Principles of Database Management: Solutions Manual

Chapter 1. Fundamental Concepts of Database Management

Question 1.1 Discuss examples of database applications.

Answer

Some examples of database applications are:

- inventory: monitoring stock levels of products;
- point-of-sale (POS): storing which customer buys what item at what time and location (e.g., in a supermarket);
- geographical information system (GIS): stores and manages geographical data (e.g., Google maps);
- stock trading: used for deciding when to buy or sell stocks depending upon, e.g., macroeconomic trends;
- medical records: stores information about patients, their illnesses, and treatments offered;
- telematics: stores information about the driving behavior of a customer;
- human resources: stores information about employees, their departments, salaries, etc.
- video streaming: stores information about videos (e.g., YouTube).

Question 1.2 What are the key differences between the file versus database approach to data management?

Answer

In the file approach to data management, every application stores its data in its own dedicated files. Since each application uses its own data files, many using similar data, duplicate or redundant information will be stored, resulting in a waste of storage resources. Furthermore, there is a danger that data are updated in only one file and not elsewhere, yielding inconsistent data. In this file-based approach to data management, there is a strong coupling, or dependency, between the applications and the data. A structural change in a data file necessitates changes in all applications that use it, which is not desirable from a maintenance perspective. It is also hard to manage concurrency control (i.e., the simultaneous access by different users or applications to the same data without conflicts). Finally, since the applications each work independently, with their own ecosystem of data files, it is difficult and expensive to integrate applications aimed at providing cross-company services.

In the database approach to data management, all data are stored and managed centrally by a DBMS. The applications now directly interface with the DBMS instead of with their own files. The DBMS delivers the desired data upon request to each application. The DBMS stores and manages two types of data: raw data and metadata. Metadata refers to the data definitions that are now stored in the catalog of the DBMS. This is a key difference from the file-based approach. The metadata are no longer included in the applications, but are now properly managed by the DBMS itself. From an efficiency, consistency, and maintenance perspective, this approach is superior to the file-based approach. Another key advantage of the database approach are the facilities provided for data querying and retrieval. In the file-based approach, every application had to explicitly write its own query and access procedures.

To summarize, the file-based approach results in a strong application–data dependence, whereas the database approach allows for applications to be independent from the data and data definitions.

Question 1.3 Discuss the elements of a database system.

Answer

The key elements of a database system are as follows:

- Database model: provides the description of the database data at different levels of detail and specifies the various data items, their characteristics and relationships, constraints, storage details, etc. Also called database schema.
- Database state: represents the data in the database at a particular moment. Also called the current set of instances.
- Data model: provides a clear and unambiguous description of the data items, their relationships, and various data constraints from a particular perspective. Examples are the conceptual data model, logical data model, internal data model, and external data model.
- Three-layer architecture: an essential element of every database application; describes how the different underlying data models are related.
- Catalog: contains the data definitions, or metadata, of your database application. Also called the heart of the DBMS.
- Database users: users who interact with the database. Examples are information architect, database designer, database administrator, application developer, and the business user.
- DBMS languages: languages used for data definition or data manipulation. Examples are: data definition language (DDL) which is used by the database administrator to express the

database's external, logical, and internal data models; and data manipulation language (DML), which is used to retrieve, insert, delete, and modify data.

Question 1.4 What are the advantages of database systems and database management?

Answer

Databases, if adequately designed and managed, offer various advantages:

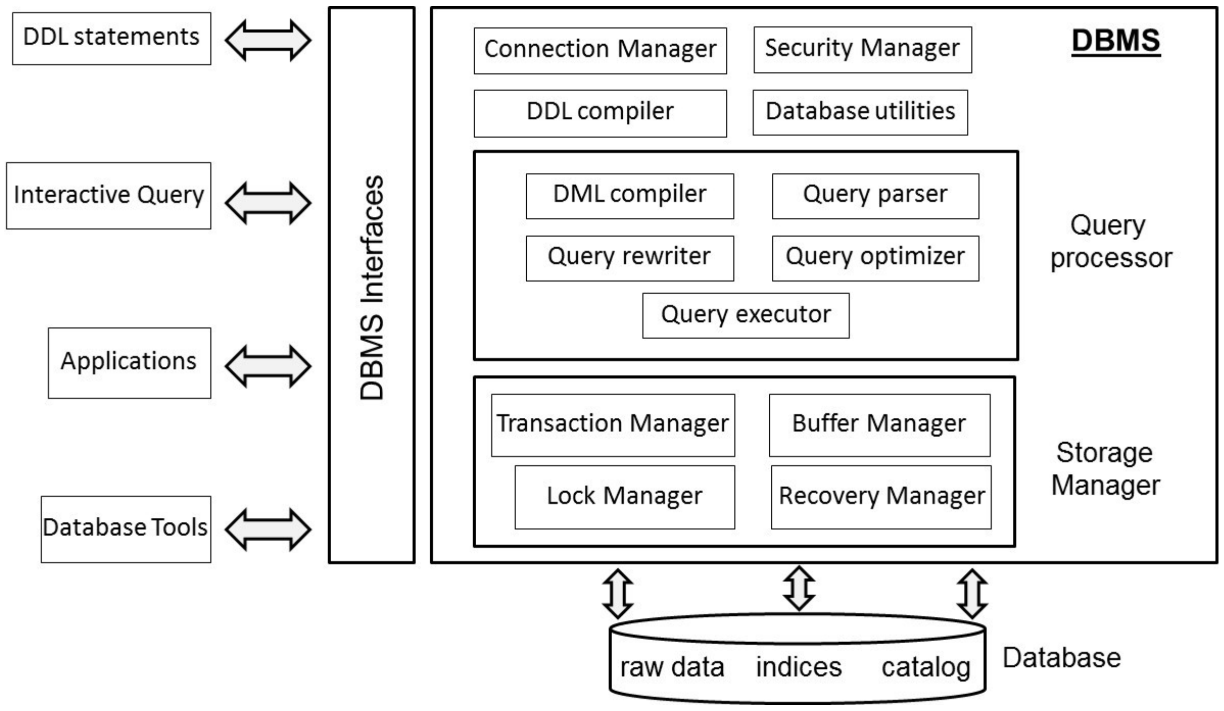
- data independence: refers to the fact that changes in data definitions have minimal to no impact on the applications using the data;
- database modeling: allows provision of an explicit representation of the data items together with their characteristics and relationships;
- managing structured, semi-structured, and unstructured data;
- managing data redundancy: redundant data can now be successfully managed;
- specifying integrity rules: data integrity rules can be explicitly defined that can be used to enforce the correctness of the data;
- concurrency control: a DBMS has built-in facilities to support concurrent or parallel execution of database programs, which allows for good performance;
- backup and recovery facilities: can be used to deal with the effect of loss of data due to hardware or network errors, or bugs in system or application software;
- data security: data security can be directly enforced by the DBMS;
- performance utilities: DBMSs come with various types of utilities aimed at improving response time, throughput rate, or space utilization.

Chapter 2. Architecture and Categorization of DBMSs

Question 2.1 What are the key components of a DBMS architecture and how do they collaborate?

Answer

The following figure illustrates the key components of a DBMS.



The most important components are: the connection manager, the security manager, the DDL compiler, various database utilities, the query processor, and the storage manager. The query processor consists of a DML compiler, query parser, query rewriter, query optimizer, and query executor. The storage manager includes a transaction manager, buffer manager, lock manager, and recovery manager. All of these components interact in various ways, depending upon which database task is executed.

The connection manager provides facilities to set up a database connection. The security manager verifies whether a user has the right privileges to execute the database actions required. The DDL compiler compiles the data definitions specified in DDL. The query processor usually includes a DML compiler, query parser, query rewriter, query optimizer, and query executor. The DML compiler compiles the data manipulation statements specified in DML. The query parser parses the query into an *internal representation format* that can then be further evaluated by the system. The query rewriter optimizes the query, independently of the current database state. The query optimizer optimizes the query based upon the current database state. The query executor takes care of the actual query execution by calling on the storage manager to retrieve the data requested. The storage manager governs physical file access and as such supervises the correct and efficient storage of data. It consists of a transaction manager, buffer manager, lock manager, and recovery manager. The transaction manager supervises the execution of database transactions. The buffer manager is responsible for managing the buffer memory of the DBMS and closely interacts with the lock manager to provide concurrency control support. The lock manager is an essential component for providing concurrency control, which ensures data integrity at all times. The recovery manager supervises the correct execution of database transactions.

Question 2.2 What is the difference between procedural and declarative DML?

Answer

The first data manipulation languages developed were predominantly procedural DML. These DML statements explicitly specified how to navigate in the database to locate and modify the data. They usually started by positioning on one specific record or data instance and navigating to other records, using memory pointers. Therefore, procedural DML is also called record-at-a-time DML. DBMSs with procedural DML had no query processor. In other words, the application developer had to explicitly define the query optimization and execution himself. To write efficient queries, the developer had to know all the details of the DBMS. This is not a preferred implementation since it complicates the efficiency, transparency, and maintenance of the database applications. Unfortunately, many firms are still struggling with procedural DML applications due to the legacy DBMSs still being used today.

Declarative DML is a more efficient implementation. Here, the DML statements specify which data should be retrieved or what changes should be made, rather than how this should be done. The DBMS then autonomously determines the physical execution in terms of access path and navigational strategy. In other words, the DBMS hides the implementation details from the application developer, which facilitates the development of database applications. Declarative DML is usually set-at-a-time DML, whereby sets of records or data instances can be retrieved at once and provided to the application. Only the selection criteria are provided to the DBMS; depending on the actual database state, zero, one, or many records will qualify. A popular example of declarative DML is SQL, as discussed in Chapter 7.

Question 2.3 Why is it important that a DBMS has a good query optimizer?

Answer

The query optimizer is a very important component of the query processor. It optimizes the query based upon the current database state. It can make use of predefined indexes that are part of the internal data model and provide quick access to the data. The query optimizer comes up with various query execution plans and evaluates their cost (in terms of resources required) by aggregating the estimated number of input/output operations, the plan's estimated CPU processing cost, and the plan's estimated execution time into the total estimated response time. A good execution plan should have a low response time. It is important to note that the response time is estimated and not exact. The estimates are made using catalog information combined with statistical inference procedures. Empirical distributions of the data are calculated and summarized by their means, standard deviations, etc. Coming up with accurate estimates is crucial in a good query optimizer. Finding an optimal execution path is essentially a classical search or optimization problem whereby techniques such as dynamic programming can be used. As already mentioned, the implementation of the query optimizer depends upon the type of DBMS and the vendor and is a key competitive asset.

Question 2.4 Give some examples of DBMS utilities and interfaces.

Answer

A DBMS comes with various utilities. A loading utility supports the loading of the database with information from a variety of sources such as another DBMS, text files, Excel files, etc. A reorganization utility automatically re-organizes the data for improved performance. Performance monitoring utilities report various key performance indicators (KPIs), such as storage space consumed, query response times, and transaction throughput rates to monitor the performance of a DBMS. User management utilities support the creation of user groups or accounts, and the assignment of privileges to them. Backup and recovery utilities can be used to recover data in case of hardware or software problems.

A DBMS needs to interact with various parties, such as a database designer, a database administrator, an application, or even an end-user. To facilitate this communication, it provides various user interfaces, such as a web-based interface, a standalone query language interface, a command-line interface, a forms-based interface, a graphical user interface (GUI), a natural language interface, an application programming interface or API for short, an admin interface, and a network interface.

Question 2.5 How can DBMSs be categorized in terms of the following?

- Data model;
- degree of simultaneous access;
- architecture;
- usage.

Answer

A DBMS can be categorized in terms of data model as follows:

- Hierarchical DBMS: one of the first DBMS types developed based on a tree-like data model.
- Network DBMS: based on a network data model that is more flexible than a tree-like data model.
- Relational DBMS (RDBMS): based on the relational data model and the most popular in industry.
- Object-oriented DBMS (OODBMS): based on the object-oriented data model.
- Object-relational DBMS (ORDBMS): based on the relational model extended with object-oriented concepts, such as user-defined types, user-defined functions, collections, inheritance, and behavior; also commonly called an extended relational DBMS (ERDBMS).
- XML DBMS: based on the XML data model to store data.

- Not-only SQL (NoSQL) database: targeted at storing big and unstructured data.

A DBMS can be categorized in terms of degree of simultaneous access as follows:

- Single-user system: only one user at a time is allowed to work with the DBMS.
- Multi-user system: allows multiple users to simultaneously interact with the database in a distributed environment.

A DBMS can be categorized in terms of architecture as follows:

- Centralized DBMS architecture: the data are maintained on a centralized host, e.g., a mainframe system.
- Client–server DBMS architecture: active clients request services from passive servers.
- n-tier DBMS architecture: a straightforward extension of the client–server architecture. A popular example is a client with GUI (graphical user interface) functionality, an application server with various applications, a database server with the DBMS and database, and a web server for the web-based access.
- Cloud DBMS architecture: the DBMS and database are hosted by a third-party cloud provider.

- Federated DBMS: DBMS that provides a uniform interface for multiple underlying data sources such as other DBMSs, file systems, document management systems, etc.
- In-memory DBMS: stores all data in internal memory instead of slower external storage such as disk-based storage.

A DBMS can be categorized in terms of usage as follows:

- Operational versus strategic usage: on-line transaction processing (OLTP) DBMSs focus on managing operational or transactional data, whereas on-line analytical processing (OLAP) DBMSs focus on using operational data for tactical or strategical decision-making.
- Big data and analytics: based on NoSQL DBMSs, which abandon the well-known and popular relational database scheme in favor of a more flexible, or even schema-less, database structure.
- Multimedia DBMS: allows for the storage of multimedia data such as text, images, audio, video, 3D games, CAD designs, etc.
- Spatial DBMS: supports the storage and querying of spatial data.
- Sensor DBMS: manages sensor data such as biometric data obtained from wearables, or telematics data which continuously record driving behavior.
- Mobile DBMS: DBMS running on smartphones, tablets, and other mobile devices.
- Open-source DBMS: DBMS for which the code is publicly available and can be extended by anyone.

Chapter 3. Conceptual Data Modeling using the (E)ER model and UML Class Diagram

Question 3.1 Fitness company "Conan" wants to set up a database for its members and trainers. One of the aims is to record information about which members participated in which sessions and which trainers supervised which sessions.

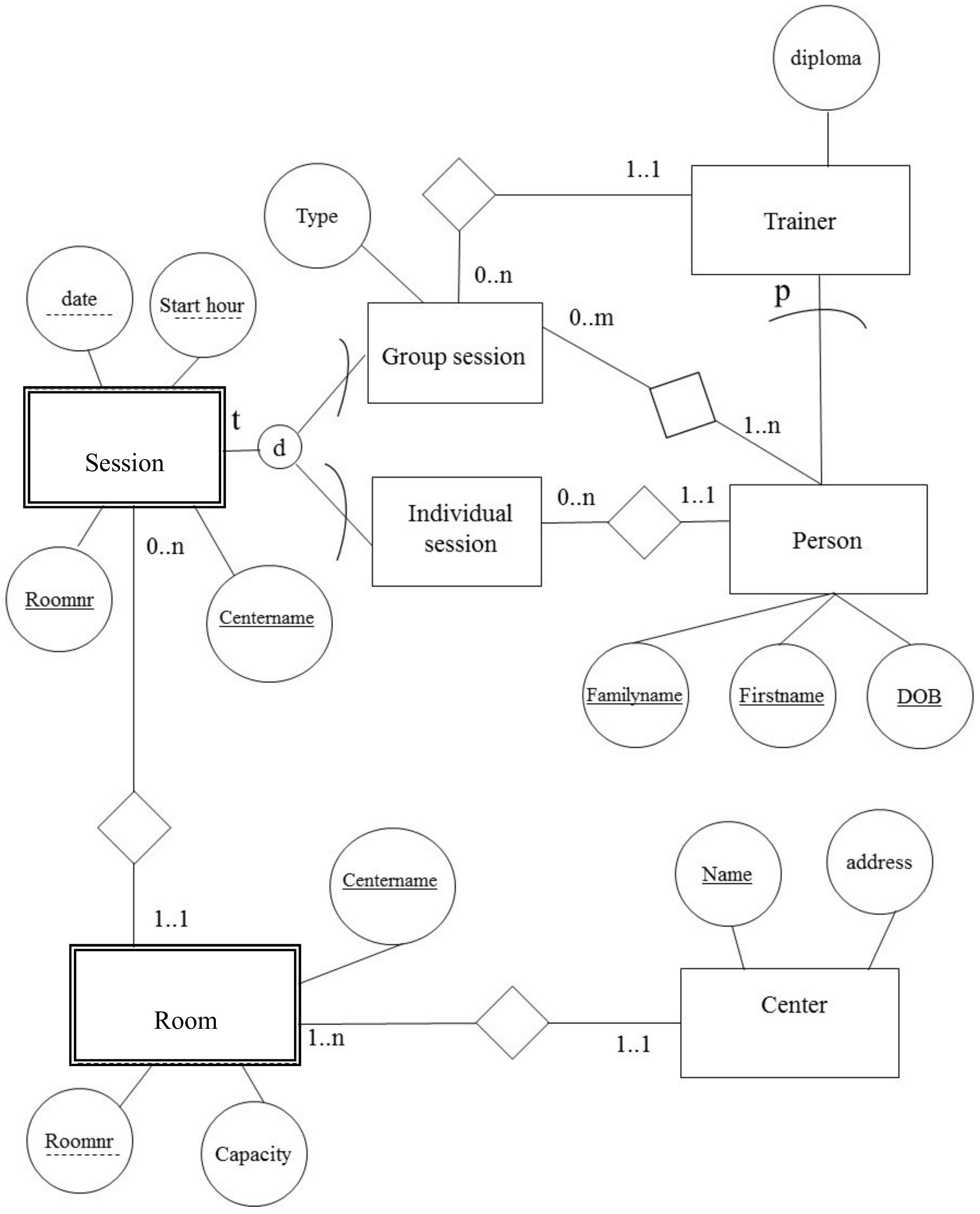
Conan operates various fitness centers in various cities. Every center is characterized by a unique name (e.g., Fitplaza, my6pack). Every center has an address and one or more rooms (you can consider address as atomic). Every room has a maximum capacity. Within a center, each room has a unique number, such as 1, 2, 3, etc.

People can register for individual or group sessions in different centers. Each group session requires exactly one trainer. Individual sessions are done without a trainer. For each person, we want to store the first name, family name, and birth date. You can assume that the combination of first name, family name, and birth date is unique. For each trainer, their diploma is also recorded. A person can be a trainer in one session and participant in another session (either individual or group session). The model should also include information about people (e.g., prospects) who have not participated in any sessions yet, or trainers (e.g., interns) who have not supervised any group sessions yet.

For each session, the date and starting hour should be recorded. For group sessions, also the type should be stored (e.g., aerobics, bodystyling). Sessions can start at the same time on the same day but in different rooms of a center or in different centers. At a given start hour of a given day, at most one individual or group session can start in a given room of a given center.

Make an EER model and UML class diagram to model the data requirements for Conan. Comment on the limitations of both models.

Answer



The EER model has certain shortcomings. Since the EER model is a snapshot in time, it cannot model temporal constraints. Examples of temporal constraints that cannot be enforced by our EER model are:

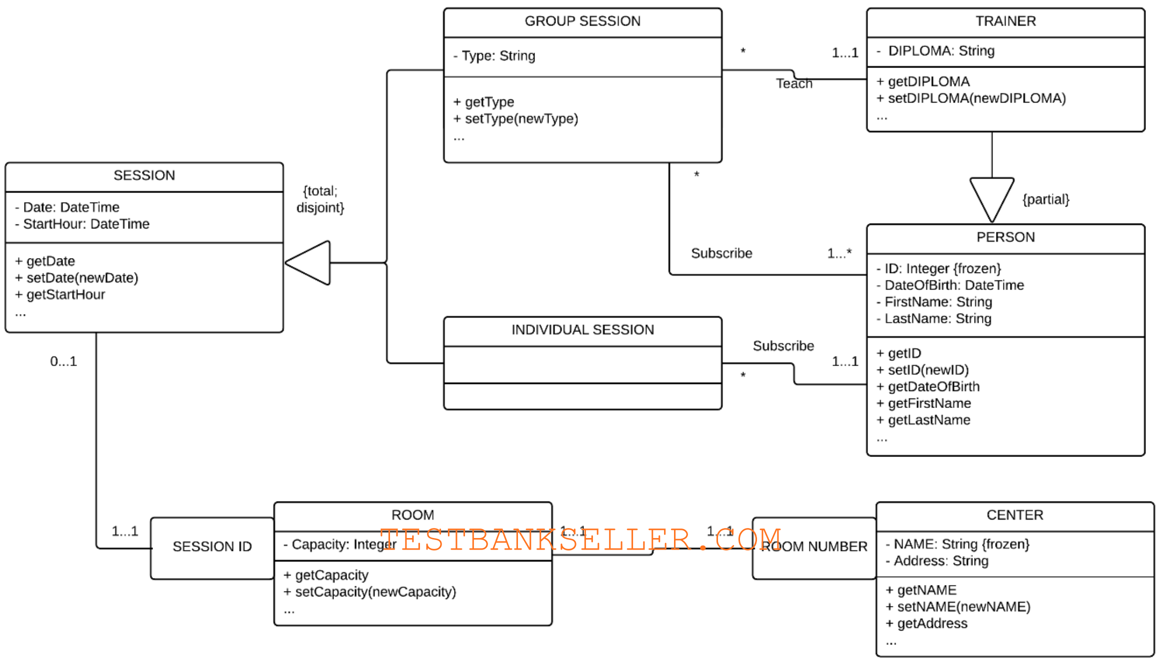
- a person cannot attend a group and individual session at the same time;
- a group session should have a trainer assigned after 1 month;
- there should be a minimum timespan of 15 minutes between subsequent group sessions in a room of a center.

The EER model cannot guarantee the consistency across multiple relationship types. Examples of business rules that cannot be enforced in the EER model are:

- a trainer cannot participate in a group session he/she supervises;
- the number of people that sign up for a group or individual session cannot exceed the room capacity.

The EER model does not support domains – e.g., we cannot specify that the attribute type room capacity should be a positive integer or the attribute type date should be stored as mm/dd/yyyy.

The UML class diagram looks as follows.



The UML specification is semantically richer than its EER counterpart. To enforce information hiding, the access modifiers of all variables have been set to private. Getter and setter methods are used to access them. Additional methods can be added where needed. An example could be a method to retrieve the number of people subscribed to a session. The UML class diagram also specifies the domains (e.g., integer, string) for each of the variables. We set the changeability property of a number of variables to frozen, which means that once a value has been assigned to any of them, it can no longer be changed.

The weak entity types ROOM and SESSION have been implemented using qualified associations. All associations have been defined as bidirectional, which implies that they can be navigated in both directions. The UML class diagram can be further enriched with OCL constraints to implement the constraints that cannot be enforced by the EER model.

Question 3.2 Recently, the European Union made funds available to set up a cross-national research database that stores information concerning scientific articles of researchers working at institutions in the EU. Science Connect is the company that will be setting up this database.

The system will store information regarding scientific staff (persons) and research institutions. Both are uniquely identified by a person ID and an institution code, respectively. Additionally, the following is also recorded for each person: a phone number, keywords that identify his or her key research topics, and the institution he or she works for. A person can be an author of one article and a reviewer of another peer-reviewed article at the same time.

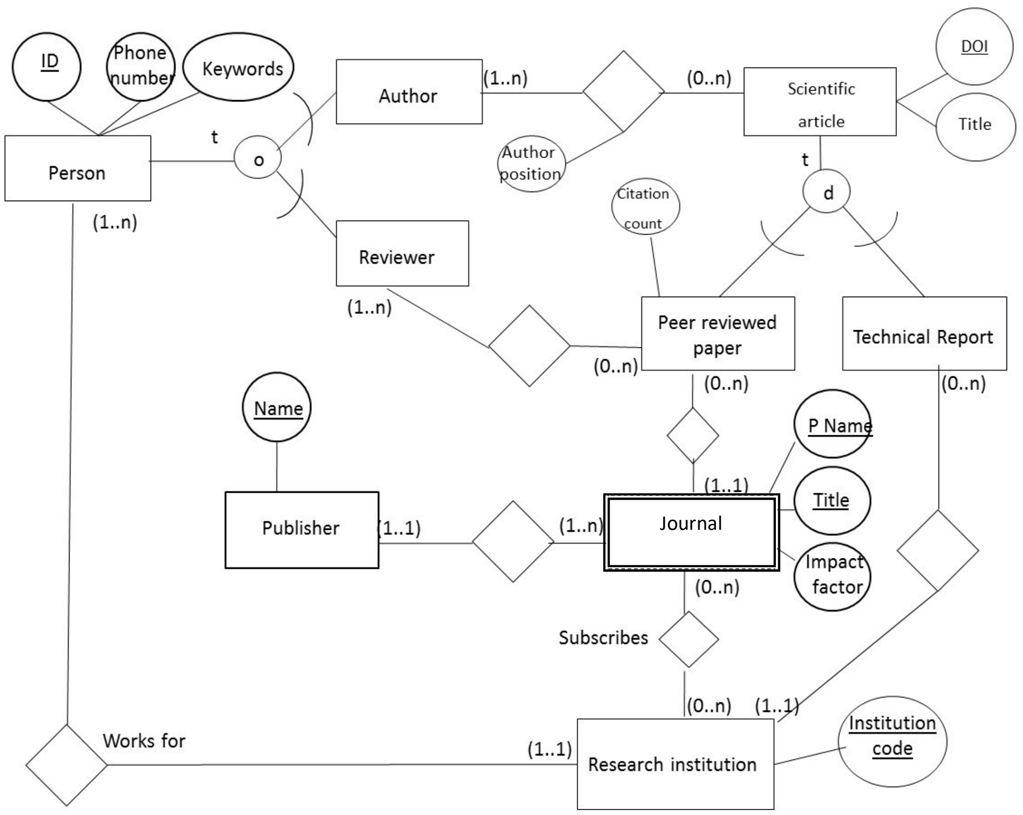
The database will store the following information concerning scientific articles. Each scientific article is uniquely identified by a DOI (a document object identifier), and the system also stores the title and the authors of the article. In the case of multiple authors, the position of each author is stored. Science Connect distinguishes between two types of scientific articles: a scientific article is either a peer-reviewed paper or a technical report. The system stores the citation count of peer-reviewed papers and which persons reviewed the paper. A technical report is always published by a single research institution, while research institutions can publish multiple technical reports.

The system also keeps track of the different scientific publishers (e.g., IEEE, Elsevier). A publisher is identified by its name. A publisher can publish multiple journals to which research institutions can subscribe. These journals are given a name by the publisher (e.g., *Decision Support Systems*). Publishers can have journals with the same name as other publishers (i.e., it is possible that both IEEE and Elsevier have a journal with the title *Management Science*). The impact factor, which measures the scientific impact of a journal, is also stored.

Finally, only peer-reviewed papers are published in journals, not technical reports.

Make an EER model and UML class diagram to model the data requirements. Comment on the limitations of both models.

Answer



The EER model has certain shortcomings. Since the EER model is a snapshot in time, it cannot model temporal constraints. Examples of temporal constraints that cannot be enforced by our EER model are:

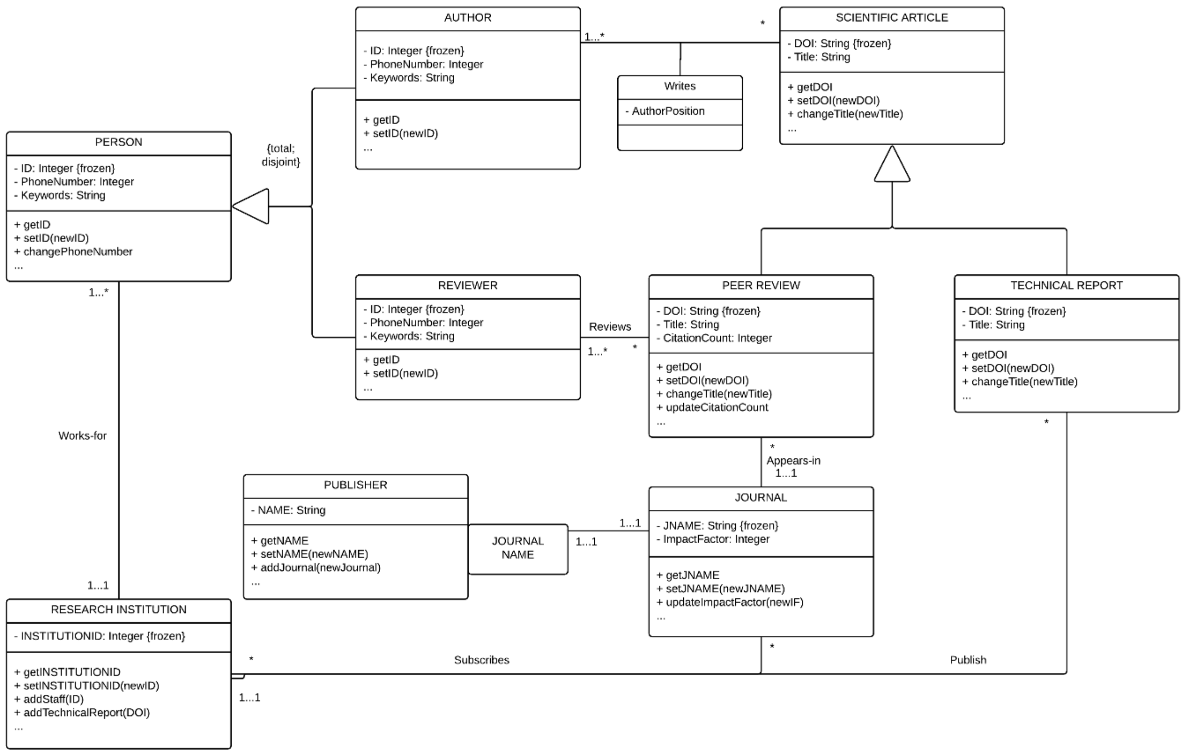
- a peer-reviewed paper should be assigned to a reviewer after one month;
- an author should publish at least three scientific articles per year.

The EER model cannot guarantee the consistency across multiple relationship types. Examples of business rules that cannot be enforced in the EER model are:

- an author cannot be the reviewer of his/her own peer-reviewed paper;
- an author cannot write a technical report for a different research institution than the one he/she works for.

The EER model does not support domains – e.g., we cannot specify that the attribute impact factor should be a positive integer or that keywords should be a set of at most three strings.

The UML class diagram looks as follows.



The UML specification is semantically richer than its EER counterpart. To enforce information hiding, the access modifiers of all variables have been set to private. Getter and setter methods are used to access them. Additional methods can be added where needed. An example could be a method that calculates the number of scientific articles a person wrote. The UML class diagram also specifies the domains (e.g., integer, string) for each of the variables. We set the changeability property of a number of variables to frozen, which means that once a value has been assigned to any of them, it can no longer be changed.

The weak entity type JOURNAL has been implemented using a qualified association. All associations have been defined as bidirectional, which implies that they can be navigated in both directions. The UML class diagram can be further enriched with OCL constraints to implement the constraints that cannot be enforced by the EER model.

Question 3.3 One of your (hipster) acquaintances thinks he has the next billion-dollar start-up idea for an app: Pizza Delivery with Entertainment. He heard from other people that you are following the course on database management, and asks you to design the EER model. Afterwards, he will use the EER model to ask programmers to implement the app.

He explains the basic functionality of the app as follows: customers can order pizzas from restaurants to get delivered to a specific address, and if they want to, they can choose a special "entertainment order." The following is a detailed explanation of the range of capabilities of the app.

When people create an account for the app and become app users, they have to indicate their birthday and fill in their name and address. Every user should also be uniquely identifiable.

Once the account is created, the users should be presented with three options.

The first option in the app is to select "business owner." We also ask these business owners to provide their LinkedIn account so we can add them to our professional network. Each business owner can own a number of pizza restaurants. Of these pizza restaurants, we want to register the zip code, address, phone number, website, and the opening hours.

Each pizza restaurant can offer a number of pizzas. Of those pizzas, we want to keep the name (e.g., margherita, quattro stagioni), the crust structure (for example, classic Italian crust, deep dish crust, cheese crust), and the price. While two pizzas from different pizza restaurants may have the same name, they will not be exactly the same, as the taste will be different, and thus should be considered unique. Moreover, pizzas should be distinguishable even if they have the same price, e.g., a pizza margherita from Pizza Pronto in New York that costs €12 must be distinguishable from a pizza margherita from Pizza Rapido in Singapore, which also costs €12.

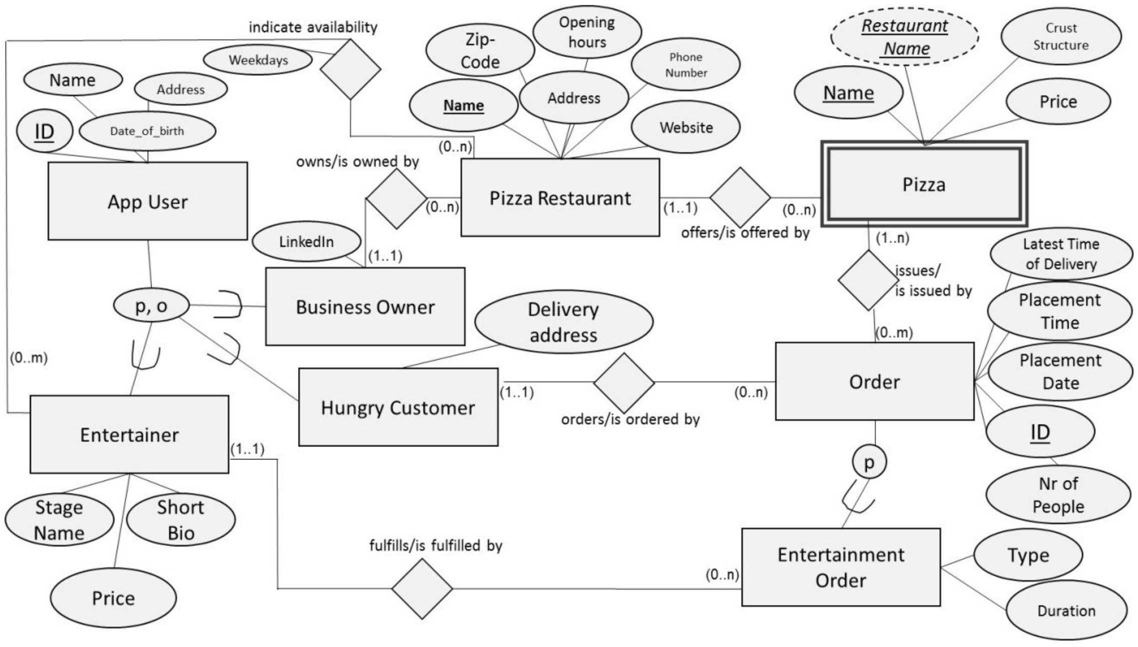
The second option in the app is to select "hungry customer." Of those hungry customers, we need a delivery address. Hungry customers can make orders for pizzas. Each order gets assigned an ID, and we want our app to log the date and time when the order was placed. We also allow the hungry customer to indicate the latest time of delivery, and ask how many people the order is for. An order can be for one or more pizzas.

A special type of order can be made: the entertainment order. When an order is an entertainment order, the delivery person stays with the customer after delivering the pizza and entertains the customers (e.g., singing, making jokes, doing magic tricks) for a certain amount of time. When a hungry customer indicates that he or she wants to be entertained while eating the pizza, we not only want to register all the regular order information, but also the type of entertainment the user requests, and for how long (a duration).

The third option in the app is that of "entertainer." When users select entertainer, they need to provide a stage name, write a short bio about themselves, and indicate their price per 30 minutes. Every entertainment order is fulfilled by exactly one entertainer. Every entertainer can choose for which pizza restaurant(s) he or she wants to work. For each pizza restaurant an entertainer wants to work with, he or she should indicate his or her availability by day (Monday, Tuesday, Wednesday, etc.).

Make an EER model and UML class diagram to model the data requirements. Comment on the limitations of both models.

Answer



The EER model has certain shortcomings. Since the EER model is a snapshot in time, it cannot model temporal constraints. Examples of temporal constraints that cannot be enforced by our EER model are:

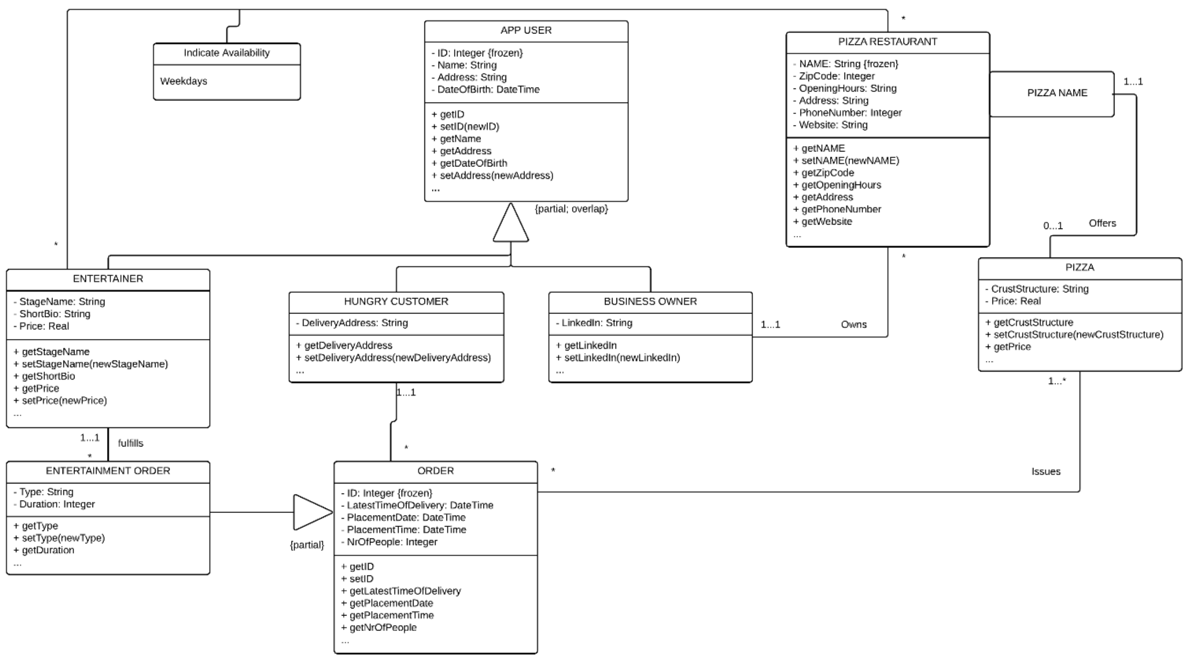
- the time between subsequent entertainment orders of an entertainer should be at least two hours;
- a business owner should own a restaurant within one year after it has been defined;

The EER model cannot guarantee the consistency across multiple relationship types. Examples of business rules which cannot be enforced in the EER model are:

- an entertainer cannot be a hungry customer for the same order;
- a business owner cannot be a hungry customer for a pizza from another business owner.

The EER model does not support domains – e.g., we cannot specify that the attribute type Price must be a positive integer or Crust Structure can only be one of the following: thin, cheesy, or thick.

The UML class diagram looks as follows.



The UML specification is semantically richer than its EER counterpart. To enforce information hiding, the access modifiers of all variables have been set to private. Getter and setter methods are used to access them. Additional methods can be added where needed. An example could be a method that calculates how many entertainers have also been registered as hungry customers. The UML class diagram also specifies the domains (e.g., integer, string) for each of the variables. We set the changeability property of a number of variables to frozen, which means that once a value has been assigned to any of them, it can no longer be changed.

The weak entity type PIZZA has been implemented using a qualified association. All associations have been defined as bidirectional, which implies that they can be navigated in both directions. The UML class diagram can be further enriched with OCL constraints to implement the constraints that cannot be enforced by the EER model.

Question 3.4 Attracted by the success of Spotify, a group of students want to build their own music-streaming website called Musicmatic. Being economists, they are unaware of the specificities of databases and have therefore asked you to create an EER model.

A large number of songs will be made available through their website, and the following information on each song needs to be stored: title, year, length, and genre. Also, artist information will be added, including date of birth, name, and a URL to a website (e.g., Wikipedia page) with additional information on the artist. You can assume an artist is uniquely identified by his name, and that a song always belongs to exactly one artist. The Musicmatic students also point out that songs having the same title are possible, and only the combination of song and artist can be assumed to be unique.

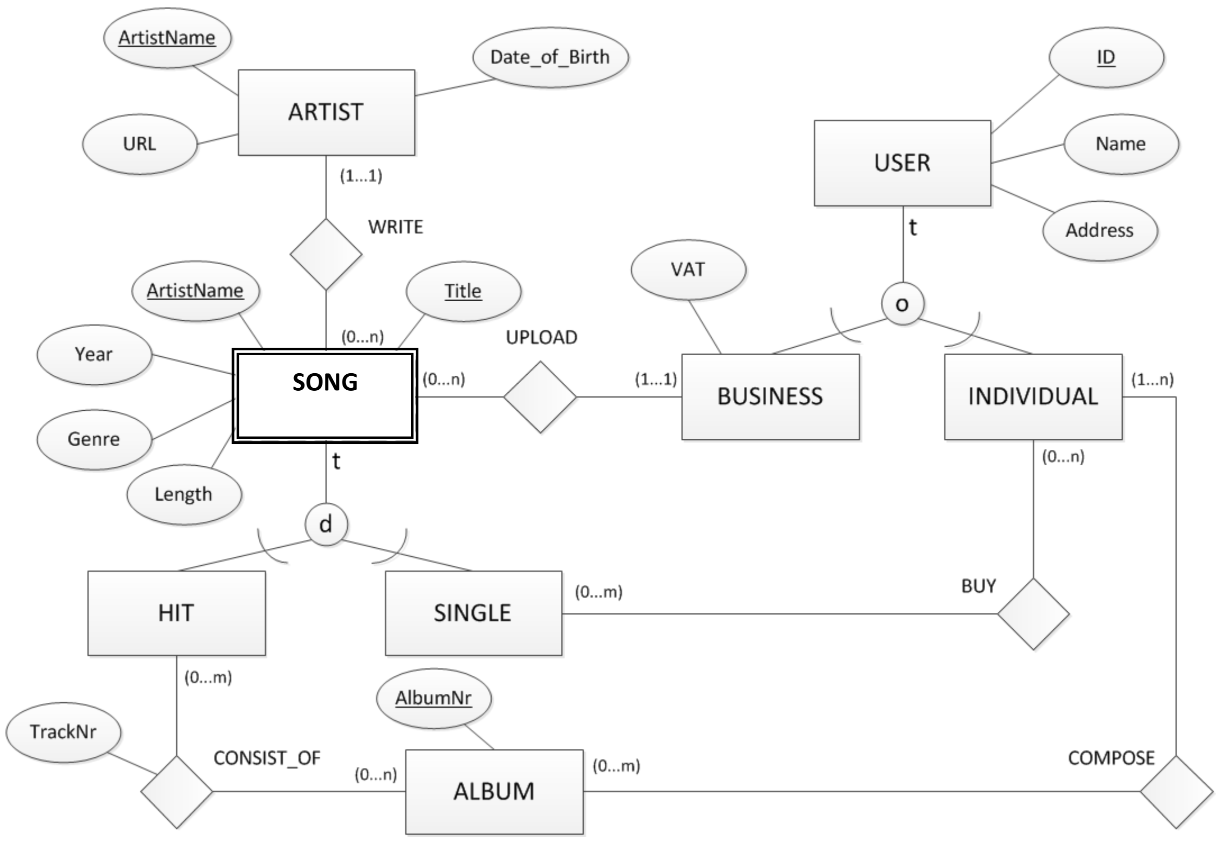
The database will also have to store information on the people using Musicmatic. It was decided to only discriminate between two types of users: regular users who will be able to buy music, and business users who will deliver the content (upload the music). The following information is recorded on each user: (unique) ID, name, and address. Business users will also have a VAT number.

The students want to offer a flexible service, and decided business users can only upload individual songs. These songs are classified either as singles or hits, and regular users can directly buy the singles. Otherwise, people can compose an album consisting of multiple hits (no singles). An album should be uniquely identifiable by an album number. The position of each hit in the album is stored as a track number in the database. The album of regular users can be turned into a suggestion to other regular users with similar purchasing behavior.

Finally, a user can be a regular user on some occasions (e.g., when downloading a single or album), and a business user at other times (e.g., when uploading self-made songs to Musicmatic).

Make an EER model and UML class diagram to model the data requirements. Comment on the limitations of both models.

Answer



The EER model has certain shortcomings. Since the EER model is a snapshot in time, it cannot model temporal constraints. Examples of temporal constraints that cannot be enforced by our EER model are:

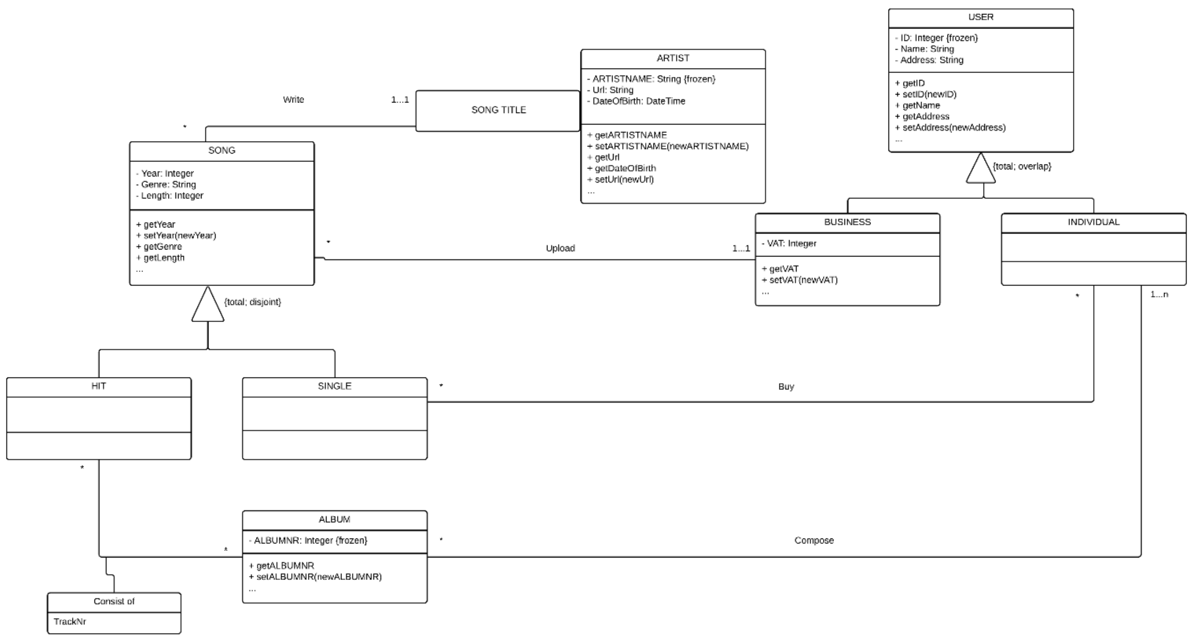
- each individual should buy a song within six months after it has been defined;
- businesses must upload songs on at least an annual basis.

The EER model cannot guarantee consistency across multiple relationship types. Examples of business rules that cannot be enforced in the EER model are:

- a user should not be able to buy a song (as an individual) it has uploaded before (as a business);
- a user can only compose an album (as an individual) of songs that he/she has not uploaded (as a business).

The EER model does not support domains – e.g., we cannot specify that the attribute type TrackNr must be a positive integer and Date_of_Birth must be stored as mm/dd/yyyy. The EER model does not support behavior. We cannot implement the album recommendation system that suggests albums to other regular users with similar purchasing behavior.

The UML class diagram looks as follows.



The UML specification is semantically richer than its EER counterpart. To enforce information hiding, the access modifiers of all variables have been set to private. Getter and setter methods are used to access them. Additional methods can be added where needed. An example could be a method that calculates the average number of hits per album. The UML class diagram also specifies the domains (e.g., integer, string) for each of the variables. We set the changeability property of a number of variables to frozen, which means that once a value has been assigned to any of them, it can no longer be changed.

The weak entity type ARTIST has been implemented using a qualified association. All associations have been defined as bidirectional, which implies that they can be navigated in both directions. The UML class diagram can be further enriched with OCL constraints to implement the constraints that cannot be enforced by the EER model.

Question 3.5 Recently, a new social network site, Facepage, was founded. Given the current trends, the managers of Facepage are convinced that this will be the new hype in the near future.

When new users want to join Facepage, they first need to fill in a form with their personal information (i.e., ID, name, email, and date of birth). A user has a unique ID. Afterwards, an account is created. An account is uniquely identified by an account number, automatically generated by the database system. The user needs to specify which type of account he/she prefers: a business account or a personal account. A business account is specifically designed to support companies in their marketing campaigns. When a user decides to open a business account, he/she has to specify the name of the company. Users with a business account pay a monthly fee. When a user opts for a personal account, he/she can keep in touch with other Facepage users. Only personal accounts can send or receive friend requests.

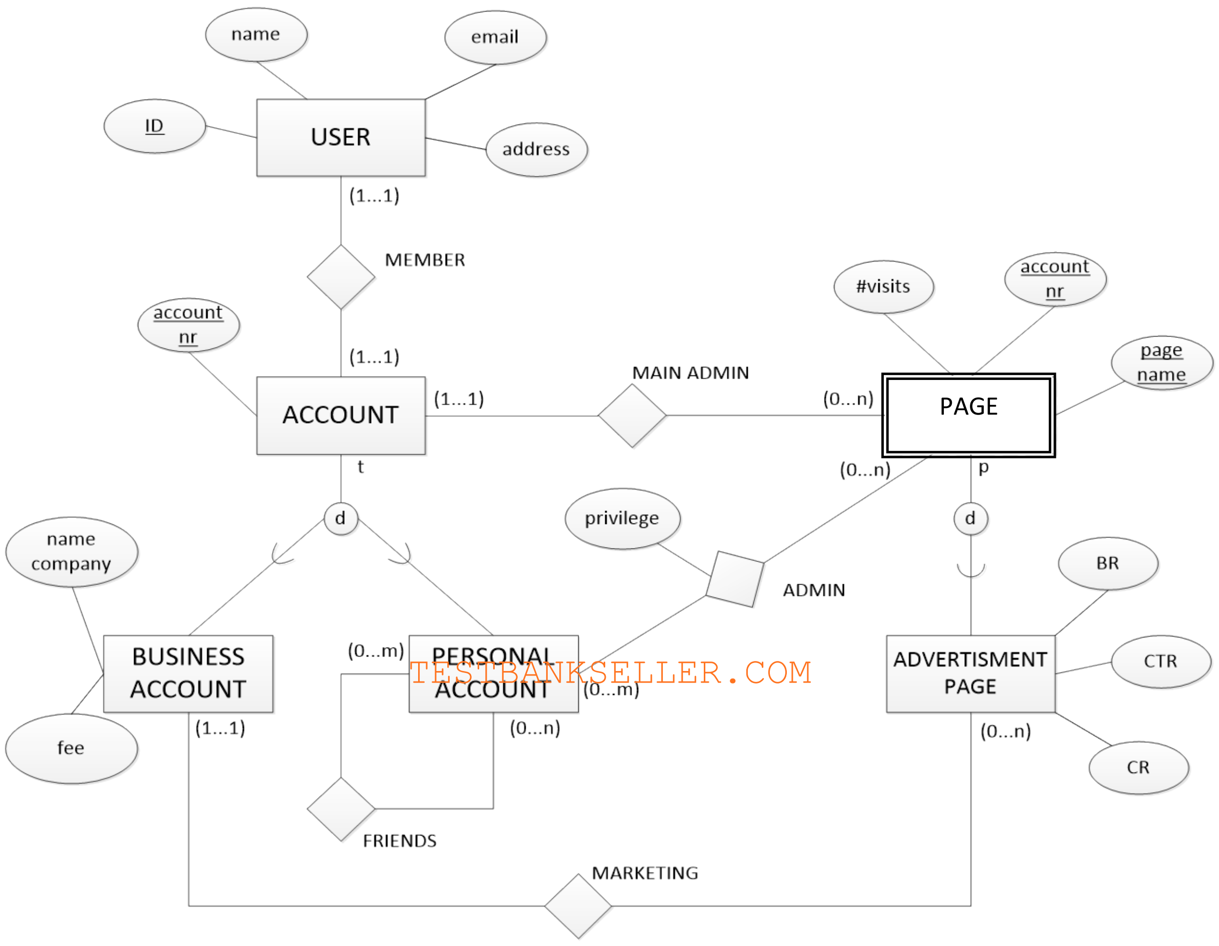
Maintaining multiple accounts, regardless of the purpose, is a violation of Facepage's Terms of Use. If a user already has a personal (business) account, then Facepage cannot allow the user to create an additional personal or business account for any reason.

Each account can create several pages. While each page must be administrated by exactly one account, personal accounts can be granted privileges (e.g., to write something on the wall of friends, adjust some information) to pages belonging to other personal accounts. For each page, the page name and the number of visits are logged. For each account, no two pages can exist with the same name. Users with a business profile can create a special type of page: an advertisement page. This page records several features, like the bounce rate, the click-through rate, and the conversion rate. The bounce rate is the percentage of visitors on the page that leave immediately. The click-through rate is the percentage of visitors that click on a certain banner on the page. The conversion rate is the percentage of visitors that accomplish the intended goal, like a purchase or a transaction.

Make an EER model and UML class diagram to model the data requirements for Facepage. Comment on the limitations of both models.

Answer

An example EER model looks as follows.



The EER model has certain shortcomings. Since the EER model is a snapshot in time, it cannot model temporal constraints. Examples of temporal constraints that cannot be enforced by our EER model are:

- a user can only have one account at a given time, but multiple accounts in time;
- each business account should set up an advertisement page within six months after it was created.

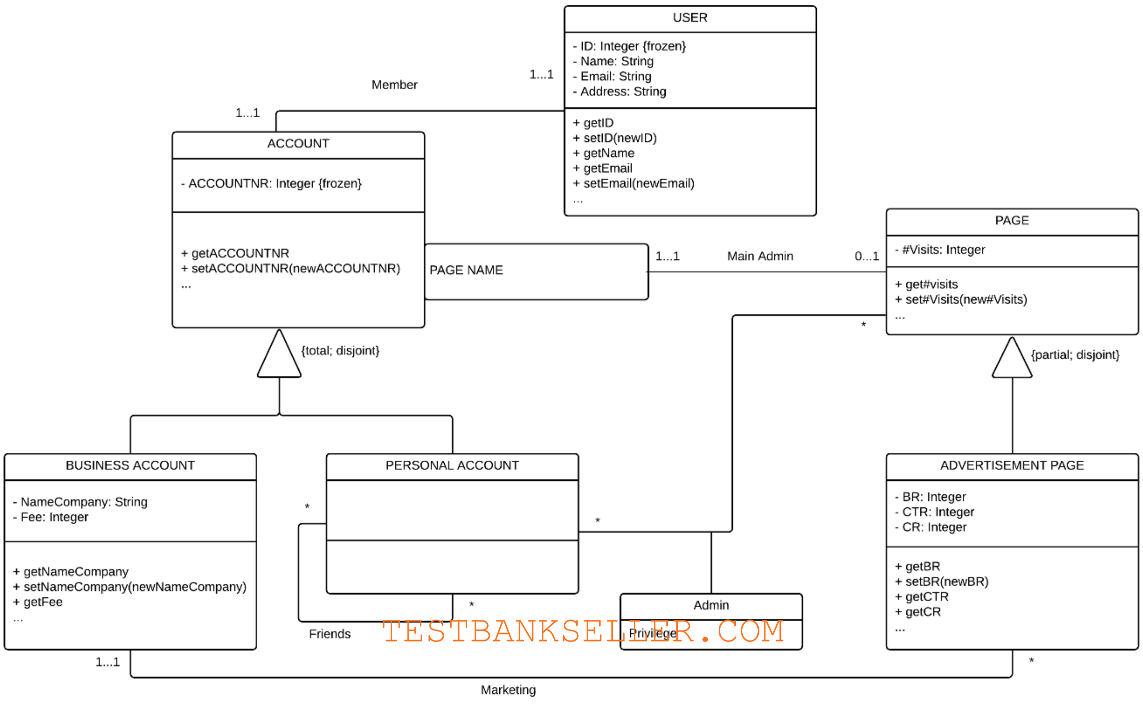
The EER model cannot guarantee the consistency across multiple relationship types. Examples of business rules that cannot be enforced in the EER model are:

- it should not be possible for a business user to create an advertisement page for an account that belongs to another business user;

- it should not be possible for a personal account to be friends with itself.

The EER model does not support domains – e.g., we cannot specify that the attribute types account-nr, BR, CTR, and CR must be positive integers.

The UML class diagram looks as follows.



The UML specification is semantically richer than its EER counterpart. To enforce information hiding, the access modifiers of all variables have been set to private. Getter and setter methods are used to access them. Additional methods can be added where needed. An example could be a method that calculates the average number of friends of an account. The UML class diagram also specifies the domains (e.g., integer, string) for each of the variables. We set the changeability property of a number of variables to frozen, which means that once a value has been assigned to any of them, it can no longer be changed.

The weak entity type PAGE has been implemented using a qualified association. All associations have been defined as bidirectional, which implies that they can be navigated in both directions. The UML class diagram can be further enriched with OCL constraints to implement the constraints that cannot be enforced by the EER model.

Chapter 4. Organizational Aspects of Data Management

Question 4.1 Discuss the importance of metadata modeling and catalogs.

Answer

The importance of good metadata management cannot be understated. In the past, this was often neglected, resulting in significant problems when applications needed to be updated or maintained. In the file-based approach to data management, the metadata were stored in each application separately, thereby creating various problems. Just as raw data, also metadata are data that need to be properly modeled, stored, and managed. Hence, the concepts of data modeling should also be applied to metadata in a transparent way. A metamodel is a data model for metadata. In other words, a metamodel determines the type of metadata that can be stored. Just as with raw data, a database design process can be used to design a database storing metadata.

In a DBMS approach, metadata are stored in a catalog, sometimes also referred to as a data dictionary or data repository, which constitutes the heart of the database system. This facilitates the efficient answering of questions such as which data are stored where in the database? Who is the owner of the data? Who has access to the data? How are the data defined and structured? Which transactions work with which data? Are the data replicated and how can consistency be guaranteed? Which integrity rules are defined? How frequently are backups made? The catalog can be an integral part of a DBMS or a standalone component that must be updated manually. Quite obviously, the integrated solution is to be preferred and thus is more prevalent in modern-day DBMSs. The catalog provides an important source of information for end-users, application developers, as well as the DBMS itself. Remember, the data definitions are generated by the DDL compiler. The DML compiler and query processor use the metadata to solve queries and determine the optimal access path.

Question 4.2 Define data quality and discuss why it is an important concept. What are the most important data quality dimensions? Illustrate with examples.

Answer

Data quality is often defined as “fitness for use,” which implies the relative nature of the concept. Data that are of acceptable quality in one decision context may be perceived to be of poor quality in another decision context, even by the same business user. Data quality determines the intrinsic value of the data to the business. Information technology only serves as a magnifier for this intrinsic value. Hence, high-quality data combined with effective technology is a great asset, but poor-quality data combined with effective technology is an equally great liability. This is sometimes also referred to as the GIGO, or Garbage In Garbage Out principle, stating that bad data results in bad decisions, even with the best technology available. Decisions made based on useless data have cost companies billions of dollars. A popular example of this is the address of a customer. It is estimated that approximately 10% of customers change their address on a yearly basis. Obsolete customer addresses can have substantial consequences for mail order companies, package delivery providers, or government services.

Data quality is a multidimensional concept in which each dimension represents a single aspect or construct, comprising both objective and subjective perspectives. Some aspects are absolute, whereas others depend on the type of task and/or experience of the data user. Therefore, it is useful to define data quality in terms of its dimensions. The following table enumerates the most important data quality dimensions grouped into four categories: intrinsic, contextual, representation, and access.

Cat.	Data quality dimensions	Definitions
Intrinsic	Accuracy	The extent to which data are certified, error-free, correct, flawless, and reliable.
	Objectivity	The extent to which data are unbiased, unprejudiced, based on facts, and impartial.
	Reputation	The extent to which data are highly regarded in terms of their sources or content.
Contextual	Completeness	The extent to which data are not missing and cover the needs of the tasks and are of sufficient breadth and depth for the task at hand.
	Appropriate amount	The extent to which the volume of data is appropriate for the task at hand.
	Value-added	The extent to which data are beneficial and provide advantages from their use.
	Relevance	The extent to which data are applicable and helpful for the task at hand.
	Timeliness	The extent to which data are sufficiently up-to-date for the task at hand
	Actionable	The extent to which data are ready for use
Representation	Interpretable	The extent to which data are in appropriate languages, symbols, and the definitions are clear.
	Easily understandable	The extent to which data are easily comprehended.
	Consistency	The extent to which data are continuously presented in the same format.
	Concisely represented	The extent to which data are compactly represented, well presented, well organized, and well formatted.
	Alignment	The extent to which data are reconcilable (compatible).
Access	Accessibility	The extent to which data are available, or easily and swiftly retrievable.
	Security	The extent to which access to data is appropriately restricted to maintain its security.
	Traceability	The extent to which data are traceable to the source.

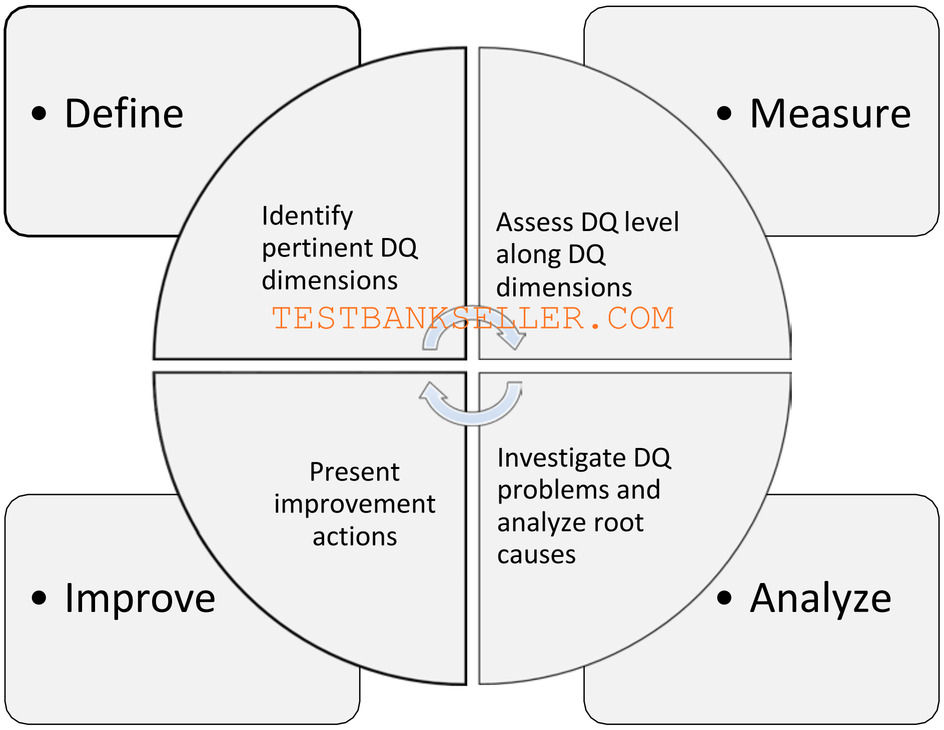
The following are examples of the above dimensions:

- Accuracy: correct birthdate is February 27, 1975, but the recorded value is February 27, 1957.
- Completeness: date of birth is missing.
- Consistency: both date of birth and age are stored, but their values do not correspond.

Question 4.3 Discuss the Total Data Quality Management (TDQM) data governance framework and illustrate with examples.

Answer

The TDQM framework is illustrated in the below figure. A TDQM cycle consists of four steps: *define*, *measure*, analyze, and *improve*, which are performed iteratively. The *define* step identifies the pertinent data quality dimensions. These can then be quantified using metrics in the *measure* step. Some example metrics are: the percentage of customer records with incorrect address (accuracy), the percentage of customer records with missing birth date (completeness), or an indicator specifying when customer data were last updated (timeliness). The *analyze* step tries to identify the root cause of the diagnosed data quality problems. These can then be remedied in the *improve* step. Example actions could be: automatic and periodic verification of customer addresses, the addition of a constraint that makes birth date a mandatory data field, and the generation of alerts when customer data have not been updated during the previous six months.


• Define
Identify pertinent DQ dimensions
• Measure
Assess DQ level along DQ dimensions
• Analyze
Investigate DQ problems and analyze root causes
• Improve
Present improvement actions

If actual data quality improvement is not an option in the short term for reasons of technical constraints or strategic priorities, it is sometimes a partial solution to annotate the data with explicit information about its quality. Such data quality metadata can be stored in the catalog, possibly along with other metadata. In this way, the data quality issues are not resolved, but at least data consumers across the organization are aware of them and can take the necessary precautions as part of their task execution. For example, credit risk models could incorporate an additional risk factor to account for uncertainty in the data, derived from data quality metadata. Unfortunately, many companies still ignore data quality problems because of a lack of perceived added value. Hence, many data governance efforts (if any) are mostly reactive and ad-hoc, only addressing the data quality issues as they occur.

Question 4.4 Discuss and contrast the various roles in data management. Clearly indicate the key activities and skills required. Discuss which job profiles can be combined.

Answer

The following roles in data management can be identified: information architect, database designer, data owner, data steward, database administrator, and data scientist. Each of these roles are essential in ensuring high data quality and transforming data into actual business value. Depending on the size of the database and the company, multiple profiles may be merged into one job description.

The information architect (also called information analyst) designs the conceptual data model, preferably in dialogue with the business users. He/she bridges the gap between the business processes and the IT environment and closely collaborates with the database designer, who may assist in choosing the type of conceptual data model (e.g., EER or UML) and the database modeling tool.

The database designer translates the conceptual data model into a logical and internal data model. He/she also assists the application developers in defining the views of the external data model. To facilitate future maintenance of the database applications, the database designer should define company-wide uniform naming conventions when creating the various data models. The information architect and database designer job profiles can be combined.

Every data field in every database in the organization should be owned by a data owner, who is in the authority to ultimately decide on the access to, and usage of, the data. The data owner could be the original producer of the data, one of its consumers, or a third party. The data owner should be able to fill in or update its value, which implies that the data owner has knowledge of the meaning of the field and has access to the current correct value (e.g., by contacting a customer, by looking into a file, etc.). Data owners can be requested by data stewards (see below) to check or complete the value of a field, as such correcting a data quality issue.

Data stewards are the data quality experts in charge of ensuring the quality of both the actual business data and the corresponding metadata. They assess data quality by performing extensive and regular data quality checks. However, data stewards are not in charge of correcting data themselves, as this is typically the responsibility of the data owner. Both the data owner and data steward job profiles can be combined.

The database administrator (DBA) is responsible for the implementation and monitoring of the database. Example activities include: installing and upgrading the DBMS software, backup and recovery management, performance tuning and monitoring, memory management, replication management, and security and authorization. A DBA closely collaborates with network and system managers. He/she also interacts with database designers to reduce operational management costs and guarantee agreed-upon service levels (e.g., response times and throughput rates).

Data scientist is a relatively new job profile within the context of data management. The data scientist analyzes data using state-of-the-art analytical techniques to provide new insights into, e.g., customer behavior. A data scientist has a multidisciplinary profile combining ICT skills (e.g., programming) with quantitative modeling (e.g., statistics), business understanding, communication, and creativity.

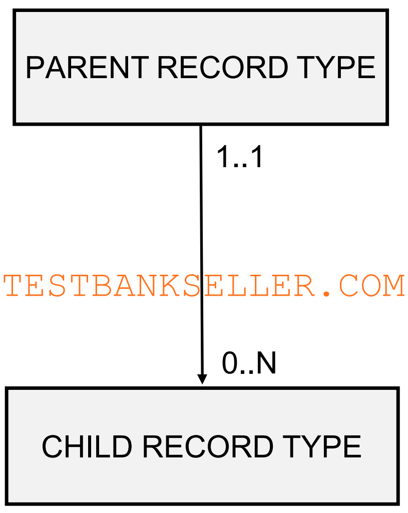
Chapter 5. Legacy Databases

Question 5.1 Contrast the hierarchical model with the CODASYL model in terms of

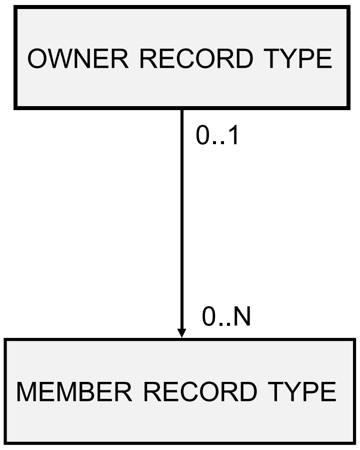
- attribute types supported;
- relationship types and cardinalities supported.

Answer

The two key building blocks of the hierarchical model are record types and relationship types. A record type consists of fields or data items. No support is provided for multivalued or composite attribute types. A relationship type connects two record types. More specifically, it models the relationships between record types. Only hierarchical structures are allowed, or in other words only 1:N relationship types can be modeled, as follows:



The two key building blocks of the CODASYL model are record types and set types. The CODASYL model provides support for both vectors and repeated groups. A vector is a multivalued attribute type. A repeated group is a composite data item for which a record can have multiple values or a composite multivalued attribute type. A set type models a 1:N relationship type between an owner record type and a member record type as follows:

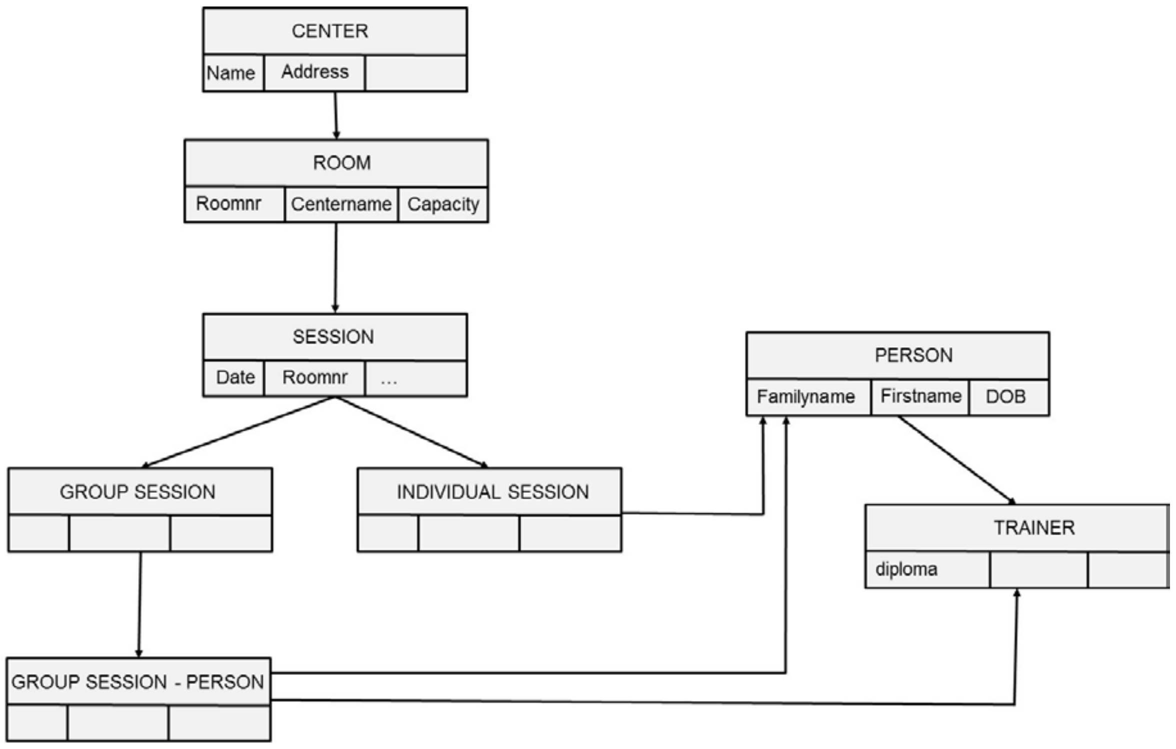


As illustrated, a member record can exist without being connected to an owner record or can be disconnected from its owner. This is a key difference to the hierarchical model. Furthermore, a record type can be a member record type in multiple set types which allows the creation of network structures. In the hierarchical model, a child record type can only be connected to one parent record type. Finally, multiple set types may be defined between the same record types, which is another difference to the hierarchical model.

Question 5.2 Make a hierarchical and CODASYL model for the Fitness company "Conan" discussed in Chapter 3. Contrast both models and discuss their limitations. Give examples of semantics that cannot be enforced. Compare the models with the ER, EER, and UML models of Chapter 3.

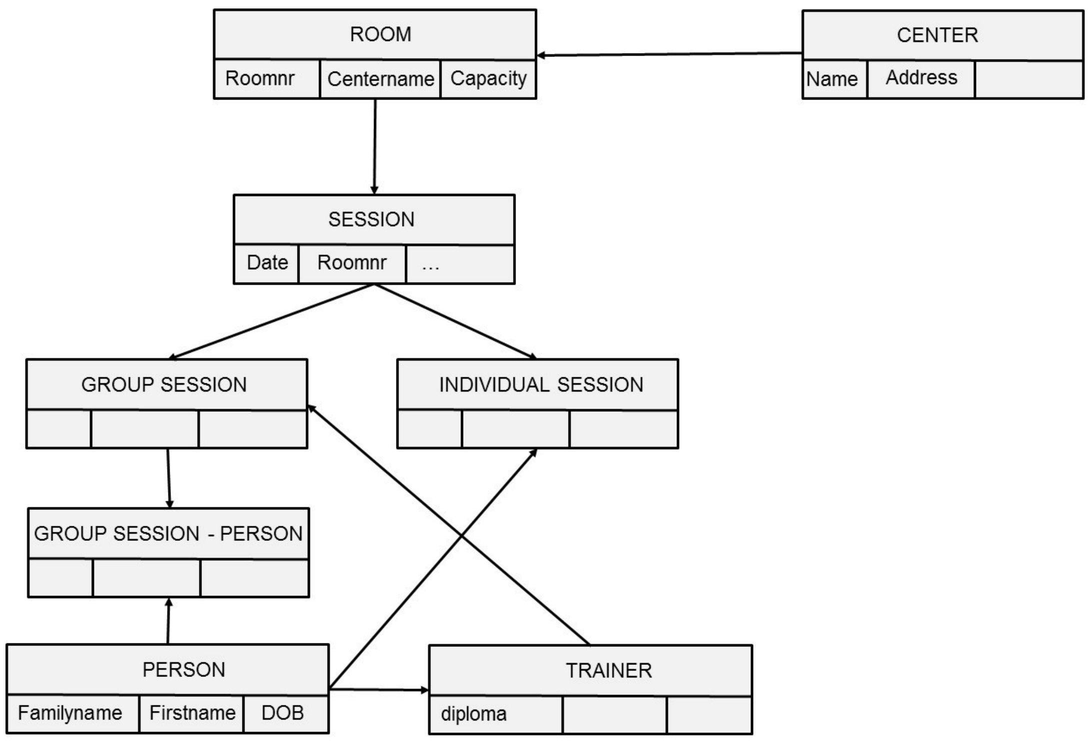
Answer

The hierarchical model for Conan looks as follows:



We used two hierarchical structures: one with CENTER as the root record type and one with PERSON as the root record type. The EER minimum cardinality of 1 from CENTER to ROOM cannot be enforced in the hierarchical model. The 1:N EER relationship type between ROOM and SESSION can be perfectly mapped. We lose semantics when mapping the EER specialization of SESSION into GROUP SESSION and INDIVIDUAL SESSION, since a SESSION parent record can now be connected to multiple GROUP SESSION child records, or multiple INDIVIDUAL SESSION child records. Also, the total and disjoint property of the specialization gets lost. The GROUP SESSION–PERSON record type and INDIVIDUAL SESSION record type are virtual child record types to establish the connection with the other hierarchy. Various semantics gets lost. For example, we cannot enforce that a group session should have exactly one trainer. Also, the EER specialization from PERSON into TRAINER is not perfectly mapped since a PERSON parent record can be connected to multiple trainer child records.

The CODASYL model for Conan looks as follows:



The EER minimum cardinality of 1 from CENTER to ROOM cannot be enforced in the CODASYL model. The 1:N EER relationship type between ROOM and SESSION can be perfectly mapped to the corresponding CODASYL set type. The specialization of SESSION into GROUP SESSION and INDIVIDUAL SESSION cannot be perfectly mapped in the CODASYL model since a specific session owner record can be related to multiple GROUP SESSION member records, or multiple INDIVIDUAL SESSION member records. Also, the total and disjoint property of the specialization gets lost. The partial specialization of PERSON into TRAINER cannot be perfectly mapped, since according to the corresponding CODASYL set type, a PERSON owner record can be related to multiple TRAINER member records. The 1:N EER relationship type between PERSON and INDIVIDUAL SESSION can be perfectly mapped to the corresponding CODASYL set type. Also, the 1:N EER relationship type between TRAINER and GROUP SESSION can be perfectly mapped to the corresponding CODASYL set type. To map the N:M EER relationship type between PERSON and GROUP SESSION, a new dummy record type GROUP SESSION–PERSON needs to be introduced since CODASYL provides no direct support for N:M relationship types. The EER minimum cardinality of 1 (from GROUP SESSION to PERSON) cannot be perfectly mapped in the CODASYL model.

From the above discussion, it becomes clear that a lot of semantics are lost in the hierarchical and CODASYL models when compared to the semantically richer EER or UML models.

Chapter 6. Relational Databases: The Relational Model

Question 6.1 A library database records the authors and the publisher of each book. Normalize the following relation and indicate the primary and foreign key attribute types

R(ISBN, title, author(name, date_of_birth), publisher(name, address(streetnr, streetname, zipcode, city)), pages, price)

The assumptions are:

- each book has a unique ISBN number;
- each author has a unique name;
- each publisher has a unique name;
- a book can have multiple authors;
- an author can write more than one book;
- a publisher can publish more than one book;
- a book has only one publisher;
- a publisher has only one address.

Suppose that one book can have multiple publishers. How can you extend your model to accommodate this? Where would you put the attribute type "number_of_copies"?

Answer

First Normal Form

The first normal form is violated because Author is a multivalued composite attribute type and Publisher is a composite attribute type. The relation R can be brought in first normal form as follows:

BOOK(<u>ISBN</u>, title, publishername, streetnumber, streetname, zipcode, cityname, pages, price)

AUTHOR(<u>*ISBN*</u>, <u>authorname</u>, date_of_birth)

Second Normal Form

The relation AUTHOR is not in second normal form because date_of_birth is partially dependent on ISBN and authorname. In fact, it only depends on authorname. The relation AUTHOR can be brought in second normal form as follows:

BOOK(<u>ISBN</u>, title, publishername, streetnumber, streetname, zipcode, cityname, pages, price)

AUTHOR(<u>authorname</u>, date_of_birth)

WRITES(*<u>authorname</u>*, *<u>ISBN</u>*)

Third Normal Form Step 1

The relation BOOK is not in third normal form since we have a transitive functional dependence from ISBN to streetnumber via publishername, from ISBN to streetname via publishername, from ISBN to zipcode via publishername and from ISBN to cityname via publishername. The relation BOOK can be brought in third normal form as follows:

BOOK(<u>ISBN</u>, title, pages, price, *publishername*)

AUTHOR(<u>authorname</u>, date_of_birth)

WRITES(*<u>authorname</u>*, *<u>ISBN</u>*)

PUBLISHER(<u>publishername</u>, streetnumber, streetname, zipcode, cityname)

Third Normal Form Step 2

The relation BOOK is still not in third normal form since we have a transitive functional dependency from publishername to cityname via zipcode. The relation BOOK can be brought in third normal form as follows:

BOOK(<u>ISBN</u>, title, pages, price, *publishername*)

AUTHOR(<u>authorname</u>, date_of_birth)

WRITES(*<u>authorname</u>*, *<u>ISBN</u>*)

PUBLISHER(<u>publishername</u>, streetnumber, streetname, *zipcode*)

CITY(<u>zipcode</u>, cityname)

If you want to extend the model by allowing a book to be published by multiple publishers, then the relation BOOK is no longer in first normal form since publishername now becomes a multivalued attribute type. The new normalized set of relations then becomes:

BOOK(<u>ISBN</u>, title, pages, price)

AUTHOR(<u>authorname</u>, date_of_birth)

WRITES(*<u>authorname</u>*, *<u>ISBN</u>*)

PUBLISHER(<u>publishername</u>, streetnumber, streetname, *zipcode*)

CITY(<u>zipcode</u>, cityname)

PUBLISHES(*ISBN,publishername*)

The attribute type "number_of_copies" can then be stored in the PUBLISHES relation.

Question 6.2 Given the following assumptions:

- a flight has a unique flight number, a passenger has a unique name, and a pilot has a unique name;
- a flight is always handled by one airline;
- a flight can have multiple passengers and a passenger can do multiple flights;
- a flight has one pilot and a pilot can do multiple flights;
- a flight is always handled by exactly one airplane.

normalize the following relation:

Flight(Flightnumber, Flighttime, airline(airlinename), passenger(passengername, gender, date of birth), pilot(pilotname, gender, date of birth), departure_city, arrival_city, airplane(planeID, type, seats))

Answer

First Normal Form

The relation Flight is not in first normal form since airline is a composite attribute types, passenger is a multivalued composite attribute type, pilot is a composite attribute type, and airplane is a composite attribute type. The relation Flight can be brought in first normal form as follows:

Flight(Flightnumber, Flighttime, airlinename, pilotname, gender, date of birth, departure_city, arrival_city, planeID, type, seats)

Flies(*Flightnumber*, passengername, gender, date of birth)

Second Normal Form

The relation Flies is not in second normal form since gender and date of birth are partially dependent on Flightnumber and passengername. In fact, both gender and date of birth only depend on passengername. The relation Flies can be brought in second normal form as follows:

Flight(Flightnumber, Flighttime, airlinename, pilotname, gender, date of birth, departure_city, arrival_city, planeID, type, seats)

Flies(*Flightnumber*, *passengername*)

Passenger(Passengername, gender, date of birth)

Boyce–Codd Normal Form Step 1

Remember, the Boyce–Codd normal form is more strict than the third normal form. The relation Flight is not in Boyce–Codd normal form since we have the following non-trivial functional dependencies: pilotname → gender and pilotname → date of birth, where pilotname is not a superkey. We can now try to bring the relation Flight in Boyce–Codd normal form as follows:

Flight(<u>Flightnumber</u>, Flighttime, airlinename, *pilotname*, departure_city, arrival_city, planeID, type, seats)

Pilot(<u>pilotname</u>, gender, date of birth)

Flies(<u>*Flightnumber*</u>, <u>*passengername*</u>)

Passenger(<u>Passengername</u>, gender, date of birth)

Boyce–Codd Normal Form Step 2

The relation Flight is still not in Boyce–Codd normal form since we have the following non-trivial functional dependencies: planeID → type and planeID → seats, where planeID is not a superkey. We can bring the relation Flight in Boyce–Codd normal form as follows:

Flight(<u>Flightnumber</u>, Flighttime, airlinename, *pilotname*, departure_city, arrival_city, *planeID*)

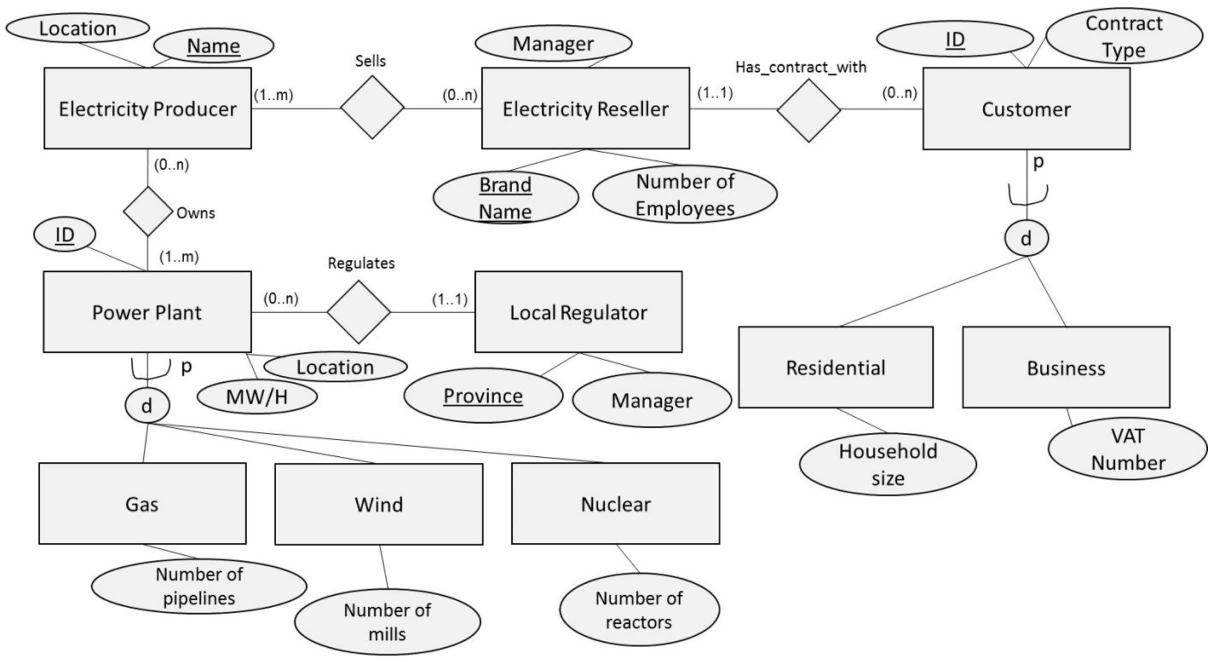
Pilot(<u>pilotname</u>, gender, date of birth)

Airplane(<u>planeID</u>, type, seats)

Flies(<u>*Flightnumber*</u>, <u>*passengername*</u>)

Passenger(<u>Passengername</u>, gender, date of birth)

Question 6.3 Given the following EER model for an electricity market:



- discuss some examples of semantics that cannot be enforced by the EER model;
- map the model to a relational model representation. Discuss the possible loss of semantics. Clearly indicate the primary-foreign key relationships and specify NOT NULL declarations where necessary.

Answer

The following semantics cannot be enforced by the EER model:

- Temporal aspects:
  - A contract between a customer and an electricity reseller could be limited in time.
  - The size of a household could change from year to year.
  - The number of windmills of a power plant could change.
- Consistency across multiple relationship types:
  - The model cannot enforce that the power plant of an electricity producer that is located in a certain province is regulated by the local regulator of that same province.
  - The model cannot enforce that one person is not both the manager of an electricity reseller and of a local regulator at the same time.
- Integrity rules:
  - Household size should not be larger than 15.
  - A VAT number should have a fixed format of ten digits.

- The province of the local regulator should be a valid province of the country we are creating this model for.

- Behavior :
  - A procedure to calculate the current price of electricity.
  - A procedure that checks whether an electricity producer has a permit from a local regulator for each of the power plants they own.

The EER model can be mapped to the following relational model (primary keys are underlined; foreign keys are in italics):

Electricity_Producer(<u>P-Name</u>, Location)

Electricity_Reseller(<u>Brand-Name</u>, Nr-employees, Manager)

Sells(<u>*P-Name*</u>, <u>*Brand-Name*</u>)

> *P-Name*: foreign key refers to P-Name in relation **Electricity_Producer**: NULL not allowed, on delete/update cascade
>
> *Brand-Name*: foreign key refers to Brand-Name in relation **Electricity_Reseller**: NULL not allowed, on delete/update cascade

Local_Regulator(<u>Province</u>, Manager)

Customer(<u>ID</u>, Contract Type, *Contract-With*)

> *Contract-with*: foreign key refers to Brand-Name in relation **Electricity_Reseller**: NULL not allowed, on delete/update cascade

Residential-Customer(<u>*R-C-ID*</u>, Household-size)

> *R-C-ID*: foreign key refers to ID in relation **Customer**: NULL not allowed, on delete/update cascade

Business-Customer(<u>*B-C-ID*</u>, VAT-number)

> *B-C-ID*: foreign key refers to ID in relation **Customer**: NULL not allowed, on delete/update cascade

Power-Plant(<u>ID</u>, MWH, Location, *Regulated*)

> *Regulated*: foreign key refers to Province in relation **Local_Regulator**: NULL not allowed, on delete/update cascade

Owns(<u>*P-P-ID*</u>, <u>*Prod-Name*</u>)

> *P-P-ID*: foreign key refers to ID in relation **Power-Plant**: NULL not allowed, on delete/update cascade
>
> *Prod-Name*: foreign key refers to P-Name in relation **Electricity_Producer**: NULL not allowed, on delete/update cascade

Gas-Power-Plant(<u>*Gas-P-P-ID*</u>, Nr-pipelines)

Gas-P-P-ID: foreign key refers to ID in relation **Power-Plant**: NULL not allowed, on delete/update cascade

Wind-Power-Plant(*Wind-P-P-ID*, Nr-mills)

Wind-P-P-ID: foreign key refers to ID in relation **Power-Plant**: NULL not allowed, on delete/update cascade

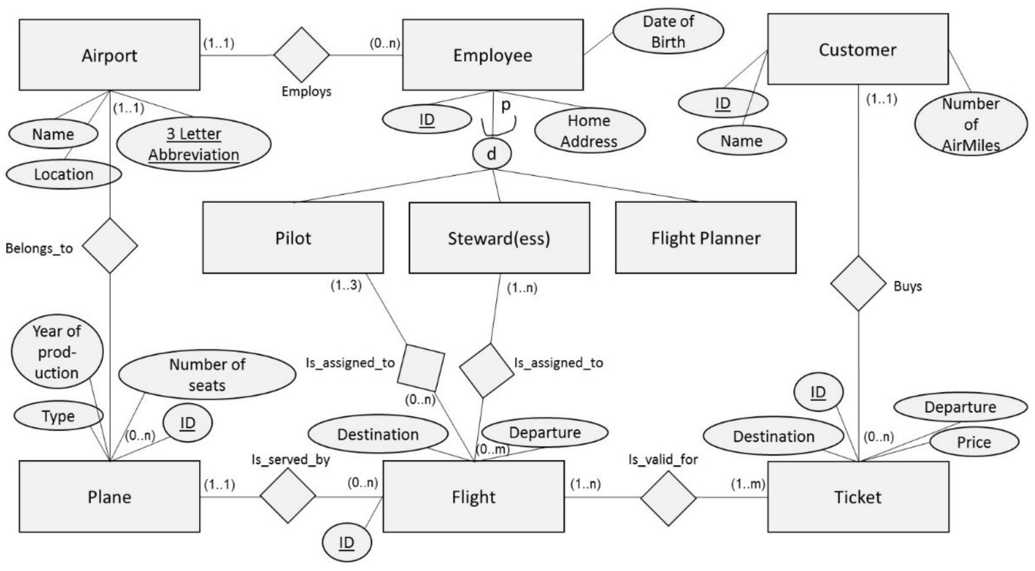
Nuclear-Power-Plant(*Nuclear-P-P-ID*, Nr-reactors)

Nuclear-P-P-ID: foreign key refers to ID in relation **Power-Plant**: NULL not allowed, on delete/update cascade

The following semantics were lost in the mapping:

- the disjointness of the specialization of Power Plant into Gas, Wind, and Nuclear cannot be enforced;
- the disjointness of the specialization of Customer into Residential and Business cannot be enforced;
- the minimum cardinality of 1 in the relationship type Owns between Electricity Producer and Power Plant cannot be enforced;
- the minimum cardinality of 1 in the relationship type Sells between Electricity Producer and Electricity Reseller cannot be enforced.

Question 6.4 Given the following EER model for an airline business:



- discuss some examples of semantics that cannot be enforced by the EER model;

- map the model to a relational model representation. Discuss the possible loss of semantics. Clearly indicate the primary–foreign key relationships and specify NOT NULL declarations where necessary.

Answer

The following semantics cannot be enforced by the EER model:

- Temporal aspects:
  - A pilot or steward(ess) should only be assigned to a flight that takes place on a future date.
  - A customer should not be able to buy tickets for a flight that has already taken place.
  - The home address of an employee can change, which has an effect on which airport the employee should be employed at.
- Consistency across multiple relationship types:
  - There can only be sold as many tickets as the total number of seats on a plane.
  - A pilot or steward(ess) should not be able to buy a ticket for a flight to which they are assigned to work on themselves.
- Integrity rules:
  - The year of production of a plane should be >1980.
  - The number of air-miles of a customer should be a positive integer.
- Behavior:
  - A procedure for calculating the ticket price of a flight, which takes into account how many seats are still available on the plane.
  - A procedure for calculating the number of air-miles that need to be added to a customer's account, taking into account the flight distance of the ticket the customer has bought.

The EER model can be mapped to the following relational model (primary keys are underlined; foreign keys are in italics):

Airport(<u>3-Letter-Abbreviation</u>, Location, Name)

Plane(<u>Plane-ID</u>, Nr-seats, Type, Year-of-production, *Home-Airport*)

Home-Airport: foreign key refers to 3-Letter-Abbreviation in relation **Airport**: NULL not allowed, on delete/update cascade

Flight(<u>Flight-ID</u>, Departure, Destination, *Plane*)

Plane: foreign key refers to Plane-ID in relation **Plane**: NULL not allowed, on delete/update cascade

Employee(Employee-ID, Home-Address, Date-of-birth, *Employer*)

Employer: foreign key refers to 3-Letter-Abbreviation in relation **Airport**: NULL not allowed, on delete/update cascade

Pilot(*Pilot-ID*)

Pilot-ID: foreign key refers to Employee-ID in relation **Employee**: NULL not allowed, on delete/update cascade

Pilot-Assignment(*Pilot-ID*, *Flight-ID*)

Pilot-ID: foreign key refers to Pilot-ID in relation **Pilot**: NULL not allowed, on delete/update cascade

Flight-ID: foreign key refers to Flight-ID in relation **Flight**: NULL not allowed, on delete/update cascade

Steward(ess)(*Steward(ess)-ID*)

Steward(ess)-ID: foreign key refers to ID in relation **Employee**: NULL not allowed, on delete/update cascade

Steward(ess)-Assignment(*Steward(ess)-ID*, *Flight-ID*)

Steward(ess)-ID: foreign key refers to Steward(ess)-ID in relation **Steward(ess)**: NULL not allowed, on delete/update cascade

Flight-ID: foreign key refers to Flight-ID in relation **Flight**: NULL not allowed, on delete/update cascade

Flight-Planner(*Flight-Planner-ID*)

Flight-Planner-ID: foreign key refers to ID in relation **Employee**: NULL not allowed, on delete/update cascade

Customer(Customer-ID, Name, Nr-Airmiles)

Ticket(Ticket-ID, Departure, Destination, Price, *Buyer*)

Buyer: foreign key refers to Customer-ID in relation **Customer**: NULL not allowed, on delete/update cascade

Valid-Flight-Ticket(*Ticket-ID*, *Flight-ID*)

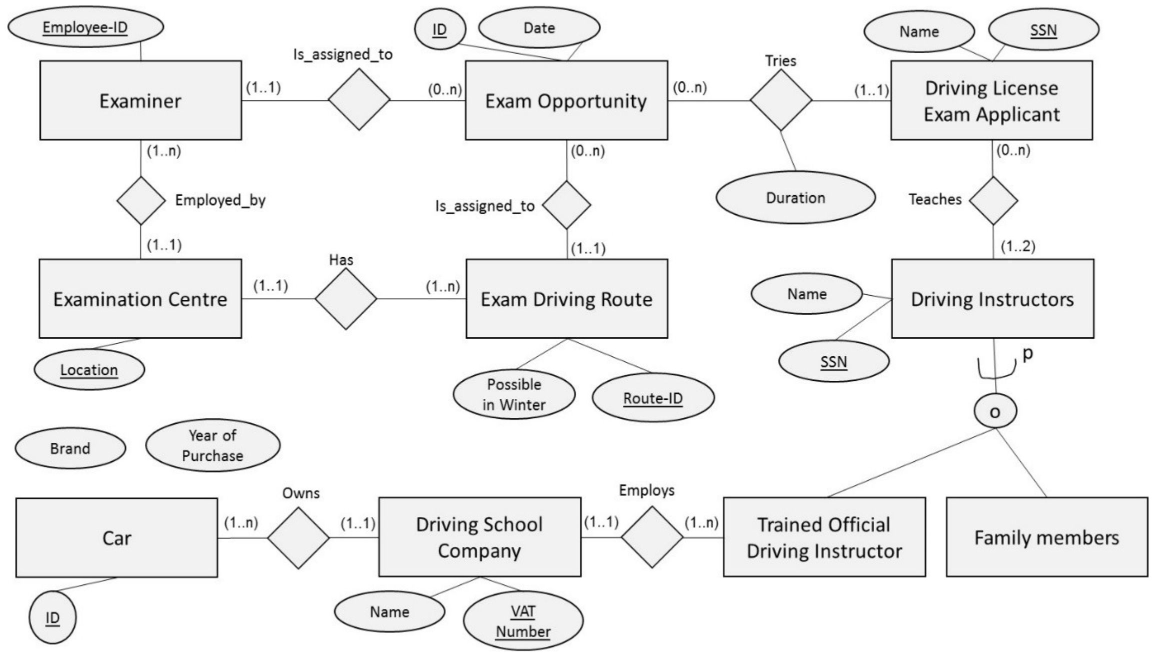
Ticket-ID: foreign key refers to Ticket-ID in relation **Ticket**: NULL not allowed, on delete/update cascade

Flight-ID: foreign key refers to Flight-ID in relation **Flight**: NULL not allowed, on delete/update cascade

The following semantics were lost in the mapping:

- The disjointness of the Employee specialization is not enforceable. It is hence possible that an employee is at the same time a pilot, steward(ess), and a flight planner.
- The minimum cardinalities of 1 in the relationship types is_assigned_to (between Pilot and Flight), is_assigned_to between Steward(ess) and Flight and is_valid_for between Flight and Ticket cannot be enforced.
- The maximum cardinality of 3 in the relationship type is_assigned_to between Pilot and Flight cannot be enforced.

Question 6.5 Given the following EER model for a driving school company:



- discuss some examples of semantics that cannot be enforced by the EER model;
- map the model to a relational model representation. Discuss the possible loss of semantics. Clearly indicate the primary–foreign key relationships and specify NOT NULL declarations where necessary.

Answer

The following semantics cannot be enforced by the EER model:

- Temporal aspects:
  - An examiner should be assigned to an exam opportunity the day before the date of the exam opportunity at the latest.
  - An applicant can only take another exam opportunity for the third time after two failures, if at least three months have passed since the date of the previous exam opportunity.
  - It should not be possible to choose certain exam driving routes during the winter.
- Consistency across multiple relationship types:
  - It should not be possible for an examiner to be the same person as the driving instructor of the exam applicant.
  - An exam applicant cannot be an examiner or a driving instructor.
  - Each examination center should only have those exam driving routes available that are near their location.
- Integrity rules:
  - The duration of one exam opportunity should be a positive integer in minutes.
  - A route ID should be a number with three digits.
- Behavior:
  - A procedure to send an examiner an email when he/she is assigned to an exam opportunity.
  - A procedure that calculates the average number of tries an applicant takes before they successfully pass the exam.

The EER model can be mapped to the following relational model (primary keys are underlined; foreign keys are in italics):

Examination_Centre(<u>Location</u>)

Examiner(<u>Employee-ID</u>, *Work-Location*)

Work-Location: foreign key refers to Location in relation **Examination_Centre**: NULL not allowed, on delete/update cascade

Exam_Driving_Route(<u>Route-ID</u>, Possible-in-Winter, *Start-Location*)

Start-Location: foreign key refers to Location in relation **Examination_Centre**: NULL not allowed, on delete/update cascade

Driving_License_Exam_Applicant(<u>SSN</u>, Name)

Exam_Opportunity(Exam-ID, Exam-Date, *Route*, *Examiner*, *Applicant*, Duration)

Route: foreign key refers to Route-ID in relation **Exam_Driving_Route**: NULL not allowed, on delete/update cascade

Examiner: foreign key refers to Employee-ID in relation **Examiner**: NULL not allowed, on delete/update cascade

Applicant: foreign key refers to SSN in relation **Driving_License_Exam_Applicant**: NULL not allowed, on delete/update cascade

Driving_School_Company(VAT-number, Name)

Car(ID, Brand, Year-of-Purchase, *Owner*)

Owner: foreign key refers to VAT-number in relation **Driving_School_Company**: NULL not allowed, on delete/update cascade

Driving_Instructors(Instructor-SSN, Name)

Trained_Official_Driving_Instructor(*I-SSN*, *Employer*)

I-SSN: foreign key refers to Instructor-SSN in relation **Driving_Instructors**: NULL not allowed, on delete/update cascade

Employer: foreign key refers to VAT-number in relation **Driving_School_Company**: NULL not allowed, on delete/update cascade

Family_Members(*I-SSN*)

I-SSN: foreign key refers to Instructor-SSN in relation **Driving_Instructors**: NULL not allowed, on delete/update cascade

Teaches(*Student-SSN*, *Teacher-SSN*)

Teacher-SSN: foreign key refers to Instructor-SSN in relation **Driving_Instructors**: NULL not allowed, on delete/update cascade

Student-SSN: foreign key refers to SSN in relation **Driving_License_Exam_Applicant**: NULL not allowed, on delete/update cascade

The following semantics were lost in the mapping:

- The specialization of driving instructors into trained official driving instructor and family members is perfectly mapped!

- We cannot enforce that the attribute type “Possible-in-Winter” should have a Boolean data type (i.e., true or false).
- We cannot enforce the maximum cardinality of 2 in the relationship type Teaches between Driving License Exam Applicant and Driving Instructors.
- We cannot enforce the minimum cardinality of 1 in the relationship Teaches between Driving License Exam Applicant and Driving Instructors.
- We cannot enforce that an examination center should have at least one examiner employed.
- We cannot enforce that an examination center should have at least one exam driving route.
- We cannot enforce that a driving school company should own at least one car.
- We cannot enforce that a driving school company should employ at least one trained official driving instructor.

Chapter 7. Relational Databases: Structured Query Language (SQL)

Note that all the below queries can be tested using our online environment.

Question 7.1 Write an SQL query that retrieves each supplier who can deliver product 0468 within one or two days, accompanied by the price of the product and the delivery period.

Answer

```
SELECT SUPNR, PURCHASE_PRICE, DELIV_PERIOD

FROM SUPPLIES

WHERE DELIV_PERIOD ≤ 2 AND PRODNR='0468'
```

Question 7.2 Write an SQL query that returns the average price and variance per product.

Answer

```
SELECT PRODNR, AVG(PURCHASE_PRICE) AS AVG_PRICE, VARIANCE(PURCHASE_PRICE) AS
VAR_PRICE

FROM SUPPLIES

GROUP BY PRODNR
```

Question 7.3 Write an SQL query that retrieves all pairs of suppliers who supply the same product, along with their product purchase price if applicable.

Answer

```
SELECT S1.PRODNR, S1.SUPNR, S1.PURCHASE_PRICE, S2.SUPNR, S2.PURCHASE_PRICE
FROM SUPPLIES S1, SUPPLIES S2
WHERE (S1.PRODNR = S2.PRODNR)
AND (S1.SUPNR < S2.SUPNR)
```

Question 7.4 Write a nested SQL query to retrieve the supplier name of each supplier who supplies more than five products.

Answer

```
SELECT SUPNAME
```

```
FROM SUPPLIER
WHERE SUPNR IN
    (SELECT SUPNR
    FROM SUPPLIES
    GROUP BY SUPNR
    HAVING COUNT(*) > 5)
```

Question 7.5 Write an SQL query to retrieve the supplier number, supplier name, and supplier status of each supplier who has a higher supplier status than supplier number 21.

Answer

```
SELECT SUPNR, SUPNAME, SUPSTATUS
FROM SUPPLIER
WHERE SUPSTATUS >
    (SELECT SUPSTATUS
    FROM SUPPLIER
    WHERE SUPNR = '21')
```

Question 7.6 Write a correlated SQL query to retrieve the number and status of all suppliers, except for the three suppliers with the lowest supplier status.

Answer

```
SELECT R1.SUPNR, R1.SUPSTATUS
FROM SUPPLIER R1
WHERE R1.SUPSTATUS IS NOT NULL
AND 2 <
    (SELECT COUNT(*)
    FROM SUPPLIER R2
    WHERE R2.SUPSTATUS < R1.SUPSTATUS)
```

Question 7.7 Write a correlated SQL query to retrieve all cities with more than one supplier.

Answer

```
SELECT DISTINCT R1.SUPCITY
FROM SUPPLIER R1
WHERE R1.SUPCITY IN
    (SELECT R2.SUPCITY
    FROM SUPPLIER R2
    WHERE R2.SUPNR <> R1.SUPNR)
```

Question 7.8 Write an SQL query to retrieve the name, number, and total outstanding orders of all suppliers that have outstanding orders, except for the supplier(s) with the least outstanding orders.

Answer

```
SELECT R1.SUPNAME, R1.SUPNR, COUNT(*)
FROM PURCHASE_ORDER PO1, SUPPLIER R1
WHERE PO1.SUPNR = R1.SUPNR
GROUP BY R1.SUPNR
HAVING COUNT(*) > ANY
    (SELECT COUNT(*)
    FROM PURCHASE_ORDER PO2, SUPPLIER R2
    WHERE PO2.SUPNR = R2.SUPNR
    GROUP BY R2.SUPNR)
```

Question 7.9 Write an SQL query using EXISTS to retrieve the supplier number and name of all suppliers that do not have any outstanding orders.

Answer

```
SELECT R.SUPNR, R.SUPNAME
```

```
FROM SUPPLIER R
WHERE NOT EXISTS
    (SELECT *
    FROM PURCHASE_ORDER PO
    WHERE PO.SUPNR = R.SUPNR)
```

Question 7.10 Create a view, SUPPLIEROVERVIEW, which retrieves for each supplier the supplier number, the supplier name, and the total quantities ordered. Once created, query this view to retrieve suppliers for which the total ordered quantity exceeds 30.

Answer

```
CREATE VIEW SUPPLIEROVERVIEW AS
SELECT S.SUPNR, S.SUPNAME, SUM(POL.QUANTITY) AS TOTQUANTITY
FROM
SUPPLIER AS S, PURCHASE_ORDER AS P, PO_LINE POL
WHERE
S.SUPNR=P.SUPNR AND
P.PONR=POL.PONR
GROUP BY S.SUPNR


SELECT *
FROM SUPPLIEROVERVIEW
WHERE TOTQUANTITY > 30
```

Question 7.11 Write an SQL query that retrieves the total available quantity of sparkling wines. Make sure to display this quantity as 'TOTAL_QUANTITY'.

Answer

```
SELECT SUM(AVAILABLE_QUANTITY) AS TOTAL_QUANTITY
FROM PRODUCT
WHERE PRODTYPE = 'sparkling'
```

Question 7.12 Write a query to select all supplier numbers, together with their supplier name and total number of outstanding orders for each supplier. Include all suppliers in the result, even if there are no outstanding orders for that supplier at the moment.

Answer

```
SELECT S.SUPNR, S.SUPNAME, COUNT(PO.PONR) AS OUTSTANDING_ORDERS
FROM SUPPLIER AS S LEFT OUTER JOIN PURCHASE_ORDER AS PO
    ON (S.SUPNR = PO.SUPNR)
    GROUP BY S.SUPNR
```

Question 7.13 Write an SQL query that returns the SUPNR and number of products of each supplier that supplies more than five products.

Answer

```
SELECT SUPNR, COUNT(*) AS TOTAL_PRODUCTS

FROM SUPPLIES

GROUP BY SUPNR

HAVING COUNT(*) > 5
```

Question 7.14 Write an SQL query that which reports the average delivery time for each supplier who supplies products.

Answer

```
SELECT S.SUPNR, AVG(S.DELIV_PERIOD) AS AVG_DELIV_PERIOD
FROM SUPPLIES S
GROUP BY S.SUPNR
```

Question 7.15 Write a nested SQL query to retrieve all purchase order numbers of purchase orders that contain either sparkling or red wine.

Answer

```
SELECT DISTINCT PONR

FROM PO_LINE
```

WHERE PONR IN (**SELECT** PONR **FROM** PO_LINE **WHERE** PRODNR IN (**SELECT** PRODNR F**ROM** PRODUCT **WHERE** PRODTYPE='sparkling'))

AND PONR **IN** (**SELECT** PONR from PO_LINE **WHERE** PRODNR **IN** (**SELECT** PRODNR F**ROM** PRODUCT **WHERE** PRODTYPE='red'))

Question 7.16 Write a correlated SQL query to retrieve the three lowest product numbers.

Answer

```
SELECT  P1.PRODNR

FROM PRODUCT P1

WHERE 3 >

    (SELECT   COUNT(*)

    FROM PRODUCT P2

    WHERE P2.PRODNR < P1.PRODNR)
```

Question 7.17 Write an SQL query with ALL or ANY to retrieve the name of the product with the highest available quantity.

Answer

```
SELECT P1.PRODNAME

FROM PRODUCT P1

WHERE P1.AVAILABLE_QUANTITY >= ALL

    (SELECT P2.AVAILABLE_QUANTITY

    FROM PRODUCT P2)
```

Question 7.18 Write an SQL query using EXISTS to retrieve the supplier name and number of the supplier who has the lowest supplier number.

Answer

```
SELECT R1.SUPNR, R1.SUPNAME

FROM SUPPLIER R1
```

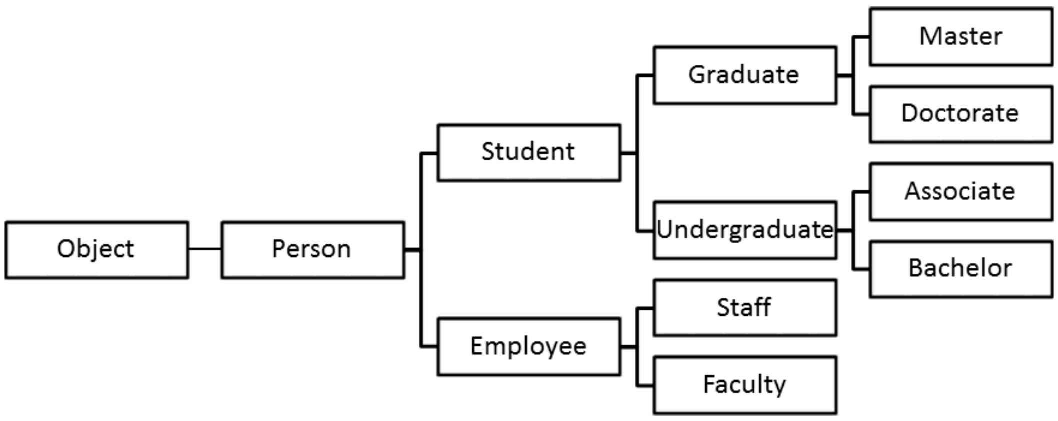
```
WHERE NOT EXISTS
    (SELECT *
    FROM SUPPLIER R2
    WHERE R1.SUPNR > R2.SUPNR)
```

Chapter 8. Object-Oriented Databases and Object Persistence

Question 8.1 What is the relationship between polymorphism and dynamic binding? Illustrate with an example.

Answer

Polymorphism refers to the ability of objects to respond differently to the same method. It is a key concept in object-oriented (OO) programming and is closely related to inheritance. Because inheritance models an “is a” relationship, one object can take on the variables and behavior of more than one class. Consider the following example of a class hierarchy.



According to the class hierarchy of the Person example, a Master is a Graduate, a Graduate is a Student, and a Student is a Person, so depending on the functionality desired, the OO environment might consider a particular Master object as a Master, a Graduate, a Student, or a Person, because, after all, a Master is still a Person.

Every method must be bound or mapped to its implementation. There are two types of binding. Static binding binds a method to its implementation at compile time. In contrast to static binding, dynamic binding means that the binding of methods to the appropriate implementation is resolved at runtime, based on the object and its class. It is also called virtual method invocation and is the binding used for methods to allow for polymorphism. When a method is overridden, multiple implementations can be called, depending on the object in question. During execution, the OO environment will first check the object class that the reference object points to the method for implementation. If it doesn’t exist, the system will look to the superclass. It will check the superclass above that if it still finds no match, and so on throughout the entire class hierarchy. By searching from the bottom up, the most specific implementation, or the one from the class lowest in the hierarchy, will be the one used.

Consider the example of the GPA calculation, which calculates the average of all grades for most Student objects and the average of grades above a threshold for graduate students. Suppose you create a PersonProgram class to run your main method:

```
public class PersonProgram {
public static void main(String[] args){
Student john = new Master("John Adams");
john.setGrades(0.75,0.82,0.91,0.69,0.79);
Student anne = new Associate("Anne Philips");
anne.setGrades(0.75,0.82,0.91,0.69,0.79);
System.out.println(john.getName() + ": " + john.calculateGPA());
System.out.println(anne.getName() + ": " + anne.calculateGPA());
}
}
```

You have two Student objects: John is a Master (subclass of Graduate) and Anne is an Associate (subclass of Undergraduate). To compare easily, assume they both have the same grades for five courses. When your main method reaches the print statements, it will first call the `getName()` method for each object. For John, none of the Master class, the Graduate class, or the Student class contain a `getName()` method. Therefore, the Master object inherits the `getName()` method directly from the Person class. Next, it must call the `calculateGPA()` method for John. The Master class does not contain a `calculateGPA()` method, but the Graduate class, the superclass of Master, does. Dynamic binding looks at the type of object that John is: a Master and, hence, a Graduate. Therefore, the `calculateGPA()` method from the Graduate class is called. For Anne, the same decision process occurs. There is no `getName()` method in the subclasses, so the class hierarchy is considered to find that Anne is an Associate, which is an Undergraduate, which is a Student, which is a Person, and the `getName()` method from the Person class is called. For her GPA, neither Associate nor Undergraduate has a `calculcateGPA()` method, so the Student version of the method is called for Anne. The output will then be as follows:

```
John Adams: 0.865
Anne Philips: 0.792
```

Since John is a master student, which is a subclass of Graduate, only the marks above 0.8 will be averaged so the result will be (0.82 + 0.91)/2 = 0.865. Since Anne is an associate, which is a subclass of Undergraduate, her average will be calculated as (0.75 + 0.82 + 0.91 + 0.69 + 0.79)/5 = 0.792. It shows two different GPAs, despite having the same grades, because dynamic method invocation allows different versions of the methods to be called depending on the object calling it.

Question 8.2 Discuss various strategies to ensure object persistence. When would you use what strategy? What are the key properties a persistence environment should support?

Answer

Various strategies can ensure object persistence. Persistence by class implies that all objects of a particular class will be made persistent. Although this method is simple, it is inflexible because it does not allow a class to have any transient objects. Persistence by creation is achieved by extending the syntax for creating objects to indicate at compile time that an object should be made persistent. Persistence by marking implies that all objects will be created as transient. An object can then be marked as persistent during program execution (at runtime). Persistence by inheritance indicates that the persistence capabilities are inherited from a predefined persistent class. Finally, persistence by reachability starts by declaring the root persistent object(s). All objects referred to (either directly or indirectly) by the root object(s) will then be made persistent as well. This strategy is adopted by the Object Database Management Group (ODMG) standard.

Ideally, a persistence environment should support persistence orthogonality, which implies the following properties: persistence independence, type orthogonality, and transitive persistence. With persistence independence, the persistence of an object is independent of how a program manipulates it. The same code fragment or function can be used with both persistent and transient objects. Hence, the programmer does not need to explicitly control the movement of objects between main memory and secondary storage as this is automatically taken care of by the system. Type orthogonality ensures that all objects can be made persistent, irrespective of their type or size. This prevents the programmer from having to write customized persistence routines and getting sidetracked. Finally, transitive persistence refers to persistence by reachability, as we discussed above. An environment supporting persistence orthogonality allows for improved programmer productivity and maintenance.

Question 8.3 What are object identifiers? Why are they used by OODBMSs? What is the difference with primary keys in an RDBMS?

Answer

In an OODBMS, every object has a unique and immutable object identifier (OID), which it keeps during its entire lifetime. These OIDs are used to uniquely identify objects. No two objects can have the same identifier, and each object has only one identifier. An OID differs from a primary key in a relational database setting. A primary key depends on the state of a tuple, whereas an OID is not dependent on an object's state. A primary key can be updated, whereas an OID never changes. A primary key has a unique value for each tuple within a relation, while an OID is unique within the entire OO environment.

In most commercial implementations, the OIDs will remain invisible to the user. They are typically system-generated. The OIDs are used to identify objects and to create and manage references between objects. The references can implement relationship types, as discussed earlier. By using these OIDs, it becomes possible that an object is shared by various other objects by simply referring to its OID. This reduces the number of updates when objects have their variable values or state changed. Because of the extensive use of these OIDs, the OO model is often called an identity-based model. The relational model, on the other hand, is a value-based model since it uses primary and foreign keys, which are based on actual data values.

Question 8.4 What are literals in the ODMG model? What types of literals are supported? How are literals mapped in Java? Illustrate with examples.

Answer

Contrary to an object, a literal has no OID and cannot exist on its own. It represents a constant value and is typically embedded in an object. Three types of literals are supported: atomic, collection, and structured literals. Examples of atomic literals are short (short integer), long (long integer), double (real number), float (real number), Boolean (true or false), char, and string. A collection literal can model a collection of elements. ODMG defines the following collection literals:

- Set: unordered collection of elements without duplicates.
- Bag: unordered collection of elements which may contain duplicates.
- List: ordered collection of elements.
- Array: ordered collection of elements that is indexed.
- Dictionary: unordered sequence of key–value pairs without duplicates.

A structured literal consists of a fixed number of named elements. ODMG has predefined these structured literals: Date, Interval, Time, and TimeStamp. User-defined structures are also supported, such as Address:

```
struct Address{
string street;
integer number;
integer zipcode;
string city;
string state;
string country;
};
```

The ODMG atomic literal types map into their respective Java primitive types. There are no structured literal types in the Java binding. The object model definition of a structure maps into a Java class. The Java binding also includes several collection interfaces (`DMap`, `DSet`), which extend their respective Java counterparts.

Question 8.5 What types of relationships and cardinalities are supported in ODMG?

Answer

Relationships can be defined in ODMG using the keyword `relationship`. Only unary and binary relationships with cardinalities of 1:1, 1:N, or N:M are supported in ODMG. Ternary (or higher) relationships and relationship attributes must be decomposed by introducing extra classes and relationships. Every relationship is defined in a bidirectional way, using the keyword `inverse` through which both directions have names assigned to facilitate navigation.

Question 8.6 Contrast Object Query Language (OQL) with Structured Query Language (SQL).

Answer

OQL is a declarative, non-procedural query language. It is based upon the SQL SELECT syntax combined with OO facilities, such as dealing with object identity, complex objects, path expressions, operation invocation, polymorphism, and dynamic binding. It can be used for both navigational (procedural) as well as associative (declarative) access. Note that navigational (procedural) access is not supported in SQL. Just as in SQL, multiple classes can be joined. This can be accomplished by traversing the paths as defined in the ODL schema. Like SQL, OQL also provides support for subqueries, GROUP BY/HAVING, and aggregate operators such as COUNT, SUM, AVG, MAX, and MIN. The EXISTS operator can also be used. The OQL language provides no explicit support for INSERT, UPDATE, and DELETE operations. These have to be directly implemented in the class definitions instead.

Question 8.7 Explain the following query in detail.

```
SELECT e1.ENAME, e1.age, d.DNAME, e2.ENAME, e2.age
FROM employees e1, e1.works_in d, d.managed_by e2
WHERE e1.age > e2.age
```

How would you solve a similar query in SQL?

Answer

This query selects the name and age of all employees with a younger manager, the name of the department they work in, and the name and age of their manager. It uses multiple traversal paths. In SQL, this query would look as follows:

```
SELECT E1.ENAME, E1.AGE, D.DNAME, E2.ENAME, E2.AGE
FROM EMPLOYEE E1, DEPARTMENT D, EMPLOYEE E2
E1.DNR=D.DNR AND
D.MGNR=E2.SSN AND
E1.AGE > E2.AGE
```

Question 8.8 Contrast OODBMSs versus RDBMSs in terms of:

- handling of complex objects;
- query performance;
- implementation of the three-layer database architecture;
- adoption in industry.

Answer

Compared to RDBMSs, OODBMSs store complex objects and relationships in a transparent way. The identity-based approach allows for improved performance when performing complex queries involving multiple interrelated objects, avoiding expensive joins. By using the same data model as the programming language to develop database applications, the impedance mismatch problem is no longer an issue. In addition, developers can be more productive, as they are confronted with only a single language and data model.

Still, the success of OODBMSs has been limited to niche applications characterized by complex, nested data structures in which an identity-based, instead of value-based, method of working pays off. An example is the processing of scientific datasets by CERN in Switzerland, where data access follows predictable patterns. The widespread use and performance of RDBMSs, however, proved hard to displace: the (ad-hoc) query formulation and optimization procedures of OODBMSs are often inferior to relational databases, which all adopt SQL as their primary database language combined with a powerful query optimizer. When compared to RDBMSs, OODBMSs are not very developed in terms of robustness, security, scalability, and fault-tolerance. They also don't provide a transparent implementation of the three-layer database architecture. More specifically, most OODBMSs provide no support for defining external database models, such as views in the relational model.

In addition, despite efforts made by the ODMG, the uniform standards proposed were not widely implemented by vendors, who were quick to realize that the concepts of an OODBMS and object persistence as such are not the same, hence the name change from ODMG to the Object Data Management Group (still abbreviated ODMG) in 1998. The aim was to focus on the standardization of an API for object persistence, still according to the same ODMG principles, regardless of whether the underlying data store was an RDBMS or an OODBMS. In hindsight, this is probably the most important contribution of the ODMG, even though the standard itself was abandoned over time.