and their functions.**

**Ans**

**Loading utility** loads existing data files into the database. The utility automatically reformats the data from the source to the target database file structure. Some vendors offer **conversion tools** that generate loading programs with source and target database storage descriptions. Hence, the main function of this utility is **loading**.

**Backup utility** makes a backup copy of the database by putting the whole database onto tape or other mass storage medium. This copy can be used to restore the database in case of disk failure. Hence, the main function of this utility is **backup**.

Utility for **database storage reorganization** is used to reorganize a set of database files into different file organizations and create new access paths to improve performance.

Utility for **performance monitoring** monitors database usage and gives whole statistics to the database administrator. The DBA then uses those statistics to come up with decisions on how to reorganize files, to improve performance, if necessary.

There are also utilities for sorting files, data compression, monitoring access by users, interfacing with the network, etc.

**CASE tools** are used in the design phase of database systems. **Data dictionary system** is another powerful tool in large organizations used as a storage for catalog information about schemas and constraints. It also stores design decisions, usage standards, application program descriptions, and user information. **Application development environments** (PowerBuilder, JBuilder) are used for developing database applications and help in database design, GUI development, querying and updating, etc. **Communications software** is also quite useful. It allows users to access the database remotely.

**Loading utility** loads existing data files into the database. The utility automatically reformats the data from the source to the target database file structure. Some vendors offer **conversion tools** that generate loading programs with source and target database storage descriptions. Hence, the main function of this utility is **loading**

**2.11. What is the additional functionality incorporated in *n*-tier architecture**

**(*n* . 3)?**

**Ans**

The advantage of *n*-tier architecture in applications and also the additional functionality is distributing programming and data throughout the network. Also, any one tier can run on an appropriate processor or operating system platform and can be handled independently.

The advantage of *n*-tier architecture in applications and also the additional functionality is distributing programming and data throughout the network.

**Exercises**

**2.12. Think of different users for the database shown in Figure 1.2. What types of**

**applications would each user need? To which user category would each**

**belong, and what type of interface would each need?**

**Ans**

Since the database is made for the university, then we can suppose that it will be used by **instructors**, **students**, **student office**, and **DBA** staff (necessary to administer the database). This database would be a relational database, which is a dominant type of database system for traditional database applications. The instructor would then have access to the STUDENT, COURSE, SECTION, and GRADE\_REPORT tables. The instructor doesn't need to have access to the PREREQUISITE table because we can suppose that the student office would not let the student sign up for a course that he/she cannot (the student office will also be responsible for changing the prerequisites). With the given access, the instructor can change a grade for any student and that is the main responsibility at the end of the semester. Hence, he would have a view of the table with all of his students for each course he holds, so that he can change their grades. For a student, it is enough that he can access table STUDENT and GRADE\_REPORT where he can find everything about his achievements at college, that is, he would have a view of his table which would consist of his courses and grades. Everything else he/she can find out by visiting the student office. Hence, the student office has access to all tables and can change, update, and access all records. It has a view of all tables and also some derived tables, such as TRANSCRIPT and COURSE\_PREREQUSITES from Figure 1.5. DBA administers this whole database gives certain access to certain kinds of users and makes sure that everything runs smoothly. Therefore, for **students,** some **keyword-based database search interface** is enough, since they won't be changing anything in the database, but rather searching themselves in the database and they are **casual end users**. **Instructor and student offices** would have some kind of **GUI (graphical user interface)** that utilizes both menu and forms with different permissions and they are **naive or parametric end users**. DBA**DBA** would have an **interface for the DBA** and this kind of interface would allow privileged commands that can be used only by the DBA staff. They would create accounts, potentially change database schema, etc.

Since the database is obviously made for the university, then we can suppose that it will be used by **instructors**, **students**, **student office** and **DBA** staff (necessary to administer the database).

**2.13. Choose a database application with which you are familiar. Design a schema**

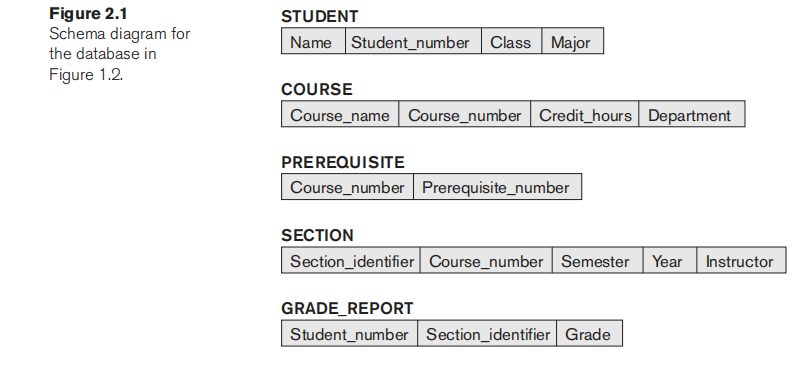
**and show a sample database for that application, using the notation of Fig**

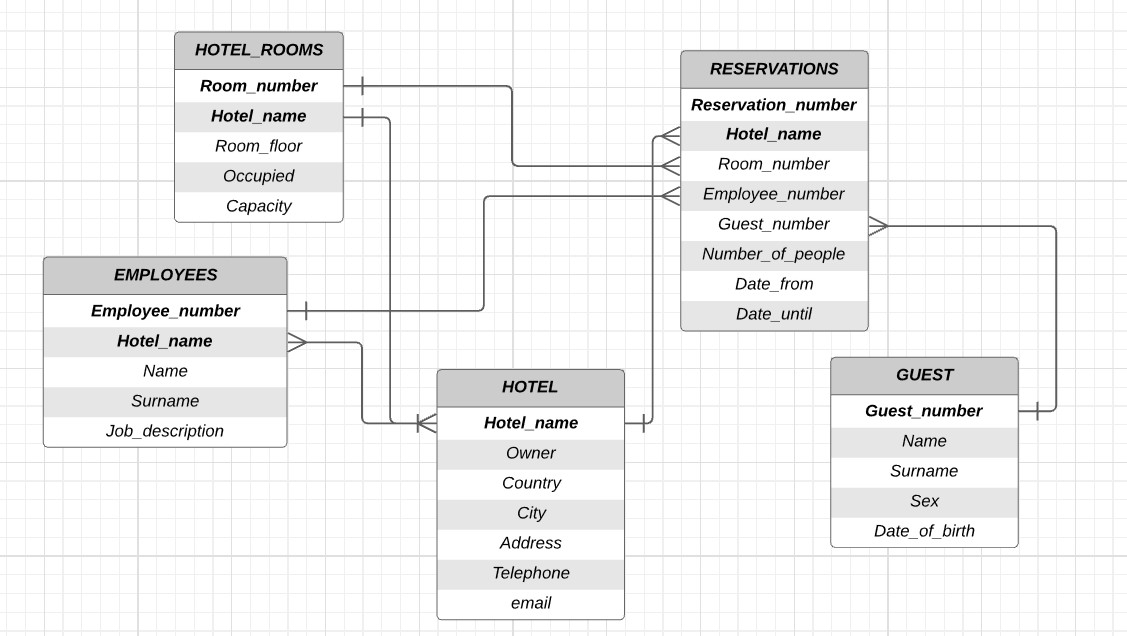
**ures 1.2 and 2.1. What types of additional information and constraints**

**would you like to represent in the schema? Think of several users of your**

**database, and design a view for each.**

**Ans**





The schema for the relation database above represents reservations in the group of hotels. Each hotel has its unique name within the group of hotels. That name represents the primary key in tables EMPLOYEES (along with primary key *Employee number*), HOTEL\_ROOMS (along with primary key *Room number*), and RESERVATIONS (along with primary key *Reservation numbers*). All primary keys are unique! Also, notice that in RESERVATIONS we have foreign keys Room\_number, Employee\_number*Room\_number, Employee\_number*, and *Guest\_number*. They are in the table so that reservations would be easy to query and update, but it doesn't mean that the same guest cannot book the same room in some other reservation. Also, there should be constraints considering the dates of reservation. That constraint would have to check the *Occupied* attribute in HOTEL\_ROOMS (let 11 be occupied and 00 not occupied value), so that they cannot book the same room twice at the same time. GUEST and EMPLOYEES tables are there to keep track of all guests that are currently in each hotel and of all employees currently working in hotels (whether as receptionists, cooks, maids, etc.). *Room number* in HOTEL\_ROOMS is a unique number (for example 101101 for the first room on the first floor) in a certain hotel with *Hotel\_name*. We can see that we can easily derive the floor on which the given room is from its *Room\_number*, but it is easier to have *Room\_floor* already as an attribute. That would be useful if the receptionist wanted to see how many rooms on floor 22 are available. Then he/she wouldn't have to first select all the rooms that begin with 22 (we suppose that the hotel has less than 2020 floors) - which are on the second floor - but the receptions would just have to select all the hotel rooms on *Room\_floor* where *Occupied* =0=0. Users of this database would for example be receptionists in each hotel. Receptionists of some hotels would only see the records of their hotel. They would have a view of all the rooms, guests, and reservations of the hotel they are working in. The owner of the hotel would see all that and also its employees. The person who is in charge of all groups of hotels would see all data for all hotels.

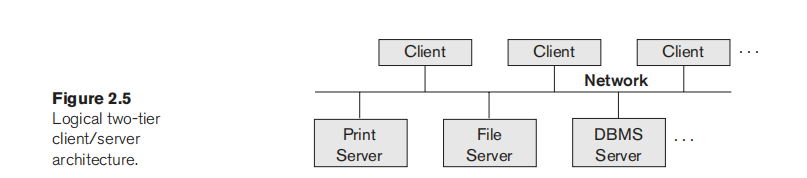
A simple example is a relational database for a group of hotels and their reservations.

**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?**

**Ans**



If I was designing a Web-based system to make airline reservations and sell airline tickets, I would choose the three-tier architecture three-tier*architecture*. Because the chosen server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server. So when the client selects a flight for which he/she wants to book or buy a plane ticket, data is perceived by the application and stored in the database. These are the 3 steps for the selected system (the middle tier would accept the client request process it and store it in the database). Also, the client can check what he/she booked or bought so the database sends data to the middle tier which would give the requested data to the client/user.

Other architectures wouldn't be the best choice in this situation because they wouldn't be safe enough and wouldn't process and check correct information to save in the database or give back to the client.

If I was designing a Web-based system to make airline reservations and sell airline tickets, I would choose the three-tier architecture three-tier*architecture*.

**2.15. Consider Figure 2.1. In addition to constraints relating to the values of col**

**umns in one table to columns in another table, there are also constraints that**

**impose restrictions on values in a column or a combination of columns**

**within a table. One such constraint dictates that a column or a group of col**

**umns must be unique across all rows in the table. For example, in the**

**STUDENT table, the Student\_number column must be unique (to prevent two**

**different students from having the same Student\_number). Identify the col**

**umn or the group of columns in the other tables that must be unique across**

**all rows in the table**.

**Ans**

*Course number* in COURSE table must be unique across all rows in that table since every course in that table must be mentioned only once and each course has its unique course number. Course name is also unique in that table if there aren't any courses with the same name but in different departments. That is possible, but it is not ordinary, so we can say that *Course name* would also be unique. Then we can ask ourselves why there is a Course number as a unique constraint? Well, it is much easier to query with some short word or number, which appears also in other tables, than with the whole Course name - that can be long (and as we said potentially, but not necessarily unique). Section identifier in the SECTION table is also unique since each section is defined by some course in a certain semester, or year and held by a certain instructor. There are no multiple-row records containing the same *Section identifier* in the SECTION table.

The course number in the COURSE table and Section identifier in the SECTION table are unique. Course name in COURSE can also be unique (it depends on the university rules).

**Review Questions**

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

**Ans**

**A high-level data model** helps with creating a conceptual schema for the database once all the requirements have been documented and analyzed.

It provides concepts that are used in expressing detailed descriptions of entity types, relationships, and constraints.

These concepts can also be used for better communication with nontechnical database users because they don't include implementation details.

**A high-level data model** helps with creating a conceptual schema for the database

**3.2. List the various cases where the use of a NULL value would be appropriate.**

**Ans**

A **NULL** value can replace any type of value and represents non-existent information. It is applied when the entity does not have an applicable attribute value, for example: when the employer has a database with its employees and has their data such as  
NAME SURNAME ADDRESS NUMBER\_MOBILEPHONE1 NUMBER\_MOBILEPHONE2 NUMBER\_MOBILEPHONE3.... NUMBER\_MOBILEPHONE2 and NUMBER\_MOBILEPHONE3 can be **NULL** values because there is a possibility that the worker has only one mobile phone and only one number. Or in the database from the university where the students and entities are: NAME NAME2 NAME3 SURNAME SURNAME2 SURNAME3 in case a student has more than one name or more surnames. Whoever has one name and one surname, the rest is marked as a NULL value.

Also, we use a **NULL** value when we do not know the value of the attribute, for example: in a marketing company, there is a database with promoters doing the work in that company. They also have a database with data on promoters. In some promotions, it is necessary to arrange a clothing combination and there are sections, entities such as CLOTHES\_SIZE and FOOTWEAR\_SIZE where some of the promoters may not want to participate in such promotions or do not want to state the size of clothing or footwear. And in this case, it is a NULL value, but we know that the value exists but we don't know what it is exactly so we call it missing.

A **NULL** value can replace any type of value and represents *non-existent* information, *unknown* information, or unknown but existing *unknown but existing* (*missing*).

**NULL values** are used in cases where a problem occurs in adding a value to an attribute. There are two categories of cases for use of NULL value: **not applicable** and **unknown**. The **unknown** category can be further divided into two subcategories: **missing** and **not known**.

Here is a brief explanation of every case:

∙**not applicable:** A particular entity does not have an applicable value for some of its attributes. For example, entity type BOOK has an attribute Illustrator, but some books in database do not have any illustrations, hence, no illustrator. In that case, the Illustrator attribute is given a NULL value.

∙**missing:** An attribute value exists, but it is missing or it is not provided. For example, entity type BOOK has attributes Publication\_year and Publication\_month, but some older books have only recorded data for their year of publication and not for the month. In that case, the Publication\_year attribute is given a proper value and the Publication\_month is given a NULL value.

∙**not known:** This case occurs when it is not known if the attribute value exists. For example, entity type BOOK has an attribute Author\_e-mail, but it is not known if the Author possesses an e-mail address. In that case, the Author\_e-mail attribute is given a NULL value.

There are two categories of cases for use of NULL value: **not applicable** and **unknown**. The **unknown** category can be further divided into two subcategories: **missing** and **not known**.

**3.3. Define the following terms: *entity, attribute, attribute value, relationship***

***instance, composite attribute, multivalued attribute, derived attribute, com***

***plex attribute, key attribute,* and *value set* (*domain*).**

**Ans**

**Entity** is an object from the real world with either physical or conceptual independent existence.

For example, BOOK, CAR, HOUSE, CUSTOMER, COMPANY, WORK HOURS, SALARY, RELATIONSHIP STATUS.

**Attribute** is a specific property that describes an entity. For example, entity BOOK may have attributes Name, Author, Price, Number\_of\_pages, Publication etc.

**Attribute value** is a single value or a set of values that describes an attribute of particular entity. Attribute values come from the domain of values (or value set, defined at the bottom). For example, entity BOOK has attribute Number\_of\_pages. Domain of the Number\_of\_pages attribute is integers between 2 and 2000  so we can assign value 450 to the attribute.

**Relationship instance** is an association of entities that represents relationship among entities similarly to the relationship of the miniworld. Limit of entities of each participating entity type in a relationship instance is set to 1.

For example, entity types BOOK and AUTHOR participate in the relationship STILL\_WORKS\_ON. One relationship instance connects *Dawn of Dragons* and *John Johnson*.

**Composite attribute** is a type of attribute that can be divided into smaller parts that can be represented by more basic independent attributes. This type is used in modeling when the user needs to refer to an attribute sometimes as a unit and in other times only to its components. If the composite attribute is formed as a hierarchy, then the value of the attribute is the concatenation of the values of its subparts.

For example, the Publication attribute of the BOOK entity can be divided into Publisher, Date\_of\_publication, and Place\_of\_publication.

**Multivalued attribute** is an attribute that can have a different number of values.

For example, the Author attribute of the BOOK entity type can have more value because the book can be written by more than one author. This way, the set of names of authors becomes the value for this multivalued attribute.

**Derived attribute** is an attribute that can be determined from another attribute, entity or relationship.

For example, we have entity AUTHOR which has attributes Date\_of\_birth and Age. Age attribute can be calculated or *derived* from the Date\_of\_birth attribute and current time. The second example is derivation from the relationship. Imagine entities BOOK and AUTHOR, every BOOK entity has attribute Author (this is the way of implementing a relationship WRITTEN\_BY) and every AUTHOR entity has attribute Number\_of\_written\_books. There is no need to count and update the number when it can easily be computed from the related entity BOOK.

**A complex attribute** is a combination of composite, single-valued, and multivalued attributes. It is represented as a grouping of composite attributes in parentheses and multivalued attributes in braces separated with commas.

For example, if we had only the entity BOOK and no entity AUTHOR, we could get a complex attribute Author that can include more than one author and looks like this {Author\_name, {Address(Street, Street\_number, Address\_phone)}, Date\_of\_birth(Day, Month, Year) }.

**Key attribute** is an attribute or a set of attributes with different values for each distinct entity. An entity can have more than one key*attribute*. For example, entity BOOK has the attribute ISBN. That is a unique number specified for each book. Book\_name can not be a *key attribute* because there are books with the same name, but a combination of attributes Book\_name and Author can be a key attribute (set of attributes in this case).

**A value set (domain)** is a set of values that can be assigned to an attribute of a particular entity. For example, we have entity BOOK and its attributes Author and Number\_of\_pages. The value set of the Author attribute is a set of strings separated with spaces and the value set of Number\_of\_pages attribute is a set of integers from 2 to 2000.

Definitions and examples of **Entity**, **Attribute**, **Attribute value**, **Relationship instance**, **Composite attribute**, **Multivalued attribute**, **Derived attribute**, **Complex attribute**, **Key attribute**, and **Value set**.

**3.4. What is an entity type? What is an entity set? Explain the differences among**

**an entity, an entity type, and an entity set.**

**Ans**

**The entity type** is a definition for a collection of entities that have the same attributes. Each entity type has its *name* and *attributes*.

**An entity set** is a set of instances of entities, of a particular entity type, with attribute values assigned.

The difference between **entity**, **entity type**, and **entity set** is that **entity** is a concept from the mini-world that we want to represent in the database, and **entity type** is used to describe the schema or plan for modeling an entity and its assigned attributes and **entity set** is a complete collection of entity instances of one particular type.

**The entity type** is a definition for a collection of entities that have the same attributes. Each entity type has its *name* and *attributes*.

**An entity set** is a set of instances of entities of a particular entity type.

The difference between **entity**, **entity type**, and **entity set** is

**3.5. Explain the difference between an attribute and a value set.**

**Ans**

The difference is that each **attribute**, which is defined as the particular properties that describe an entity, has its own **domain**, and **value set**. For example, if the database is STUDENT, entities can be NAME, SURNAME, MAJOR ... Each of these entities has its own attributes like Mark, Sarah... that has their own domain (in this case, length). Therefore, the student's name can be from 1 to 100 characters. (assuming names cannot be longer than 100 characters).

Every attribute has its own domain and value set.

**Attribute** is a specific property assigned to an entity. The attribute of an entity type has a value for every entity in the entity set. All possible values for a particular attribute come from a specific collection. That collection is called the **value set**. **Value set** specifies **data type** or **data types** that can be used when assigning value or values to an attribute. **Data types** are similar to those in programming, for example, integers, floating point numbers, strings with whitespaces, date formats DD/MM/YYYY, etc.

**Attribute** is a specific property assigned to an entity. The attribute of an entity type has a value for every entity in the entity set. All possible values for a particular attribute come from a specific collection. That collection is called the **value set**. **Value set** specifies

**3.6. What is a relationship type? Explain the differences among a relationship**

**instance, a relationship type, and a relationship set.**

**Ans**

**The relationship type** is a *definition* of a set of associations among a defined number (*n*) of entity types.

**Relationship instance** is an association of entities that represents that these entities are somehow related in the mini-world. There can be no more than one entity of each type in a relationship instance.

The set of associations among entities is called a **relationship set**.

We can illustrate the differences in an example.

Let's say there are two entity types BOOK and CUSTOMER. One **relationship type** between those entities is called IS\_BORROWED\_BY.

Next, let there be entities of BOOK entity type with Name attributes *Dawn of Dragons* and Dawn of Dragons 2: The Rabbit *Dawn of Dragons 2: The Rabbit* and entities of CUSTOMER entity type with Name attributes *Drake* and *Josh*.

If *Josh* borrowed Dawn of Dragons 2: The Rabbit *Dawn of Dragons 2: The Rabbit* we would get a **relationship instance** relating *Josh* and Dawn of Dragons 2: The Rabbit *Dawn of Dragons 2: The Rabbit* by relationship IS\_BORROWED\_BY. That relationship**instance** is the element of the relationship**set**.

Furthermore, if *Drake* also borrowed Dawn of Dragons 2: The Rabbit *Dawn of Dragons 2: The Rabbit*, those two would be part of a new **relationship instance** that is also an element of the **relationship set**.

**The relationship type** is a *definition* of a set of associations among a defined number (*n*) of entity types.

**3.7. What is a participation role? When is it necessary to use role names in the**

**description of relationship types?**

**Ans**

The **participation role** is a title given to each entity type that participates in a particular relationship type, which helps in explaining the functionality of the participation.

For example, let's say entity types BOOK and AUTHOR participate in relationship type IS\_WRITTEN\_BY. Then the role of *creation* or *product* is given to entities of entity type BOOK, and the role of *creator* or *writer* is given to entities of entity type AUTHOR.

Using role names is necessary only when one entity type participates with more than one instance in a relationship type and has different functionalities for each instance. These relationship types are known as **self-referencing** or **recursive relationships**.

Imagine there is a bookshop where you can either buy or sell books. Workers of the bookshop record each transaction of customers exchanging books. In their database, the CUSTOMER entity participates in relationship type BOUGHT\_BOOK\_FROM twice. One customer instance has the role of *seller* and the other has the role of *buyer*.

The **participation role** is a title given to each entity type that participates in a particular relationship type, which helps in explaining the functionality of the participation.

Using role names is necessary only when one entity type participates with more than one instance in a relationship type and has different functionalities for each instance.

**3.8. Describe the two alternatives for specifying structural constraints on**

**relationship types. What are the advantages and disadvantages of each?**

**Ans**

An alternative notation for specifying structural constraint is known as the **min-max** ( (min, max) ) constraint. It includes two alternatives for specifying structural constraints of *participation* and *cardinality*.

**Min-max** notation is represented by replacing the standard cardinality ratio of the relationship type with two integers.

∙**Cardinality**

The first integer placed on a line leading to the relationship-type diamond-shaped box is the *minimum* and the second one is the *maximum*. *Minimum* represents the minimal required number and *maximum* represents the maximal possible number of entities that can participate in a single relationship instance of that relationship type.

The advantages of this method are better precision of defining cardinalities and easier to represent structural constraints for relationship types of some higher degree. The disadvantage is that it isn't sufficient to specify some types of key constraints on relationship types of degree 3 or more.

∙**Participation**

This alternative for displaying the *participation* of an entity type in a relationship type replaces the single/double-line single*/double-line* notation with an integer. The first number or the *minimum* plays the role of defining *total* or partial*participation*. If it is set to 00, the participation is *partial* and if set to ">0>0" the participation is *total*.

The advantage of this method is a simpler notation without interweaving of single and double lines. The disadvantage is in displaying key constraints of relationship types of a higher degree

An alternative notation for specifying structural constraint is known as the **min-max** ( (min, max) ) constraint. It includes two alternatives for specifying structural constraints of participation and cardinality.

∙**Cardinality** First integer placed on a line leading to

∙**Participation** This alternative for displaying *participation*

**3.9. Under what conditions can an attribute of a binary relationship type be**

**migrated to become an attribute of one of the participating entity types?**

**Ans**

A necessary condition for an attribute of a relationship type to be migrated to one of the participating entities is that the relationship type has to be either **1:1** or **1:N** cardinality.

In the case of **1:1** cardinality, the database schema designer decides to which entity type should the attribute be moved to. For example, entity types BOOK and READER participate in relationship type IS\_BORROWED\_BY that has an attribute Return date. The return date attribute can be migrated to the BOOK entity type so we can check when some book is supposed to be returned, or it can be migrated to the READER entity type so the reader can check when he is supposed to return the borrowed book.

In the case of **1:N** cardinality, the attribute can only be moved to the N side of the relationship. Let's say there is a competition to win a signed copy of some books. Entity type COMPETITOR is related to entity type BOOK via relationship type IS\_COMPETING\_FOR that has attribute Entered competition. That relationship type is of cardinality 1:N because many competitors can compete for a single book. The entered competition attribute cannot be migrated to the BOOK entity type because every competitor has its own starting time, so it can only be migrated to the COMPETITOR entity type.

A necessary condition for an attribute of a relationship type to be migrated to one of the participating entities is that the relationship type has to be either **1:1** or **1:N** cardinality.

In case of **1:1** cardinality, the database schema designer decides

In case of **1:N** cardinality, the attribute can only be moved to

**3.10. When we think of relationships as attributes, what are the value sets of these**

**attributes? What class of data models is based on this concept?**

**Ans**

Let's say we have a particular entity type A, and it has an attribute that represents the relationship with entity type B. There are two possible value sets relationship as attribute *relationship as attribute* can have.

If the relationship is of a cardinality 1 : 1, then the attribute representing the relationship is singlevalued*singlevalued*, and the attribute's value set is a set of all entities*set of all entities* of entity type B (or B entity set*B entity set*).

If the relationship is of any other cardinality, then the attribute representing the relationship is multivalued, and the attribute's value set is a power set*power set* (set of all sets*set of all sets*) of entity set B. This means that multivalued attribute can have any combination of entities from entity set B as its value.

The functional data models**functional data models** is a class of data models that is based on concept of relationships as attributes*relationships as attributes*.

In object*object* and relational databases*relational databases* reference attributes**reference attributes** are used to represent relationships.

There are two possible value sets relationship as attribute*relationship as attribute* can have.

If the relationship is of a cardinality 1 : 1, then the attribute \ldots

If the relationship is of any other cardinality, then the attribute \ldots

The functional data models**functional data models** is a class of data models that \ldots

**3.11. What is meant by a recursive relationship type? Give some examples of**

**recursive relationship types.**

**Ans**

**Recursive** or self-referencing**self-referencing** relationship type means that there is more than one instance of entity type participating in a single relationship instance and each of the entity instances has a different role in that relationship.

∙∙ Example 1:**Example 1:** Two entities of entity type BOOK can participate in the relationship named IS\_PREDECESSOR\_OF. Book Dawn of Dragons 2: The Rabbit*Dawn of Dragons 2: The Rabbit* is related to book Dawn of Dragons*Dawn of Dragons*. There are two roles in that relationship type, one being predecessor*predecessor* and the other being successor*successor*.

∙∙ Example 2:**Example 2:** Imagine there is a library where you can only borrow books from other people that have membership. All the people with membership would be added to the database as entities of entity type MEMBER, and there would be a relationship between two MEMBER entities with name BORROWED\_BOOK\_TO. The MEMBER that is the rightful owner of the borrowed book plays the role of lender*lender* and the MEMBER who borrows that book plays the role of borrower*borrower*

**Recursive** or self-referencing**self-referencing** relationship type means that there is more than one instance of entity type participating in a single relationship instance and each of the entity instances has a different role in that relationship.

∙∙ Example 1:**Example 1:** Two entities of entity type BOOK can participate in \ldots

∙∙ Example 2:**Example 2:** Imagine there is a library where you can \ldots

**3.12. 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*.**

**Ans**

*Concept of weak entity types* is used in data modeling when we want to specify a new entity type similar to the one we already have, but we want this new entity type to have specific attributes and to be able to participate in specific relationships.

If the newly made entity type doesn't have key attributes, it is called weak entity type**weak entity type** and it needs to be related to a strong entity type**strong entity type** (directly or indirectly) that has key attributes of its own.

**Owner (or identifying) entity type** is a title for entity type that identifies weak entity types related to it. Weak entity types include one of their attributes when combining with owner entity type.

Weak entity type**Weak entity type** is an entity type that does not have key attributes of its own.

Identifying relationship type**Identifying relationship type** is a title for relationship type that relates one weakr entity type to one owner entity type. In this relationship type, weak entity has a total participation constraint*total participation constraint*, meaning it cannot exist without its owner entity type that identifies it.

Partial key (or discriminator)**Partial key (or discriminator)** is an attribute or set of attributes of a weak entity type that distinguishes weak entities related to the same owner entity type.

*Concept of weak entity types* is used in data modeling when we want to \ldots

Owner (or identifying) entity type**Owner (or identifying) entity type** is a title for entity type that \ldots

Weak entity type**Weak entity type** is an entity type that does not have \ldots

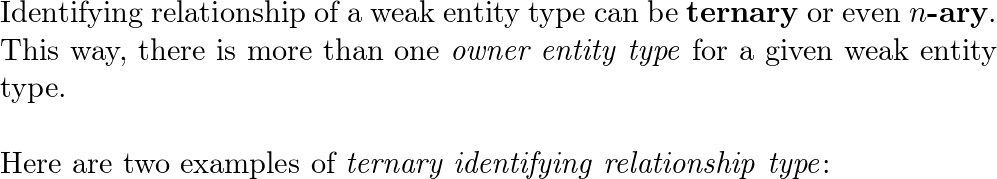
Identifying relationship type**Identifying relationship type** is a title for relationship type that \ldots

Partial key (or discriminator)**Partial key (or discriminator)** is an attribute or set of attributes of a \ldots

**3.13. Can an identifying relationship of a weak entity type be of a degree greater**

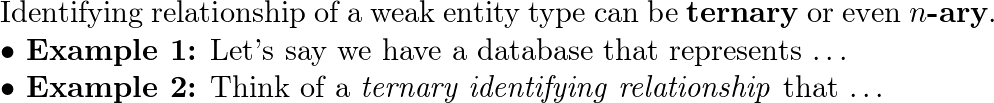
**than two? Give examples to illustrate your answer.**

**Ans**



∙ Example 1:**Example 1:** Let's say we have a database that represents music recording studio. The studio holds an event for new artists where every new artist gets an opportunity to record a song, in one studio room, which will be published if the publishers approve it. We have entity types ARTIST, ROOM, DEMO, and PUBLICATION. In this case DEMO is a weak entity type that participates in ternary identifying relationship*ternary identifying relationship* RECORD with strong entity types ARTIST and ROOM. Partial key of DEMO entity type is Hour of recording. It is not unique because many DEMO instances may be recorded in different rooms by different artists. DEMO is also related to PUBLICATION via relationship type IS\_PUBLISHED which gives a selected DEMO a Name before making it into a PUBLICATION instance. Both ARTIST and ROOM are owner entity types of DEMO entity type.

∙ Example 2:**Example 2:** Think of a ternary identifying relationship*ternary identifying relationship* that involves strong entity types STUDENT and PROFESSOR and weak entity type CHALLENGE that is called ASSIGN. CHALLENGE is further related to PRIZE entity type via relationship IS\_AWARDED. Professor can give different challenges to students, and students can ask different professors for a challenge. There can be more than one CHALLENGE between particular STUDENT and PROFESSOR. CHALLENGE is a weak entity type that has specific attribute Chapter that is a partial key. CHALLENGE, STUDENT and PROFESSOR are related via ternary identifying relationship*ternary identifying relationship* and if the given CHALLENGE is completed, its information is stored PRIZE entity type via the IS\_REWARDED relationship.

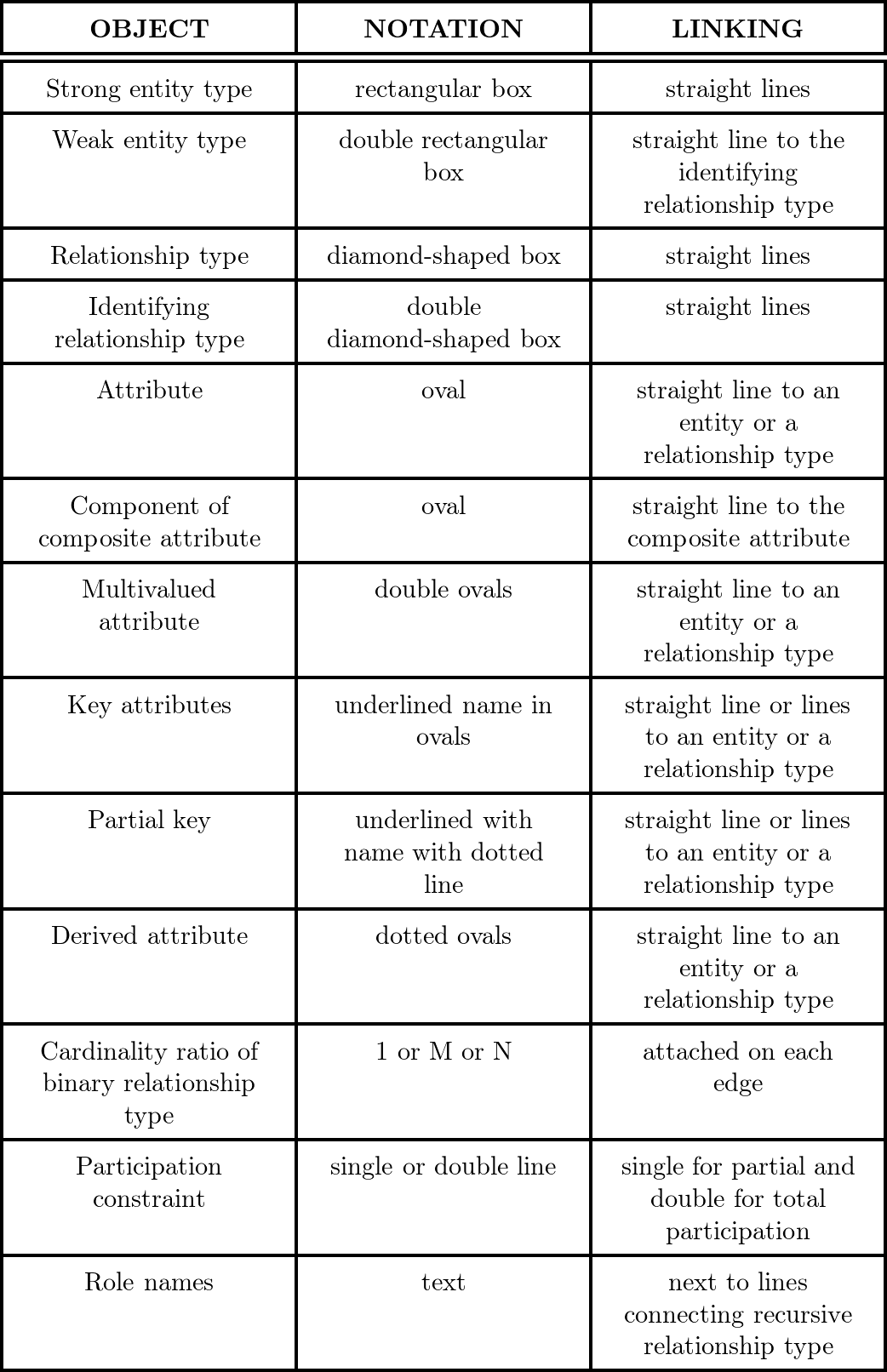


**3.14. Discuss the conventions for displaying an ER schema as an ER diagram.**

**Ans**

**ER schema** is not supposed to change frequently so the emphasis in ER diagram**ER diagram** is on types rather than instances.

Here is the standard notation for displaying specific database objects and their connections in the ER diagram:



**ER schema** is not supposed to change frequently so the emphasis in ER diagram**ER diagram** is on types rather than instances.

Click here for the ER diagram notation table

**3.15. Discuss the naming conventions used for ER schema diagrams.**

**Ans**

Conventions for proper naming of database objects in ER diagram involve capitalization**capitalization**, deciding word types**deciding word types** and positioning**positioning**. Although, every database schema designer has freedom of choices, these conventions are proven the best to be followed:

∙∙ Entity type is named with a singular name*singular name* which represents its individuality. Examples: BOOK, DRAGON, TREE, \ldots

∙∙ Entity and relationship types are named in uppercase letters. Examples: WORKER, COMPANY, WORKS\_FOR, WORKS\_ON, \ldots

∙∙ Attributes are named with only the first letter capitalized. Examples: Author, Number of pages, Illustrator, Publication year, \ldots

∙∙ Role names are named in lowercase letter. Examples: supervisor, supervisee, lender, \ldots

∙∙ Important nouns appearing in the database description tend to become entity types.

∙∙ Nouns describing important nouns that appear in the database description are usually turned into attributes.

∙∙ Verbs appearing in the database description indicate relationship type names.

∙∙ For readability of the diagram, we name the relationship types to be readable from left to right, or from top to bottom. For example, entity types BOOK and CUSTOMER in the library database should be related with the relationship of borrowing a book. Depending on the positioning of BOOK and CUSTOMER boxes on the diagram, relationship type could be names IS\_BORROWED\_BY or HAS\_BORROWED.

Conventions for proper naming of database objects in ER diagram involve capitalization**capitalization**, deciding word types**deciding word types** and positioning**positioning**. Although, every database schema designer has freedom of choices, these conventions are proven the best to be followed: \ldots

**Exercises**

**3.16. Which combinations of attributes have to be unique for each individual**

**SECTION entity in the UNIVERSITY database shown in Figure 3.20 to enforce**

**each of the following miniworld constraints:**

**a. During a particular semester and year, only one section can use a particu**

**lar classroom at a particular DaysTime value.**

**Ans**

Each subtask refers to a particular semester and year, so Sem and Year attributes are included in each unique combination of attributes.

a. (Sem, Year, SecId, CRoom, DaysTime)**a. (Sem, Year, SecId, CRoom, DaysTime)**

Section uniqueness is represented by including SecId attribute in combination, unique classroom is represented by including CRoom composite attribute and DaysTime attribute is included because of subtask requirement.

b. (Sem, Year, SecId, DaysTime)**b. (Sem, Year, SecId, DaysTime)**

It is sufficient just to include SecId and DaysTime. Id attribute of the INSTRUCTOR entity type is not necessary here because relationship type TEACHES states that a section must be taught by a single instructor.

c. (Sem, Year, SecId)**c. (Sem, Year, SecId)**

This is sufficient since section numbers are different for multiple sections during the same semester and year and the SECTION entity type is related to the COURSE entity type by the SECS relationship type of 1:1 cardinality.

Similar constraints examples:

∙∙ Example 1:**Example 1:** Every instructor in a single department has to have a unique name.

Unique attribute combination: (DName(or DCode), IName)**(DName(or DCode), IName)**

∙∙ Example 2:**Example 2:** Students belonging to the same department must have different addresses and phone numbers.

Unique attribute combinations: (DName(or DCode), Addr) and (DName(or DCode), Phone)**(DName(or DCode), Addr) and (DName(or DCode), Phone)**

**a. (Sem, Year, SecId, CRoom, DaysTime)**

b. (Sem, Year, SecId, DaysTime)**b. (Sem, Year, SecId, DaysTime)**

c. (Sem, Year, SecId)**c. (Sem, Year, SecId)**

∙∙ Example 1:**Example 1:** Every instructor in a single department has to \ldots

∙∙ Example 2:**Example 2:** Students belonging to the same department must \ldots

**b. During a particular semester and year, an instructor can teach only one**

**section at a particular DaysTime value.**

**Ans**

For each SECTIONSECTION **entity** in the UNIVERSITYUNIVERSITY database from the given figure, we are asked to identify what unique **attribute** combinations are required to ensure that an instructor teaches only one section at a specific DaysTimeDaysTime during a given semester and year?

Let's get started!

Let us discuss some key concepts.

**Entity-Relationship** (ER) Model: This is a conceptual tool for modeling data in terms of entities and relationships in a specific domain.

**Entity**: In an ER diagram, an entity represents a real-world object or concept with an independent existence, such as an instructor or a course section in the university context.

**Attributes**: These are properties or details that define or describe an entity.

Let's discuss the indicated figure.

The image is a diagram representing an **Entity-Relationship** (ER) model for a university database schema. It includes various entities such as COLLEGECOLLEGE, DEANDEAN, INSTRUCTORINSTRUCTOR, STUDENTSTUDENT, DEPARTMENT (DEPT)DEPARTMENT (DEPT), COURSECOURSE, and SECTIONSECTION, each with their respective attributes.

For instance, the COLLEGECOLLEGE entity has attributes like CNameCName (College Name) and CPhoneCPhone (College Phone).

The relationships between these entities are also depicted, with cardinality ratios indicating the nature of the relationships (one-to-one, one-to-many, etc.).

In the ER diagram, a SECTIONSECTION entity is uniquely identified by its attributes SectIdSectId (Section ID), DaysTimeDaysTime (the days and times the section meets), YearYear, and SemSem (Semester).

To enforce the miniworld constraint that an instructor can teach only one section at a particular DaysTimeDaysTime value during a specific semester and year, the following combination of attributes would need to be unique for each SECTIONSECTION entity:

* Instructor ID (which would be a foreign key from the INSTRUCTOR entity)
* DaysTimeDaysTime
* YearYear
* SemSem
* This unique combination would ensure that no instructor is assigned to more than one section at the same DaysTimeDaysTime during the same semester and year.
* It's important to note that the actual attribute names corresponding to the instructor's ID may differ based on the schema. The SectIdSectId alone wouldn't be sufficient for this constraint because it only uniquely identifies a section, not the instructor's assignment to a section at a specific time.
* In this exercise, we're examining an **Entity-Relationship** (ER) model of a university database to determine the unique **attribute** combination needed to ensure an instructor teaches only one section at a specific time during a semester and year.
* In the ER diagram, **entities** like SECTIONSECTION have attributes such as Section ID and DaysTimeDaysTime. To meet the constraint of unique teaching assignments, a combination of Instructor ID, DaysTimeDaysTime, YearYear, and SemesterSemester must be unique for each SECTIONSECTION entity.
* This combination prevents an instructor from being assigned to multiple sections at the same time.

**c. During a particular semester and year, the section numbers for sections**

**offered for the same course must all be different.**

**Can you think of any other similar constraints?**

**Ans**

In this exercise, we are asked to determine what unique **attribute** combinations must each SECTIONSECTION **entity** in the UNIVERSITYUNIVERSITY database (indicated figure) have to ensure that section numbers are different for the same course in a specific semester and year?

Let's get started!

Let us discuss some key concepts.

**Entity-Relationship** (ER) Model: This is a conceptual tool for modeling data in terms of entities and relationships in a specific domain.

**Entity**: In an ER diagram, an entity represents a real-world object or concept with an independent existence, such as an instructor or a course section in the university context.

**Attributes**: These are properties or details that define or describe an entity.

To enforce the miniworld constraint that section numbers for sections offered for the same course must be different during a particular semester and year, the following combination of attributes would need to be unique for each SECTIONSECTION entity:

* Course Code (CCodeCCode) - This attribute would link the section to a specific course.
* Section Number (SecNoSecNo) - This would be the specific section number for the course.
* Semester (SemSem) - This indicates the academic term when the course is offered. Year - This represents the calendar year in which the course is offered.
* The unique combination of these attributes (Course Code, Section Number, Semester, and Year) ensures that no two sections of the same course can have the same section number in the same semester and year.
* This combination effectively creates a unique identifier for each course offering in a given time frame, preventing any overlap in section numbers for the same course.
* In this exercise, we focus on identifying the unique **attribute** combination needed in a university database to ensure that each course section has a distinct number for the same course in a specific semester and year.
* To meet this requirement, each SECTIONSECTION **entity** must have a unique combination of Course Code, Section Number, Semester, and Year.
* This combination ensures that section numbers do not overlap for the same course within the same semester and year, thereby creating a unique identifier for each course offering in a given timeframe

**3.17. Composite and multivalued attributes can be nested to any number of lev**

**els. 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 com**

**posed of college name, start and end dates, degree entries (degrees awarded**

**at that college, if any), and transcript entries (courses completed at that col**

**lege, 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 informa**

**tion. Use the conventions in Figure 3.5.**

**Ans**

We need to create a complex*complex* attribute of the STUDENT entity type. Let's call that attribute "Attended\_college"**"Attended\_college"**.

Firstly, student may have attended more than one college, so that fact makes this attribute multivalued*multivalued*. Multivalued attributes are put between braces. {Attended\_college}**{Attended\_college}**

Secondly, we know that each attended college has to have additional attributes "College\_name", "Start\_date", "End\_date", "Degree" and "Completed\_course". We can display composite*composite* attribute by opening parentheses and listing all the attributes.

{Attended\_college(College\_name, Start\_date, End\_date, Degree, Completed\_course}**{Attended\_college(College\_name, Start\_date, End\_date, Degree, Completed\_course}**

Both attributes, Degree and Completed\_course, are composite*composite* and multivalued*multivalued*. Degree attribute has additional attributes "DName", "DYear", "DMonth", and Completed\_course attribute contains attributes "CName", "CSemester", "CYear" and "CGrade".

Lastly, the Attended\_college attribute should look like this:

{Attended\_college(College\_name, Start\_date, End\_date, {Degree(DNname, DYear, DMonth)}, {Completed\_course(CName, CSemester, CYear, CGrade)}}**{Attended\_college(College\_name, Start\_date, End\_date, {Degree(DNname, DYear, DMonth)}, {Completed\_course(CName, CSemester, CYear, CGrade)}}**

The "Attended\_college"**"Attended\_college"** attribute should look like this:

{Attended\_college(College\_name, Start\_date, End\_date, {Degree(DNname, DYear, DMonth)}, {Completed\_course(CName, CSemester, CYear, CGrade)}}**{Attended\_college(College\_name, Start\_date, End\_date, {Degree(DNname, DYear, DMonth)}, {Completed\_course(CName, CSemester, CYear, CGrade)}}**

**3.18. Show an alternative design for the attribute described in Exercise 3.17 that**

**uses only entity types (including weak entity types, if needed) and relation**

**ship types.**

**Ans**

This is an alternative design for the complex attribute in the previous exercise. Multivalued*Multivalued* and composite*composite* attributes are replaced with entity and relationship types. More detailed description of the given diagram follows.

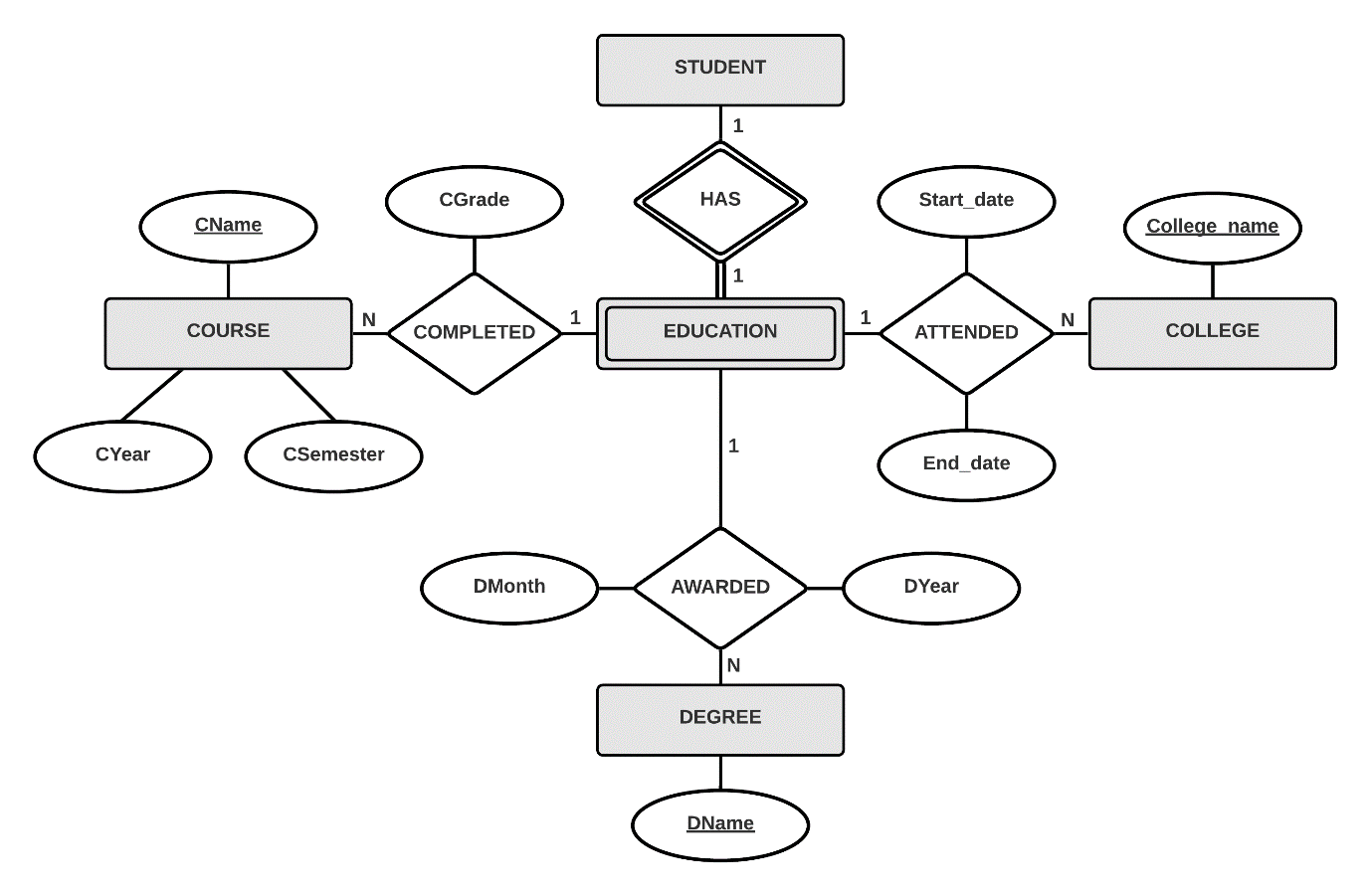
∙∙ STUDENT entity type is related to the weak entity type EDUCATION by identifying relationship type HAS. Every student has exactly one education history, so the cardinality is set to 1:1.

∙∙ EDUCATION is related to academic content*academic content* entity types COURSE, DEGREE and COLLEGE. All the relationship types connecting EDUCATION to academic content*academic content* are of cardinality 1:N because a single EDUCATION may include attendances of many colleges, many awarded degrees and many completed courses.

∙∙ ATTENDED relationship type has attributes that represent start and end date of attending particular college and COLLEGE entity type has a key attribute College\_name.

∙∙ When a DEGREE is awarded to someone's EDUCATION, attributes representing month and year of getting a degree are recorded as attributes of relationship type AWARDED. DEGREE entity type has key attribute DName.

∙∙ During one's education, many courses are completed. Every COURSE entity type has a unique key attribute CName and year and semester attributes. When a COURSE is COMPLETED, certain grade is recorded in someone's education.



This is an alternative design for the complex attribute in the previous exercise. Multivalued*Multivalued* and composite*composite* attributes are replaced with entity and relationship types.

Click for the ER diagram and its description**ER diagram and its description** \ldots

**3.19. Consider the ER diagram in Figure 3.21, which shows a simplified schema**

**for an airline reservations system. Extract from the ER diagram the require**

**ments and constraints that produced this schema. Try to be as precise as**

**possible in your requirements and constraints specification**.

**Ans**

The ER diagram represents an airport and its ticket reservation system. The requirements for this database are as follows:

∙∙ AIRPORT has a Name, a unique Airport\_code and a location represented by City and State.

∙∙ AIRPORT CAN\_LAND many different AIRPLANE\_TYPEs. Each AIRPLANE\_TYPE has a unique Type\_name, Company it belongs to and a number of maximum seats, Max\_seats. AIRPLANEs belong to a specific AIRPLANE\_TYPE and are distinguished by Airplane\_id. They also have more precise information about seats, Total\_number\_of\_seats.

∙∙ FLIGHT is unique by its Number, and has information to which Airline it belongs as well as on which Weekdays it is scheduled. Every flight also has its FARE which has its Amount and Restrictions. FARE also has a Code. A single FLIGHT can have more than one FLIGHT\_LEG. FLIGHT\_LEG is defined with its number, Leg\_no, and AIRPORTs that play the roles of DEPARTURE\_AIRPORT and ARRIVAL\_AIRPORT with time being recorded as Scheduled\_dep\_time and Scheduled\_arr\_time.

∙∙ FLIGHT\_LEG on a particular Date is recorded as LEG\_INSTANCE. LEG\_INSTANCE also holds the information about the number of available seats, No\_of\_avail\_seats. One or more LEG\_INSTANCEs must be assigned to an AIRPLANE. LEG\_INSTANCE ARRIVES and DEPARTS at certain times, noted Arr\_time and Dep\_time, to and from particular AIRPORT.

∙∙ Finally, customer can reserve a particular seat by the seat number, Seat\_no, and while reserving a seat, customer's name and phone number are recorded as Customer\_name and Cphone.

The ER diagram represents an airport and its ticket reservation system. The requirements for this database are as follows:

∙∙ AIRPORT has a Name, a unique Airport\_code and a location \ldots

∙∙ AIRPORT CAN\_LAND many different AIRPLANE\_TYPEs. Each \ldots

∙∙ FLIGHT is unique by its Number, and has information to which \ldots \ldots

**3.20. In Chapters 1 and 2, we discussed the database environment and database**

**users. We can consider many entity types to describe such an environment,**

**such as DBMS, stored database, DBA, and catalog/data dictionary. Try to**

**specify all the entity types that can fully describe a database system and its**

**environment; then specify the relationship types among them, and draw an**

**ER diagram to describe such a general database environment.**

**Ans**

This is a simple specification of entity and relationship types that form the database environment**database environment**. Security, maintaining and different types of users are omitted in the following presentation because of simplicity.

**ENTITY TYPES:**

∙∙ DBMS represents database management system and is separated into many modules. Every DBMS has to be administrated by one or more database administrators.

∙∙ MODULES entity type represents set of modules which work together in query processing, data accessing and creating database catalog (there are usually many more modules...).

∙∙ QUERY PROCESSING is a module responsible for processing canned transactions. Canned transactions are represented as TRANSACTION entity type. END\_USER is the entity type representing end users that work with transactions. Before working with transactions, they have to be granted access to the database by the database manager.

∙∙ DATA\_ACCESSING is another module that is used to access the database. To access the database, it uses database CATALOG, designed by the module CATALOG\_DESIGN. The database it accesses is represented as entity type STORED\_DATABASE, which is STORED on STORAGE\_MEDIUMs. STORED\_DATABASE has its database SCHEMA which is stored in the CATALOG by the database administrator.

∙∙ The database administrator is represented by the DBA entity type. It is responsible for granting access to END\_USER, administering DMBS and storing the database SCHEMA into database CATALOG.

**RELATIONSHIP TYPES:**

∙∙ MADE\_OF relationship type is between DBMS and MODULES. It has cardinality of 1:1 because entity type MODULES represents a set of modules.

∙∙ IS\_TYPE\_OF is a relationship type between entity types that represent modules, and MODULES entity type. Set of modules contains only one of each given module types so the cardinality is 1:1.

∙∙ PROCESSES relationship type relates the QUERY\_PROCESSING and TRANSACTION entity types. One module can process many transactions. Cardinality is 1:N.

∙∙ END\_USER and TRANSACTION entity types are related by relationship type WORKS\_WITH. It is of cardinality M:N because more than one user can work with a single transaction, and a single user can work with many transactions.

∙∙ END\_USER GETS\_ACCESS from the DBA. This relationship is M:N because END\_USER can get required accesses from more than one DBA and a DBA can grant access to more than one END\_USER.

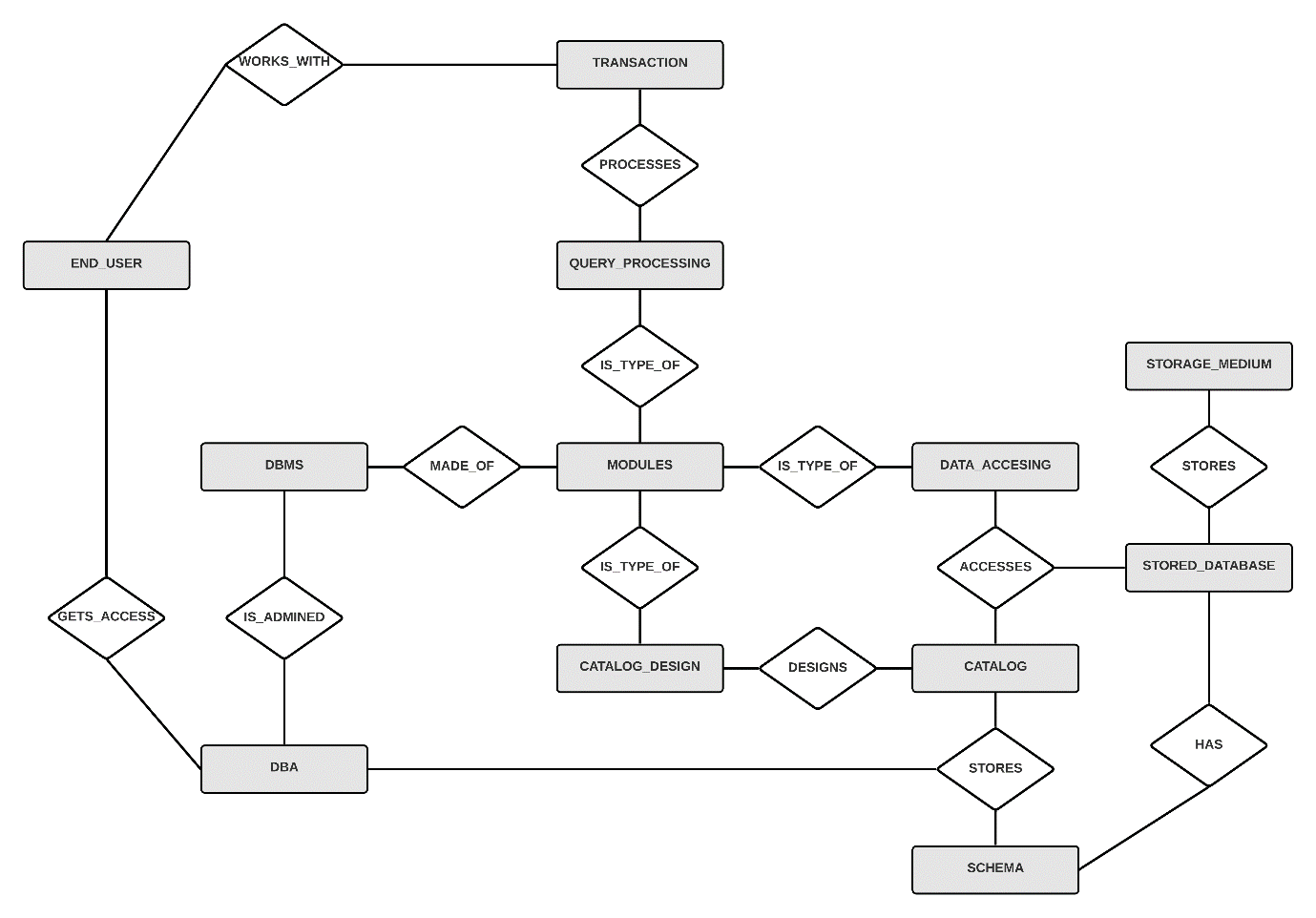
∙∙ IS\_ADMINED relationship type relates DBMS and DBA entity types. Cardinality of this relationship is 1:N because a DBMS can be admined by many database administrators and database administrator can administrate a single DBMS at time.

∙∙ STORES is a relationship types that relates a DBA, a SCHEMA and a CATALOG.

∙∙ DATA\_ACCESSING module is related to the STORED\_DATABASE and CATALOG by relationship type ACCESSES. Before searching through the database, catalog is used for more efficient search.

∙∙ HAS relationship type relates STORED\_DATABASE and SCHEMA entity types. Schema can change (usually rarely) so a single STORED\_DATABASE can have more schemas. Cardinality is 1:N.

∙∙ STORES relationship type denotes to which STORAGE\_MEDIUM the database is stored to. This relationship type is of cardinality M:N because a single STORED\_DATABASE can be stored on many STORAGE\_MEDIUMs and a STORAGE\_MEDIUM can have more than one STORED\_DATABASE in it.



This is a simple specification of entity and relationship types that form the database environment**database environment**. Security, maintaining and different types of users are omitted in the following presentation because of simplicity.

Click for the detailed description**detailed description** and {ER diagram}.

**3.21. Design an ER schema for keeping track of information about votes taken in**

**the U.S. House of Representatives during the current two-year congress**

**ional session. The database needs to keep track of each U.S. STATE’s Name**

**(e.g., ‘Texas’, ‘New York’, ‘California’) and include the Region of the state**

**(whose domain is {‘Northeast’, ‘Midwest’, ‘Southeast’, ‘Southwest’, ‘West’}).**

**Each CONGRESS\_PERSON in the House of Representatives is described by**

**his or her Name, plus the District represented, the Start\_date when the con**

**gressperson was first elected, and the political Party to which he or she**

**belongs (whose domain is {‘Republican’, ‘Democrat’, ‘Independent’,**

**‘Other’}). The database keeps track of each BILL (i.e., proposed law),**

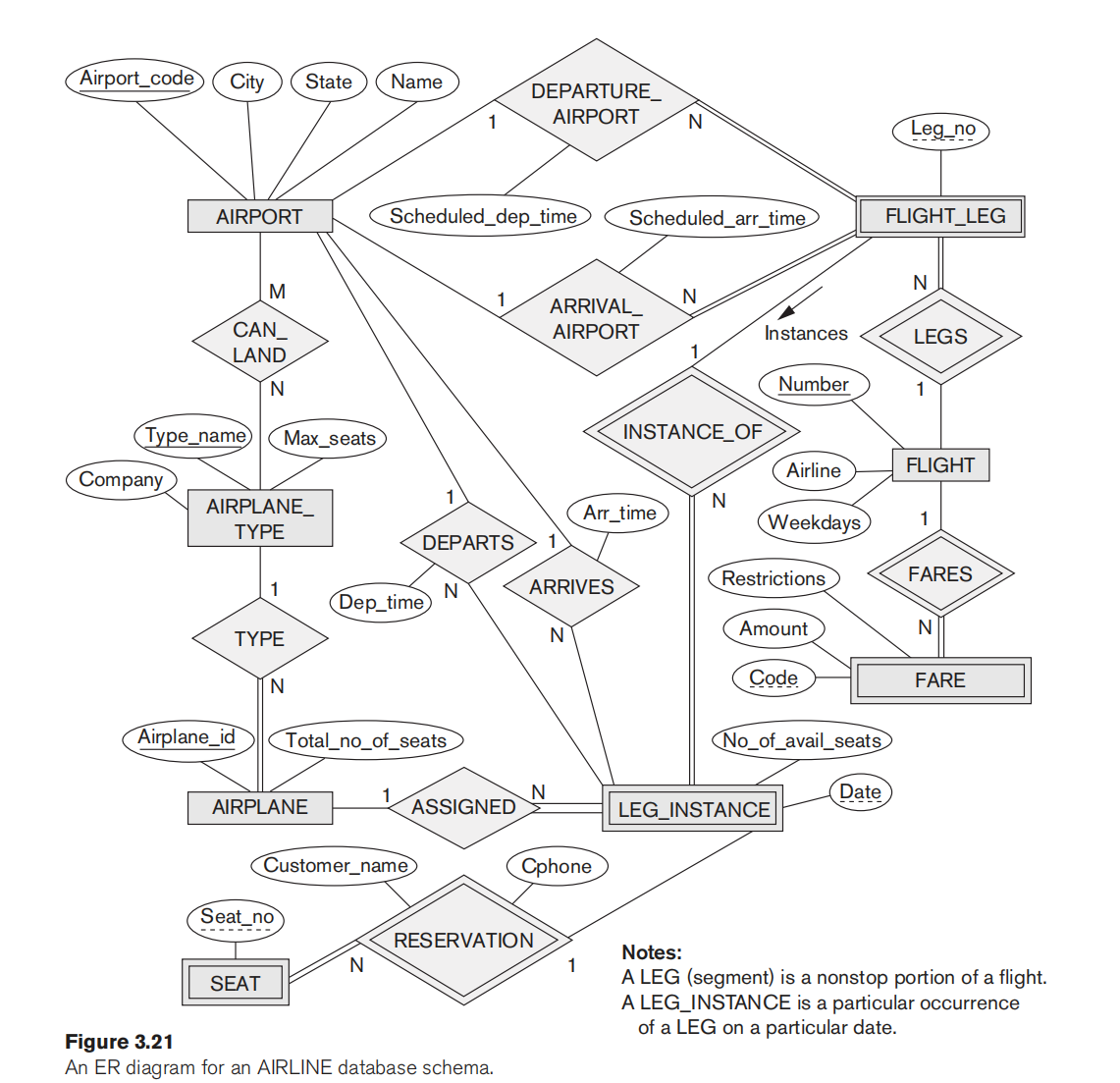
**including the Bill\_name, the Date\_of\_vote on the bill, whether the bill**

**Passed\_or\_failed (whose domain is {‘Yes’, ‘No’}), and the Sponsor (the**

**congressperson(s) who sponsored—that is, proposed—the bill). The data**

**base also keeps track of how each congressperson voted on each bill (domain of Vote attribute is {‘Yes’, ‘No’, ‘Abstain’, ‘Absent’}). Draw an ER schema**

**diagram for this application. State clearly any assumptions you make.**



**Ans**

This is the ER schema**the ER schema** and the ER diagram**the ER diagram** for recording and keeping track of votes on proposed bills in the U.S. House of Representatives.

While reading through the exercise, we can single out three entity types**three entity types** and three relationship types**three relationship types**

**ENTITY TYPES:**

∙∙ CONGRESS\_PERSON**CONGRESS\_PERSON** represents state representatives. Key attribute of this entity type is CName*CName* which represents the congressman's name (let's think that no two congressmen can have identical full name). It also has attributes District*District* (which congressman represents), Start\_date*Start\_date* (date when elected) and Party*Party* (one of Republican, Democrat, Independent or Other).

∙∙ BILL**BILL** represents proposed law. It is unique by its name BName*BName*, and has two additional attributes Date\_of\_vote*Date\_of\_vote* (the date on which the bill was voted) and Passed\_or\_failed*Passed\_or\_failed* ("Yes" value if the bill passed and "No" if the bill failed).

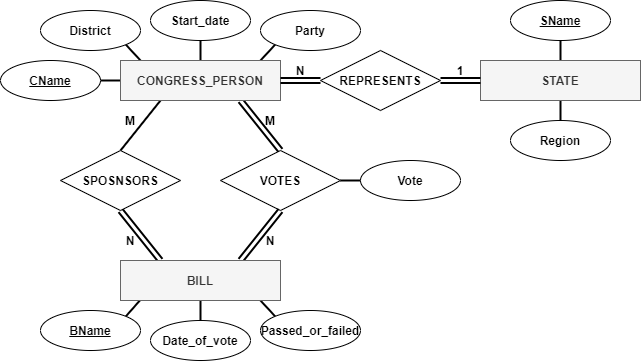
∙∙ STATE**STATE** represents one U.S.A. state. Every state's name is unique so the attribute SName*SName* plays the role of key attribute. Beside state's name, there is also an attribute Region*Region* which can have one of five regions as its value ("Northeast", "Midwest", "Southeast", "Southwest" or "West").

**ENTITY TYPES:**

∙∙ CONGRESS\_PERSON**CONGRESS\_PERSON** represents state representatives. Key attribute of this entity type is CName*CName* which represents the congressman's name (let's think that no two congressmen can have identical full name). It also has attributes District*District* (which congressman represents), Start\_date*Start\_date* (date when elected) and Party*Party* (one of Republican, Democrat, Independent or Other).

∙∙ BILL**BILL** represents proposed law. It is unique by its name BName*BName*, and has two additional attributes Date\_of\_vote*Date\_of\_vote* (the date on which the bill was voted) and Passed\_or\_failed*Passed\_or\_failed* ("Yes" value if the bill passed and "No" if the bill failed).

∙∙ STATE**STATE** represents one U.S.A. state. Every state's name is unique so the attribute SName*SName* plays the role of key attribute. Beside state's name, there is also an attribute Region*Region* which can have one of five regions as its value ("Northeast", "Midwest", "Southeast", "Southwest" or "West").



This is the ER schema**the ER schema** and the ER diagram**the ER diagram** for recording and keeping track of votes on proposed bills in the U.S. House of Representatives. While reading through the exercise, we can single out three entity types**three entity types** and three relationship types**three relationship types**.

Click for the ER schema**ER schema** and the ER schema diagram**ER schema diagram** \ldots

**3.22. 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).**

**Ans**

This is the ER schema**ER schema** and the ER schema diagram**ER schema diagram** for the database of National Basketball Association, better known as the NBA. Of course, very simplified.

During the regular part of season of the NBA league, each team plays 82 games. Half of those games is played at home court, as "home team", and the other half is played at the home courts of other teams, as "away team". There are 12 players in each team. Every player has its usual playing position but can play any other position in the game, if needed.

Game scores and players participations are recorded after each game.

Here are entity and relationship types descriptions:

**ENTITY TYPES:**

∙∙ TEAM**TEAM** entity type has a key attribute TName*TName* that represent team's name. It also has an attribute Home\_court*Home\_court* that represents the location of the team's home court.

∙∙ PLAYER**PLAYER** entity type represents players belonging to the teams of NBA league. Players are distinguished by their Ssn*Ssn* (unique social security number) and have attributes PName*PName* (player's full name) and Usual\_position*Usual\_position* (position they are best at) with possible values "Center", "Power forward", "Small forward", "Point guard", and "Shooting guard".

∙∙ GAME**GAME** entity type has a key composed of three other attributes. The key is named Time\_place*Time\_place* and is a composite attribute composed of attributes Date\_time*Date\_time* (date and hour of the game), Address*Address* (of the place with courts) and Court\_no*Court\_no* (identification of the court on particular address).

**Step 3**

3 of 5

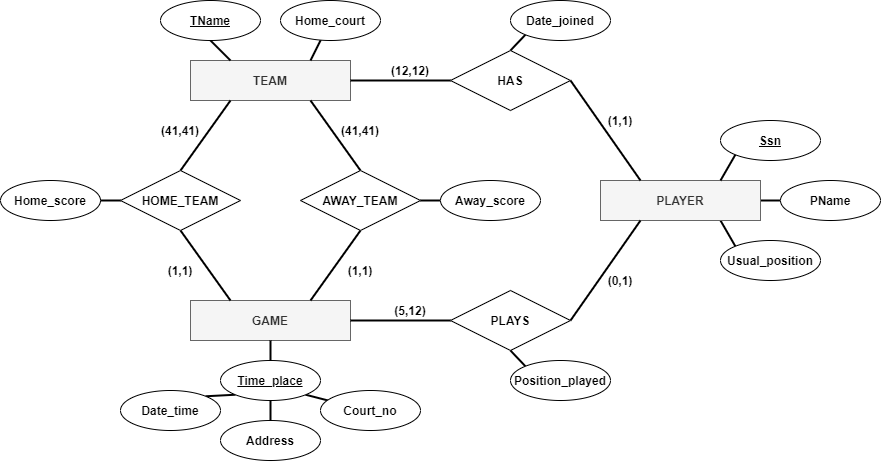
RELATIONSHIP TYPES:**RELATIONSHIP TYPES:** (using (min,max) cardinality notation)

∙∙ HAS**HAS** is a relationship type which relates TEAM and PLAYER entity types. It has an attribute Date\_joined which represents the date when the player signed a contract with some team. A team has to have a total of 12 players and a player belongs to only one team.

∙∙ PLAYS**PLAYS** relationship type relates PLAYER and GAME entity types. PLAYS has an attribute Position\_played*Position\_played* that records a position particular player plays in a game. In each game at least 5 players from each team must play a game, other 7 players are reserves and they can, but don't have to play the game.

∙∙ HOME\_TEAM**HOME\_TEAM** (full name would be "PLAYS\_AS\_HOME\_TEAM", but that wouldn't fit in the relationship box) relationship type represents team's participation in a game as the home team. Score is recorded as the value of the Home\_score*Home\_score* attribute. In every game there is one home team and a team plays as a home team in 41 games of the regular part of the NBA season.

∙∙ AWAY\_TEAM**AWAY\_TEAM** is pretty much the same as the HOME\_TEAM relationship type. The only difference is that the relationship's attribute is now called Away\_score*Away\_score*.



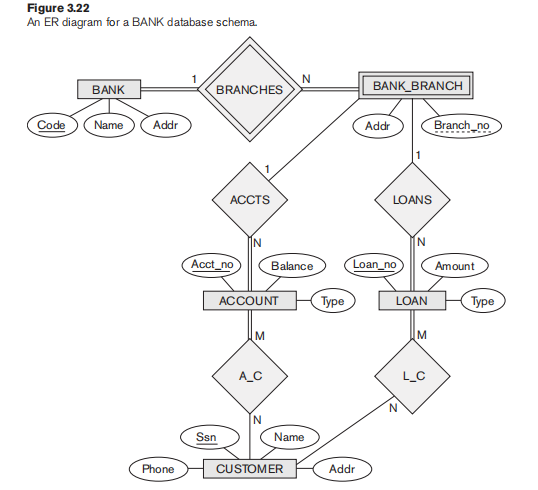
This is the ER schema**ER schema** and the ER diagram**ER diagram** for the database of National Basketball Association, better known as the NBA. Of course, very simplified.

Click for the ER schema**ER schema** and the ER schema diagram**ER schema diagram** \ldots

**3.23. Consider the ER diagram shown in Figure 3.22 for part of a BANK database.**

**Each bank can have multiple branches, and each branch can have multiple**

**accounts and loans.**



1. **List the strong (nonwe000ak) entity types in the ER diagram.**

**Ans**

Strong entity types in the given BANK database are **BANK** (key attribute: Code), **ACCOUNT** (key attribute: Acct\_no), **LOAN** (key attribute: Loan\_no), and **CUSTOMER** (key attribute: Ssn).

**b. Is there a weak entity type? If so, give its name, partial key, and identify**

**ing relationship.**

**Ans**

There is a weak entity type **BANK\_BRANCH** which has a partial key **Branch\_no** and an identifying relationship **BRANCHES**.

**c. What constraints do the partial key and the identifying relationship of the**

**weak entity type specify in this diagram?**

**Ans**

The partial key **Branch\_no** of the entity type BANK\_BRANCH puts a constraint on BANK entities so that **every branch of a bank has to have a different branch number**. The identifying relationship type BRANCHES puts a **total participation constraint** on both related entity types. This means that a bank cannot exist without bank branches and a bank branch cannot exist without a bank. There is also a **cardinality ratio constraint** which states that a single bank can have many bank branches and a bank branch can belong to only one bank. Cardinality ratio is one to many **(1:N)**.

**d. List the names of all relationship types, and specify the (min, max)**

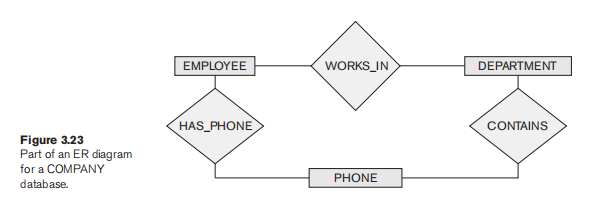
**constraint on each participation of an entity type in a relationship type.**

**Justify your choices**

**Ans**

The list of all relationship types and their (min, max) constraints:

* **BRANCHES** between BANK and BANK\_BRANCH. BANK entity type has a (min, max) constraint of **(1,N)**. Minimal constraint is 1 because it is connected to BRANCHES with double line (representing total participation) and N because a maximum number of branches is not defined. Entity type BANK\_BRANCH has a (min, max) constraint of **(1,1)** because a branch cannot exist without a bank, and a single branch can belong to only one bank.
* **ACCTS** relationship type relates BANK\_BRANCH and ACCOUNT entity types. BANK\_BRANCH has a (min, max) constraint of **(0,N)** because it can have many but doesn't have to have any accounts for customers. ACCOUNT has a (min, max) constraint of **(1,1)** because if the account exists, it must belong to one and only one BANK\_BRANCH.
* **LOANS** relationship type relates BANK\_BRANCH and LOAN entity types. Similarly to ACCTS relationship type, BANK\_BRANCH entity type has a (min, max) constraint of **(0,N)** because it can but doesn't have to give loans, and LOAN has (min, max) constraint of **(1,1)** because if the loan exists, it had to be given by one and only one bank branch.
* **A\_C** is the relationship type between ACCOUNT and CUSTOMER. Account cannot exist without being assigned to a customer, and a single account can be shared between many customers so the (min, max) cardinality of ACCOUNT's participation is **(1,N)**. Customer can have many accounts, but doesn't have to have any. The (min, max) cardinality of CUSTOMER's participation is **(0,N)**.
* **L\_C** relates LOAN and CUSTOMER in a similar way to A\_C. Loan has to belong to a customer and a single loan can be shared between many customers. Also, a customer doesn't have to have any, but can have many loans. So the (min, max) constraint of the CUSTOMER's participation is **(0,N)** and of the LOAN's participation is **(1,N)**.



**e. List concisely the user requirements that led to this ER schema design.**

**Ans**

These are the database requirements for the BANK database:

* Database should keep track of **banks**, **bank branches**, **accounts**, **loans**, and **customers**. Each bank has a *name*, *an address*, and a unique *code*.
* Every bank has one or more branches. A branch can belong to a single bank, can have *address* different than the bank address, and has a *branch number* that is unique for the bank it belongs to.
* Bank branches are in charge of giving accounts and loans to customers. Every bank branch can give many accounts and loans. Accounts and loans have their *types* and unique *numbers*. Accounts also store information about account *balance* and loans store information about the *amount* on a loan. Accounts and loans must be given from some bank branch to a customer; otherwise, they can't exist.
* Customers are identified by their *Ssn* (social security number). Additional information about the customer is stored in the database and includes their *phone number*, *name*, and *address*. Customers can have more than one account and loan, and more than one customer can share an account or a loan.

**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?**

**Ans**

Customer's participation in the A\_C relationship type would be noted as **(1,1)**, and in the L\_C relationship type as **(0,2)**. Bank's participation in the LOANS relationship type would be noted as **(0,1000)**.

**3.24. Consider the ER diagram in Figure 3.23. Assume that an employee may**

**work in up to two departments or may not be assigned to any department.**

**Assume that each department must have one and may have up to three**

**phone numbers. Supply (min, max) constraints on this diagram. *State clearly***

***any additional assumptions you make.* Under what conditions would the**

**relationship HAS\_PHONE be redundant in this example?**

**Ans**

From the task requirements, we can determine two (min, max) constraints.

First one is the (0,2)**(0,2)** constraint on participation of EMPLOYEE entity type in WORKS\_IN relationship type and the second one is the (1,3)**(1,3)** constraint on participation of DEPARTMENT entity type in CONTAINS relationship type.

Other requirements are not specified so we have to think of additional assumptions ourselves.

∙∙ Let's start with assuming that each department must have at least 5 employees and can have up to 400 employees. This puts a (5,400)**(5,400)** constraint on DEPARTMENT's participation in WORKS\_IN relationship type.

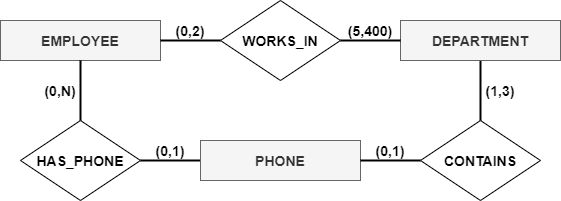
∙∙ The department contains at least one and up to 3 phones. It would be reasonably to assume that a single phone can belong to only one department. And, since the PHONE entity type is related to EMPLOYEE and can also represent their phones, the phone doesn't have to belong to a department at all. So the constraint on PHONE's participation in the CONTAINS relationship type is (0,1)**(0,1)**.

∙∙ EMPLOYEE doesn't have to have a phone, but can also have more than one phone. Constraint on EMPLOYEE's participation in HAS\_PHONE relationship type is (0,N)**(0,N)**.

∙∙ Lastly, a phone can belong to a single employee, but also doesn't have to belong to any employee (if it's the department's phone only) so the constraint on PHONE's participation in the HAS\_PHONE relationship type is (0,1)**(0,1)**.

HAS\_PHONE relationship type would be redundant if we had a constraint that all the phones in database belong to departments and employees use only the department's phones. In that case, there would be a ternary relationship type CONTAINS (or some other name) which would relate entity types EMPLOYEE (using the phone), PHONE and DEPARTMENT (to which the phone belongs).

Other possible constraint would be that EMPLOYEE'S phones are stored as multivalued attribute of EMPLOYEE entity type. This way, the database would only store DEPARTMENT's phones as entity type.



From the task requirements, we can determine two (min, max) constraints. First one is the (0,2)**(0,2)** constraint on participation of EMPLOYEE \ldots

HAS\_PHONE relationship type would be redundant if we had a constraint that \ldots

Click for the detailed assumptions descriptions**detailed assumptions descriptions** and the supplied ER diagram**supplied ER diagram** \ldots

**3.25. Consider the ER diagram in Figure 3.24. Assume that a course may or may**

**not use a textbook, but that a text by definition is a book that is used in some**

**course. A course may not use more than five books. Instructors teach from**

**two to four courses. Supply (min, max) constraints on this diagram. *State***

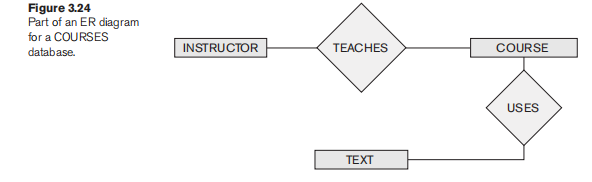
***clearly any additional assumptions you make.* If we add the relationship**

**ADOPTS, to indicate the textbook(s) that an instructor uses for a course,**

**should it be a binary relationship between INSTRUCTOR and TEXT, or a**

**ternary relationship among all three entity types? What (min, max) con**

**straints would you put on the relationship? Why?**



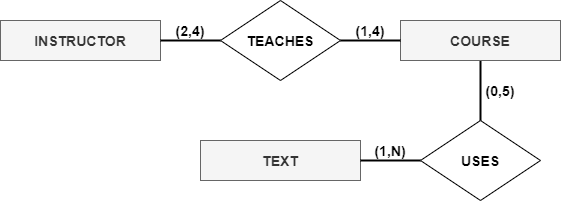
**Ans**

From the requirements of the task we can determine (min, max) constraints on COURSE's participation in USES relationship type to be (0,5)**(0,5)**, on INSTRUCTOR's participation in TEACHES relationship type to be (2,4)**(2,4)** and the integer representing the minimum in constraint on TEXT's participation in USES relationship type to be 1**1** because of text's definition in the exercise.

We are left with choices for the remaining participations.

∙∙ Let's say that a course can be taught by at least one instructor and by a maximum of 4 instructors (for different groups). Then the constraint on COURSE's participation in TEACHES relationship type could be noted as (1,4)**(1,4)**.

∙∙ Lastly, we have to decide the maximum number of courses in which a textbook can be used. At universities, some similar courses can use the same textbook, and since we are not informed with how big this database is, we can put N**N** as the maximum, meaning some indeterminate number.



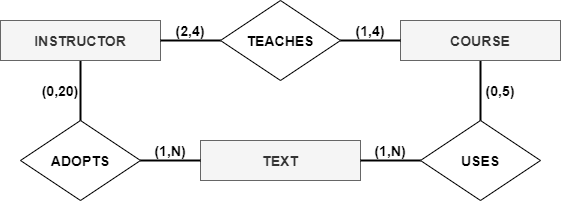
There is no need for the new ADOPTS relationship type to be a ternary relationship. It can store all the necessary information as a binary relationship.

To see which textbooks are used by particular instructor, it is sufficient to check the three relationships we have. Firstly we can check which textbooks are offered for the particular course we are interested in. Then, we check which instructors teach that particular course and pick one we were assigned to. And lastly, we check which textbooks that instructor adopts. If there is a congruence between adopted textbooks and textbooks used by a course for the instructor teaching that course, we got the wanted result for this query.

This may seem a bit complicated to search, but a simple query could be made so this would be a two click action. (more about this in next chapters)

And finally, (min, max) constraints for binary ADOPTS relationship type. Constraint on TEXT's participation is (1,N)**(1,N)** because textbook must be adopted by at least one instructor that teaches one course using that textbook. This is because of defintion of the text in this exercise. A single textbook can be adopted by many instructors, hence, the maximum is N.

Constraint on INSTRUCTOR's participation is (0,20)**(0,20)** because instructor can teach a course without a textbook since some courses do not require a textbook. Because an instructor can teach maximum of 4 courses, and a course can use a maximum of 5 textbooks, maximum of instructor's adopted textbooks is 4 times 5, or 20 textbooks.



From the requirements of the task we can determine (min, max) constraints on COURSE's participation in USES relationship type \ldots There is no need for the new ADOPTS relationship type to be a ternary relationship. It can store all the necessary information as a \ldots

Click to see supplied ER diagrams**supplied ER diagrams** and detailed descriptions**detailed descriptions** \ldots

**3.26. Consider an entity type SECTION in a UNIVERSITY database, which describes**

**the section offerings of courses. The attributes of SECTION are**

**Section\_number, Semester, Year, Course\_number, Instructor, Room\_no (where**

**section is taught), Building (where section is taught), Weekdays (domain is**

**the possible combinations of weekdays in which a section can be offered**

**{‘MWF’, ‘MW’, ‘TT’, and so on}), and Hours (domain is all possible**

**time periods during which sections are offered {‘9–9:50 a.m.’, ‘10–10:50**

**a.m.’, . . . , ‘3:30–4:50 p.m.’, ‘5:30–6:20 p.m.’, and so on}). Assume that**

**Section\_number is unique for each course within a particular semes**

**ter/year combination (that is, if a course is offered multiple times during**

**a particular semester, its section offerings are numbered 1, 2, 3, and so**

**on). There are several composite keys for section, and some attributes**

**are components of more than one key. Identify three composite keys,**

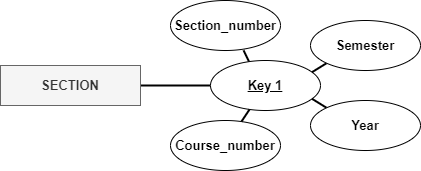
**and show how they can be represented in an ER schema diagram.**

**Ans**

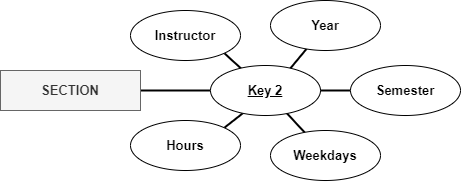
In this exercise there are 3 composite keys because of the property of composite key that states that a composite key must be minimal*minimal*. This means that no superfluous*superfluous* attribute can be a part of the composite key. Here are the 3 composite keys:

**Key 1:**

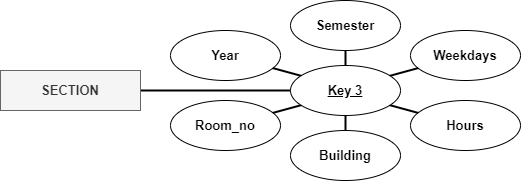
First composite key is explained in the exercise requirements. Section, represented by Section\_number**Section\_number**, is assumed to be unique in particular combination with Semester**Semester** and Year**Year** for particular course, represented by Course\_number**Course\_number**.



Second composite key is based on uniqueness of the Instructor**Instructor** attribute. The instructor can teach a single course in particular time. This unique time for a section is achieved by combining attributes representing time, Year**Year**, Semester**Semester**, Weekdays**Weekdays** and Hours**Hours**.



Third composite key is based on the unique combination of time and place attributes. At specific time and place, only one section can be taught. Attributes used in creating this unique composite key are Year**Year**, Semester**Semester**, Weekdays**Weekdays**, Hours**Hours**, Building**Building** and Room\_no**Room\_no**



**Key 1:** Section\_number**Section\_number**, Semester**Semester**, Year**Year** and Course\_number**Course\_number**

Key 2:**Key 2:** Instructor**Instructor**, Year**Year**, Semester**Semester**, Weekdays**Weekdays** and Hours**Hours**

Key 3:**Key 3:** Year**Year**, Semester**Semester**, Weekdays**Weekdays**, Hours**Hours**, Building**Building** and Room\_no**Room\_no**.

Click for the descriptions**descriptions** and the ER diagram representations**ER diagram representations** \ldots

**3.27. Cardinality ratios often dictate the detailed design of a database. The cardi**

**nality ratio depends on the real-world meaning of the entity types involved**

**and is defined by the specific application. For the following binary relation**

**ships, suggest cardinality ratios based on the common-sense meaning of the**

**entity types. Clearly state any assumptions you make.**

**Entity 1 Cardinality Ratio Entity 2**

**1. STUDENT \_\_\_\_\_\_\_\_\_\_\_\_\_\_ SOCIAL\_SECURITY\_CARD**

**2. STUDENT \_\_\_\_\_\_\_\_\_\_\_\_\_\_ TEACHER**

**3. CLASSROOM \_\_\_\_\_\_\_\_\_\_\_\_\_\_ WALL**

**4. COUNTRY \_\_\_\_\_\_\_\_\_\_\_\_\_\_ CURRENT\_PRESIDENT**

**5. COURSE \_\_\_\_\_\_\_\_\_\_\_\_\_\_ TEXTBOOK**

**6. ITEM (that can be found**

**in an order)**

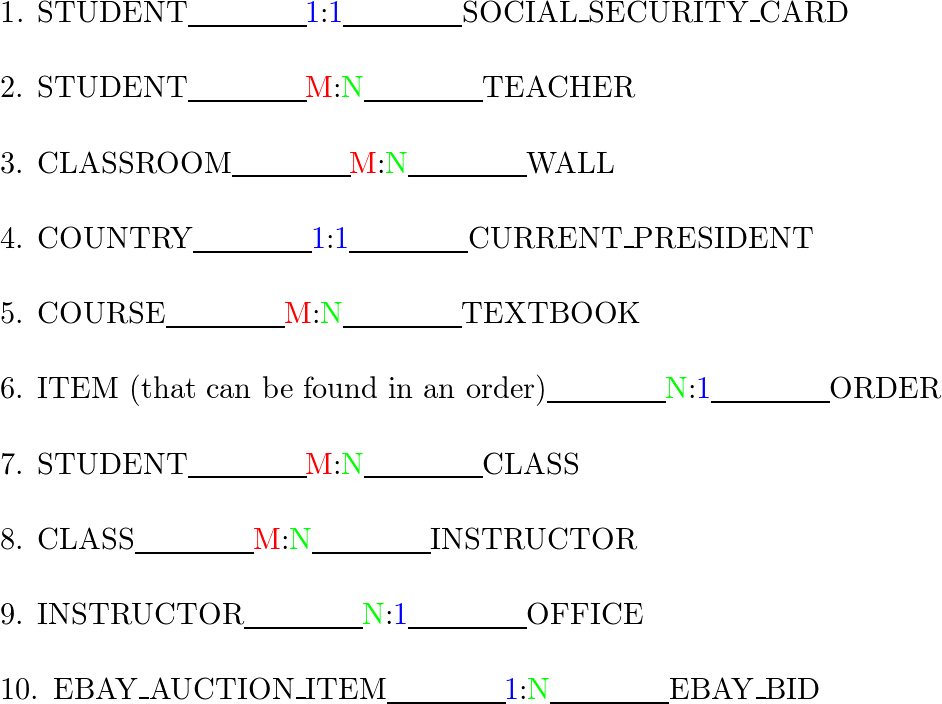
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ORDER**

**7. STUDENT \_\_\_\_\_\_\_\_\_\_\_\_\_\_ CLASS**

**8. CLASS \_\_\_\_\_\_\_\_\_\_\_\_\_\_ INSTRUCTOR**

**9. INSTRUCTOR \_\_\_\_\_\_\_\_\_\_\_\_\_\_ OFFICE**

**10. EBAY\_AUCTION\_ITEM \_\_\_\_\_\_\_\_\_\_\_\_\_\_ EBAY\_BID**



**3.28. Consider the ER schema for the MOVIES database in Figure 3.25.**

**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, although 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.**

**Ans**

**True -** The ACTOR entity type is related to PERFORMS\_IN relationship type by a double line, which represents total participation. Hence, the actor entity cannot exist in the database without being in some movie

**b. There are some actors who have acted in more than ten movies.**

**Ans**

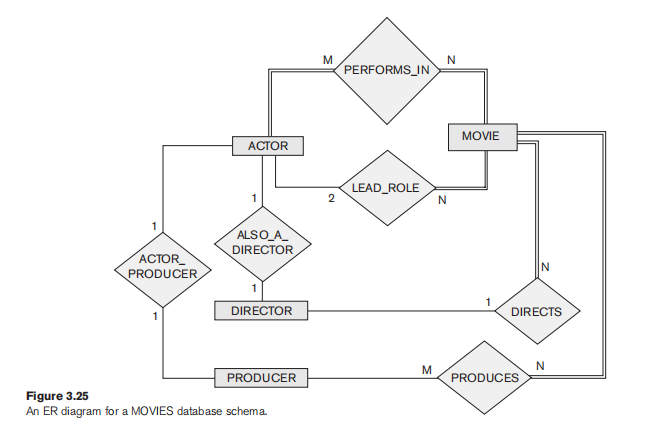
**Maybe -** The ACTOR entity type does not have an upper bound on participation in PERFORMS\_IN relationship type. However, we can't be sure if there is any actor who performed in more than 10 movies

**c. Some actors have done a lead role in multiple movies.**

**Ans**

**Maybe -** The ACTOR entity type does not have an upper bound on participation in LEAD\_ROLE relationship type. Without entity instances, we cannot know if there is any actor who had been a lead role in more than one movie

**d. A movie can have only a maximum of two lead actors.**



**Ans**

**True -** The number 2 beneath the line connecting the ACTOR entity type to the LEAD\_ROLE relationship type symbolizes the maximum number of lead actors in a single movie

**e. Every director has been an actor in some movie.**

**Ans**

**Maybe -** This statement may be true or false depending on entity instances that are not given for this exercise. Director's participation in ALSO\_A\_DIRECTOR relationship type is not mandatory, but that doesn't mean that every director in this database hasn't been an actor in some movie.

**f. No producer has ever been an actor.**

**Ans**

**Maybe -** The producer can be an actor but doesn't have to. However, without seeing entity instances, we cannot conclude the truthfulness of this statement.

**g. A producer cannot be an actor in some other movie.**

**Ans**

**False -** The producer can produce many movies and can also be an actor in movies he produced and in movies he didn't produce. There is no noted restriction that would disallow this

**h. There are movies with more than a dozen actors.**

**Ans**

**Maybe -** We cannot know this because only the maximum participation limit is noted in this ER diagram (and the maximum is also not specified, noted by M). A movie has to have at least one actor, but it is not known whether any movie has more than 12 actors.

**i. Some producers have been a director as well.**

**Ans**

**Maybe -** There is no notation that would either confirm or deny this. It may be true or false, and the truthfulness of this statement can be confirmed or denied only by actually inspecting entity instances

**j. Most movies have one director and one producer.**

**Ans**

**Maybe -** A movie does have only one director. This is noted by the integer above the line connecting DIRECTOR entity type to DIRECTS relationship type. However, a movie can have many producers and we have no information whether "most" movies have a single or many producers

**k. Some movies have one director but several producers.**

**Ans**

**Maybe -** Every movie has one director, so that part of the statement is true, but it may be possible that every movie in this particular database has only one producer. Taking this into account, this statement is not necessarily true or false.

**l. There are some actors who have done a lead role, directed a movie, and**

**produced a movie.**

**Ans**

**Maybe -** In this database, we do not have any information if this statement is true or false. There may be some actors that did each of the three roles, but we are not given that information from this ER diagram

**m. No movie has a director who also acted in that movie.**

**Ans**

**Maybe -** There are no restrictions for this to be a true statement, so it can be false. But, on the other hand, this may be true if all the movies in the database don't have the director who acted in the movie he directed. This statement's truthfulness is also based on entity instances in the database.

**3.29. Given the ER schema for the MOVIES database in Figure 3.25, draw an**

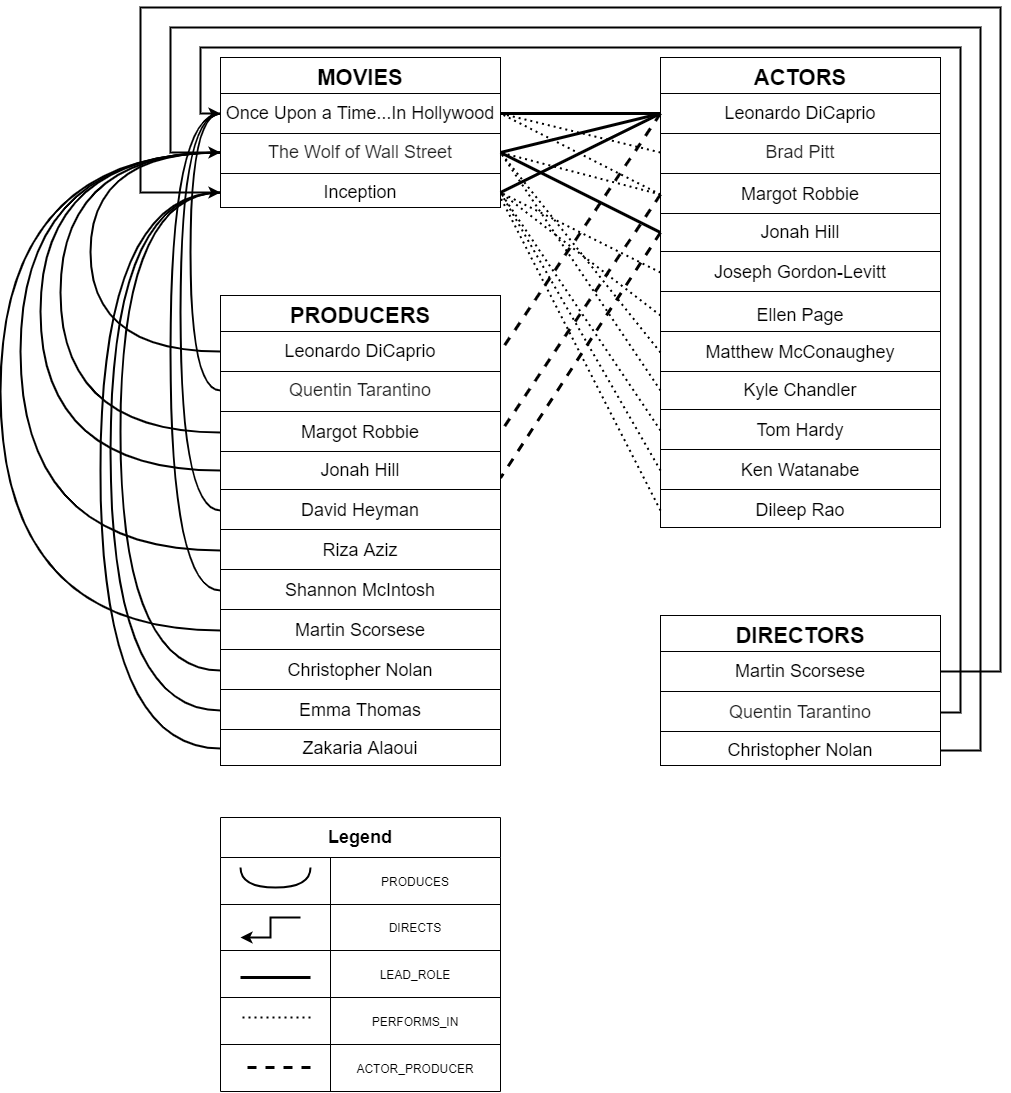
**instance diagram using three movies that have been released recently.**

**Draw instances of each entity type: MOVIES, ACTORS, PRODUCERS,**

**DIRECTORS involved; make up instances of the relationships as they exist in**

**reality for those movies.**

**Ans**



This instance diagram**instance diagram** shows instances of three popular movies, their actors, directors and producers and relationships among them. There is also a legend showing which types of lines are representing which relationship type. \ldots

**3.30. Illustrate the UML diagram for Exercise 3.16. Your UML design should**

**observe the following requirements:**

**a. A student should have the ability to compute his/her GPA and add or**

**drop majors and minors.**

**Ans**

**b. Each department should be able to add or delete courses and hire or ter**

**minate faculty.**

**Ans**

**c. Each instructor should be able to assign or change a student’s grade for a**

**course.**

***Note*: Some of these functions may be spread over multiple classes.**

**Ans**

In this exercise, we present you with the list of added classes and operations**the list of added classes and operations** and an illustration of the UML diagram**the UML diagram** for the UNIVERSITY database.

∙∙ INSTRUCTOR**INSTRUCTOR** class has been added to this UML diagram because of the exercise's requirement that every instructor must be able to assign or change course grades. New association named HAS\_INSTRUCTOR**HAS\_INSTRUCTOR** is made to associate INSTRUCTOR and SECTION classes. Min..max constraint (or multiplicity*multiplicity*) on INSTRUCTOR's participation in this association is 0..\***0..\*** and on SECTION's participation is 1..1**1..1**.

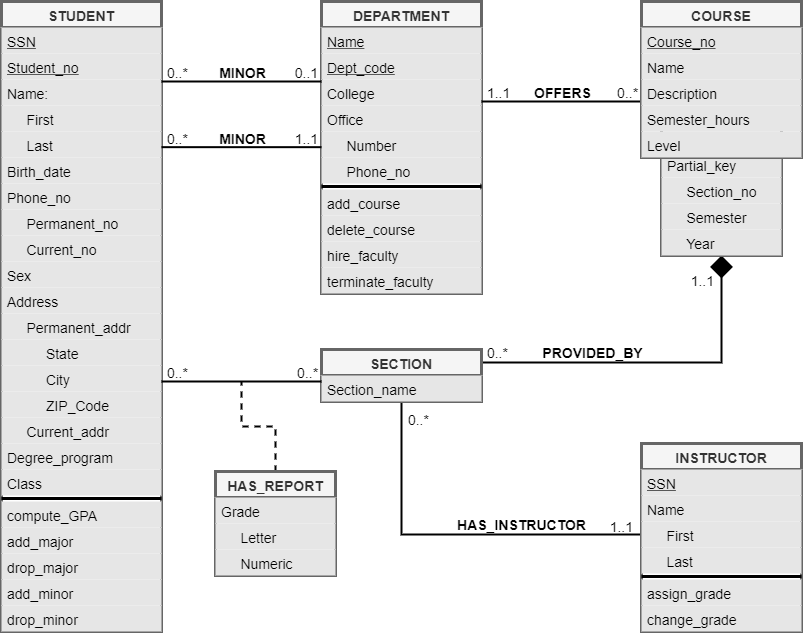
This class has a key attribute SSN‾*SSN*​ and a composite attribute Name*Name* composed of attributes First*First* and Last*Last*.

INSTRUCTOR class has two operations "assign\_grade" and "change\_grade".

∙∙ STUDENT**STUDENT** class is upgraded by operations: ''compute\_GPA'', "add\_major", "drop\_major", "add\_minor" and "drop\_minor". These operations represent what a real life student can do when interacting with the database.

∙∙ DEPARTMENT**DEPARTMENT** class is upgraded with four operations: "add\_course", "delete\_course", "hire\_faculty" and "terminate\_faculty".

∙∙ SECTION**SECTION** class is associated with the COURSE class by a qualified association*qualified association* named PROVIDED\_BY. Partial key attribute now plays the role of the discriminator*discriminator* and is attached to owner class COURSE.



In this exercise, we present you with the list of added classes and operations**the list of added classes and operations** and an illustration of the UML diagram**the UML diagram** for the UNIVERSITY database.

∙∙ INSTRUCTOR**INSTRUCTOR** class has been added to this UML diagram because of the exercise's requirement that every instructor must be able to assign or change course grades. New association named HAS\_INSTRUCTOR**HAS\_INSTRUCTOR** is made ……

**Laboratory Exercises**

**3.31. Consider the UNIVERSITY database described in Exercise 3.16. Build the ER**

**schema for this database using a data modeling tool such as ERwin or**

**Rational Rose.**

**Ans**

**3.32. Consider a MAIL\_ORDER database in which employees take orders for parts**

**from customers. The data requirements are summarized as follows:**

**■ The mail order company has employees, each identified by a unique em**

**ployee number, first and last name, and Zip Code.**

**■ Each customer of the company is identified by a unique customer number,**

**first and last name, and Zip Code.**

**■ Each part sold by the company is identified by a unique part number, a**

**part name, price, and quantity in stock.**

**■ Each order placed by a customer is taken by an employee and is given a**

**unique order number. Each order contains specified quantities of one or**

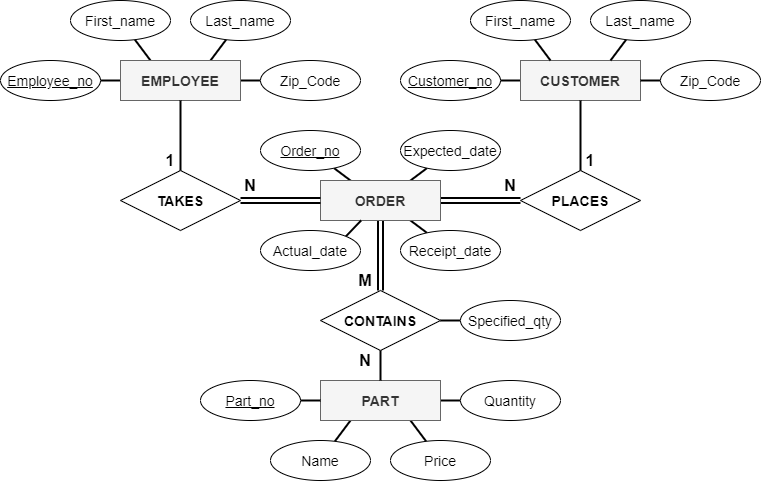
**more parts. Each order has a date of receipt as well as an expected ship**

**date. The actual ship date is also recorded.**

**Design an entity–relationship diagram for the mail order database and build**

**the design using a data modeling tool such as ERwin or Rational Rose.**

**Ans**



**3.33. Consider a MOVIE database in which data is recorded about the movie**

**industry. The data requirements are summarized as follows:**

**■ Each movie is identified by title and year of release. Each movie has a**

**length in minutes. Each has a production company, and each is classified**

**under one or more genres (such as horror, action, drama, and so forth).**

**Each movie has one or more directors and one or more actors appear in it.**

**Each movie also has a plot outline. Finally, each movie has zero or more**

**quotable quotes, each of which is spoken by a particular actor appearing**

**in the movie.**

**■ Actors are identified by name and date of birth and appear in one or more**

**movies. Each actor has a role in the movie.**

**■ Directors are also identified by name and date of birth and direct one or**

**more movies. It is possible for a director to act in a movie (including one**

**that he or she may also direct).**

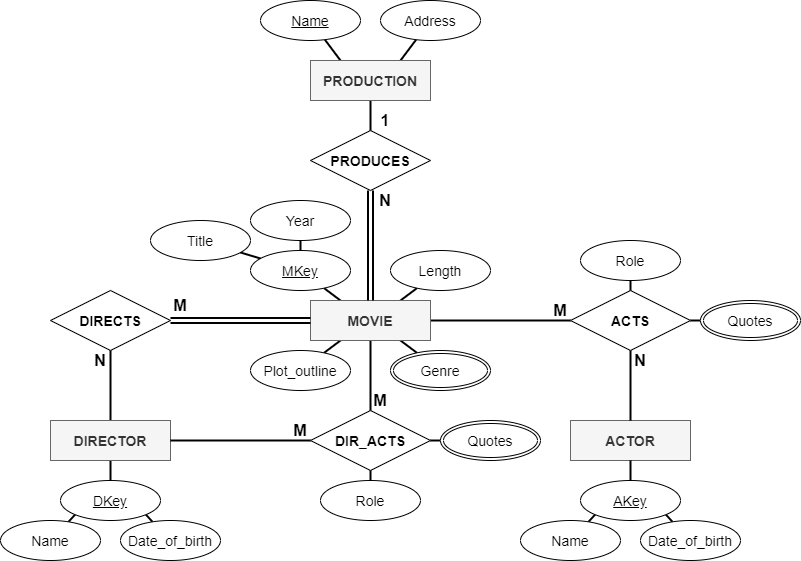
**■ Production companies are identified by name and each has an address. A**

**production company produces one or more movies.**

**Design an entity–relationship diagram for the movie database and enter the**

**design using a data modeling tool such as ERwin or Rational Rose.**

**Ans**



**3.34. Consider a CONFERENCE\_REVIEW database in which researchers submit**

**their research papers for consideration. Reviews by reviewers are recorded**

**for use in the paper selection process. The database system caters primarily**

**to reviewers who record answers to evaluation questions for each paper they**

**review and make recommendations regarding whether to accept or reject**

**the paper. The data requirements are summarized as follows:**

**■ Authors of papers are uniquely identified by e-mail id. First and last names**

**are also recorded.**

**■ Each paper is assigned a unique identifier by the system and is described**

**by a title, abstract, and the name of the electronic file containing the paper.**

**■ A paper may have multiple authors, but one of the authors is designated as**

**the contact author.**

**■ Reviewers of papers are uniquely identified by e-mail address. Each re**

**viewer’s first name, last name, phone number, affiliation, and topics of in**

**terest are also recorded.**

**■ Each paper is assigned between two and four reviewers. A reviewer rates**

**each paper assigned to him or her on a scale of 1 to 10 in four categories:**

**technical merit, readability, originality, and relevance to the conference.**

**Finally, each reviewer provides an overall recommendation regarding**

**each paper.**

**■ Each review contains two types of written comments: one to be seen by**

**the review committee only and the other as feedback to the author(s).**

**Design an entity–relationship diagram for the CONFERENCE\_REVIEW data**

**base and build the design using a data modeling tool such as ERwin or**

**Rational Rose.**

**Ans**

**3.35. Consider the ER diagram for the AIRLINE database shown in Figure 3.21.**

**Build this design using a data modeling tool such as ERwin or Rational Rose**.

**Ans**