**1. Database Theory**

a) Question: What are the three levels of abstraction? What is data independence and how is it related to data abstraction?

Answer: Physical level (Internal level): this is the lowest level of data abstraction. It describes how the data is actually stored in the database, including the complex data structures, indexing methods, and the actual storage mechanism on the disk.

Logical level (Conceptual level): This level of abstraction describes what data is stored in the database and the relationships among those data. At this level, the schema of the database is defined. It provides a community user view of the database, describing the entire database in terms of a small number of relatively simple structures.

View level (External level): This is the highest level of data abstraction. It describes only the top part of the entire database. It focuses on how the data is viewed and manipulated by end users. This level also makes sure that users only access data relevant to their task.

Data abstraction is the procedure of hiding or concealing irrelevant or unwanted data from the end user.

Data independence is mainly defined as a *property* of DBMS that helps you change the database schema at one level of the system without requiring to change the schema at the next level. It helps to keep the data separated from all programs that make use of it.

b) Question: What is 1-tier, 2-tier, and 3-tier architecture? What is the difference between 2-tier and 3-tier architecture?

Answer: A 1-tier architecture, all elements of an application reside on the same machine or platform. This is the simplest form of application architecture. Example: MS Office. In a 2-tier architecture, the application is divided into two parts: a client application, which deals with the user side of it, and a server, which deals with database management and storage. The client interacts directly with the server. Example: A desktop application that connects to something like an SQL Server. In a 3-tier architecture, another layer is added, resulting in three layers, client layer, application layer, and data storage layer. The difference between a 3-tier and a 2-tier architecture is as follows: 3-tier is more complex but offers more scalability. 2-tier is simpler and more cost-efficient.

c) Question: What is conceptual model and which concepts does it contain?

Answer: A conceptual model is a high-level representation of a system’s structure and behavior, without going into details of implementation. Its concepts are, entities (main objects or concepts), attributes (properties/characteristics), relationships (how entities relate to each other) and constraints (rules/limitations on the data).

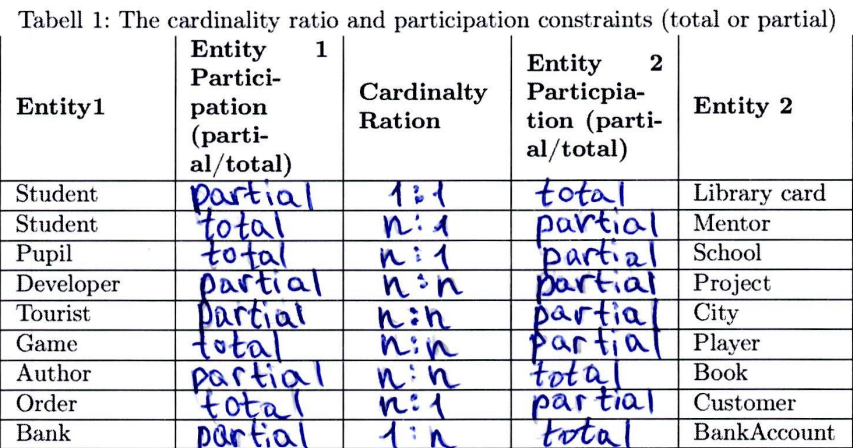
d) Question: Why should an entity have a key? Can an entity not have a key?

Answer: An entity having a unique key is crucial in database security. It helps identify entities uniquely, avoiding crashes between objects with the same attributes. It also plays a big role in defining relationships between different entities (using foreign and primary keys for example).

**2. Entity Relation**

a) The cardinality ratio depends on the real-world meaning of the entity types involved and is defined by the specific application. For the binary relationships below (Tabell 1), suggest cardinality ratios based on common-sense meaning of the entity types. Also determine participation constraints: total or partial.

Answer and explanation: Total participation means that entity needs to participate in a relationship to exist. For example, pupils must exist for school to exist. Partial participation means that an entity can exist, but also doesn’t need to.. For example, a customer can exist without placing an order.



b) Question: Consider the ER schema for the hospital database shown in Figure 1, respond to the following statements with True or False.

A diagram of a patient

Description automatically generated

Answer (see 2b. for more information on how to solve the problem):

A list of medical personnel

Description automatically generated

c) A database is being constructed to keep track of the teams and games of a sport league. A team has a number of players, not all of whom participate in each game. The database keeps track of the players participating in each game for each team, the positions/roles they played in that game, and the result of the game. Design Entity-Relation (ER) diagram for this application and specify the cardinality ratio.

Answer (note that this gave full points, i.e. it’s not that hard):  
Rectangle: Entity  
Oval: Attribute  
Diamond: Relationship  
Lines: Connects entities with relationships/attributes  
*Note that there are more components to learn*

A diagram of a game

Description automatically generated

**3. Functional Dependencies and Normalization**

a) Question: What are functional dependencies?

Answer: In relational database management, functional dependency is a concept that specifies the relationship between two sets of attributes where one attribute determines the value of another attribute. It is denoted by X → Y, where the left attribute (X) is called **Determinant**, and the right attribute (Y) is called the **Dependent**. You can think of this as X decides Y. Functional dependencies are used to mathematically express relations among database entities. An example of this can be found below:

A screenshot of a computer

Description automatically generated

Here, the Employee ID is the Determinant of both Employee Name and Department. We write this as Employee ID → Employee Name, Department. Note that Employee Id is also a primary key. Primary keys naturally create functional dependencies, but a functional dependency can occur between non-primary key attributes.

Let’s go over the three types of functional dependencies:

**Full functional dependency:** This occurs when an attribute is functionally dependent on a whole composite key (a key in which multiple columns in a table are used to create a unique identifier for each row), and not on any subset of that key. This means that the attribute cannot be determined by only part of the key, it requires the entire key.

**Partial dependency:** This means that an attribute is functionally dependent on part of a composite key rather than on the entire composite key. This usually leads to redundancy and various forms of anomalies in your database.

**Transitive dependency:** This occurs when an attribute is functionally dependent on another non-key attribute, which in turn is functionally dependent on a key attribute. In simpler terms, if A determines B, and B determines C, then C has transitive dependency through B on A. This can also lead to redundancy anomalies in your database.

**In Summary:**

**Full Functional Dependency:** An attribute depends on the whole composite key.

**Partial Dependency:** An attribute depends on just part of a composite key, leading to redundancy.

**Transitive Dependency:** An attribute depends on another attribute that itself depends on a key, creating a chain of dependencies.

b) Question: What undesirable dependencies are avoided when a relation is in 2NF?

Answer: 0NF, or Unnormalized form is just a table where there’s almost no rules, and where columns can repeat all they want (very inefficient and bad). 1NF just deletes repeating groups.   
  
2NF (second normal form) is based on the concept of fully functional dependency. Here’s the definition for 2NF: A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key, then the relation is in Second Normal Form (2NF). If we put this into simpler terms, A relation is in 2NF if it is already in 1NF and has no partial dependency. The fact that it is in 1NF and additionally has no partial dependency is **literally** the definition of 2NF. So obviously, it avoids partial dependency.

c) Question: What undesirable dependencies are avoided when a relation is in 3NF?

Answer: To be in 3NF, a relation must first satisfy the Second Normal Form (2NF), and in addition, it must ensure that all its non-prime attributes are directly dependent on the primary key and not on any other non-prime attribute. This means a relation in 3NF allows no transitive dependencies of non-prime attributes on the primary key. So, in this case, it avoids the other of our three types of functional dependencies, transitive dependency.

d) Question:

A diagram of a book

Description automatically generated

Answer: I can’t understand students answer one this one, I asked in Slack so hopefully someone answers.

**4. SQL**

a) Question:

A screenshot of a hotel registration form

Description automatically generated

Answer: All SQL prompts:

1. SELECT \* FROM Hotel
2. SELECT \* FROM Hotel WHERE hotelCity = “Malmö”
3. SELECT g.guestNo, g.firstName, g.lastName, g.guestAddress  
   FROM Guest as g  
   INNER JOIN Booking as b  
   ON g.guestNo = b.guestNo  
   INNER JOIN Hotel as h  
   ON b.HotelNo = h.hotelNo  
   WHERE h.hotelCity = “Malmö” ORDER BY g.lastName DESC
4. SELECT COUNT(\*)   
   FROM Booking WHERE hotelNo = 5
5. SELECT h.hotelNo, h.hotelName, h.hotelType, h.hotelAddress, h.hotelCity, h.numRoom   
   FROM Hotel as h  
   INNER JOIN Room as r ON h.hotelNo = r.hotelNo  
   WHERE r.roomPrice IN(SELECT MAX(roomPrice) FROM Room)  
   GROUP BY h.hotelNo
6. SELECT b.bookingNo, b.hotelNo, b.guestNo, b.checkInDate, b.checkOutDate, b.roomNo  
   FROM Booking as b  
   INNER JOIN Hotel as h ON b.hotelNo = h.hotelNo  
   WHERE h.hotelName = “Scandic” AND b.checkInDate IS MONTH(7)
7. SELECT hotelCity, COUNT(\*) FROM Hotel GROUP BY hotelCity