1. Introduction

Q.3.1.1.1 Give an example of data which is (i) a database, (ii) not a database.
Answer: A database is a collection of related data. We provide here an example of a database that follows relational model.
(i) Student file

rollNo	*name*
101	Pravin Dessai
105	Kavita Nanda
129	Nandita Dalvi
140	Reboti Sinha
150	Rakesh Paroyal

Marks file

rollNo	*mark1*	*mark2*	*mark3*
101	44	62	61
105	75	92	82
129	96	90	97
140	90	63	72
150	66	69	75

The records in Student file and Marks file are related through field *rollNo*. The given collection of (data) files is a database.

(ii) Student file

rollNo	*sName*
101	Pravin Dessai
105	Kavita Nanda
129	Nandita Dalvi
140	Reboti Sinha
150	Rakesh Paroyal

Product file

pcode	*cost*	*pName*
CC231	1421	y-small
ET890	560	x-small
GI501	69	x-medium
JK883	1129	y-medium
VB907	672	x-big

Records in Student file and Product file are not related. So, this collection of (data) files is not a database. Rather, it is a collection of two databases, each containing one file.

Q.3.1.1.2 State any two types of information that may be available in a database.
Answer: A database might contain information about entities and relationships. A many-to-many relationship in an ERD, gets mapped into a table (relation), it contains relationship information. When an entity in an ERD gets mapped into a table (relation), it contains entity information.

Q.3.1.1.3 Why does database management system (DBMS) come into existence?
Answer: Due to technological advancements, a system, nowadays, can capture a large volume of data. The value of data is widely recognized. Tools are required to update data, manage data and extract information within a short span of time. Such tools are commonly called as database management systems.

Q.3.1.1.4 Describe some situations, where the use of DBMS is not desirable.
Answer: We describe here a few situations, where a DBMS is not useful.
i. Real time applications, where certain well-defined critical operations are required to be performed.
ii. An application that needs to process data in a specific way, which is not supported by a DBMS.
iii. The database and applications are simple, well defined, not expected to change.

Q.3.1.1.5 Discuss steps involved to setup a database for an organization.
Answer: Important steps for setting up a database for an organization are given below.
∗ Specify the requirements of the organization.
∗ Specify a model containing all appropriate types of data and data relationships.
∗ Specify the integrity constraints on the data.
∗ Specify the physical level.
∗ Define a user interface to carry out a task, and write the necessary

application programs to implement the user interface.
∗ Create and initialize the database.

Q.3.1.1.6 What is information integration? Discuss approaches to implement it.
Answer: Information integration means joining the information contained in many related databases to make it into a single one. If a company has several divisions, and each division may have built its own database over the time. Then the goal of information integration is to build a structure on the top of the existing databases, and access and modify information from the top.
One approach is to build a data warehouse, where information from many legacy databases is copied periodically. Another approach is to create a middleware, whose function is to support an integrated model of the data of the various databases by translating between the model used by it and the actual model used by each database.

Q.3.1.1.7 Explain the following terms in the realm of DBMS.
metadata, log records, statistical information and indexes
Answer: Metadata is data about data. It includes the database schema definitions and constraints on the database. Log records are the records pertaining to the sequence of update activities on the database. Statistical information is gathered and stored by the DBMS about data properties, such as the sizes of and values in, the various relations or other components of the database. Various data structures used to access database efficiently are called indexes.

Q.3.1.1.8 What is schema? Give an example.
Answer: A schema is a description of data in terms of a data model. Then a schema of relation using a relational model is specified by attribute names, types of attributes, length of attribute, etc.
Example: Consider that there is a customer table in an application. Then the schema of customer table can be specified as follows:
$Customer = (custId : string(15), custName : string(30), custAddr : string(60), contactNo : string(12))$
But, the description of $Customer$ schema might vary from an application to another application.

Q.3.1.1.9 Illustrate the concept of semantic data model.
Answer: Semantic data model (SDM) is a high-level semantics-based

2. Basic Concepts

Q.3.1.2.1 Find the candidate keys of relation *Student* as given below.
$Student = (rollNo, name, dateOfBirth, contactNo,$
$major, class, address1, address2)$, where $address1$, aand $address2$ together forms the address of a student.
Answer: We assume that $contactNo$ attribute is single valued. Possible candidate keys are $\{rollNo\}$, $\{contactNo, name\}$, $\{name, dateOfBirth\}$, $\{name, address1, address2\}$.

Q.3.1.2.2 What do you mean by conceptual schema? Explain it with the help of an example.
Answer: Conceptual schema is commonly known as logical schema. It describes data in terms of a data model. Using conceptual schema, data can be represented as entities and their relationships.
Example: We assume here the underline data model as relational.
$Customer = (custId : string(15), custName : string(30), CustAddr : string(60), contactNo : string(12))$
$Account = (accNo : integer, bankId : string(15), balance : real)$
$OpenAccount = (custId : string(15), accNo : integer, dateOpened : date)$
Here data are represented by relational model. Relation *OpenAccount* is a relationship between entities *Customer* and *Account*.

Q.3.1.2.3 What is logical data independence? Why does it difficult to achieve as compared to physical data independence?
Answer: When the application programs are independent of the logical structure of data then programs are independent of logical data. In case of physical data independence, application programs are independent of physical storage details. Most of the application programs are dependent on the logical structure of data. Therefore, it is difficult for application programs to be independent of logical structure of data.

Q.3.1.2.4 Make distinction between primary key, candidate key, and superkey.
Answer: A superkey is a set of one or more attributes that, taken collectively, can identify uniquely an entity in the entity set. A superkey may contain extraneous attributes. If K is a superkey, then so is any superset

of K. A superkey for which no proper subset is also a superkey, is called a candidate key. It is possible that several distinct sets of attributes could serve as candidate keys. The primary key is one of the candidate keys that is chosen by the database designer as the principal means of identifying entities within the entity set.

Q.3.1.2.5 What is atomic value? Give three examples of data that are not atomic.
Answer: Edgar Codd (1990) defined an atomic value as one that cannot be decomposed into smaller pieces by the DBMS.
For example, *birthDate* is not atomic, since it is composed of day, month and year. Also, the *name* of a person is not atomic, as name is a combination of first name, middle name and surname. Alos, $registrationNumber$ of a student may be viewed as non-atomic, since it is composed of year, branch code and serial number.

Q.3.1.2.6 Justify the validity of the statement.
Each foreign key refers to a primary key of a relation.
Answer: Each foreign key ideally refers to a candidate key of a relation. A primary key is also a candidate key. Thus, the above statement is also true. One can change the above statement, and a revised statement is given below.
Each foreign key refers to a candidate key of a relation.
Such integrity constraint is required. Suppose a relationship table connects two base tables. When we enter a record in the relationship table, we must check that the corresponding record must exist in these two base relations.

Q.3.1.2.7 Consider a multi-floor office having the following relation:
$Cabin = (cabinNo, roomNo, floorNo, phoneNo)$,
where each floor has multiple rooms, each room has multiple cabins. Assume that each floor has a unique phone number, and cabin numbers in different rooms may be the same. Find the candidate keys of Cabin relation.
Answer: Note that *cabinNo* may get repeated in different rooms. Assume that $roomNo$ is unique. Possible candidate keys are $\{cabinNo, roomNo\}$, $\{cabinNo, phoneNo\}$.

Q.3.1.2.8 Consider a relation $R = (X, Y, Z)$ having all unique valued attributes. Find the candidate keys of R. Check whether $\{X, Y\}$ is a

superkey.
Answer: Each attribute in R is unique valued. Then each attribute can form a candidate key. Thus, candidate keys are $\{X\}, \{Y\}, \{Z\}$. Any superset of a candidate key is a superkey. Then $\{X, Y\}$ is a superkey.

Q.3.1.2.9 There are two integer arrays A and B of order $m \times n$ have been used in a program. With respect to this example, distinguish between (i) three levels of data abstraction, (ii) a schema and an instance.
Answer: (i) At the physical level, each array is an $m \times n$ (possibly consecutive) locations, each of size 2^p (say, 32) bits. At the conceptual level, each array is a grid of boxes arranged in m rows and n columns. There are $2^{m \times n}$ possible views. A view consists of any part of the given grid of data.
(ii) The schema of an array, *grid*, is given in C language as follows.

```
typede struct arrayType {
        int box [20][30];
} grid;
```

Two instances A and B of type *grid* are given below.
grid A, B;
Here, instances are like variables in a programming language, and schema is like type definition of a variable.

Q.3.1.2.10 What is data dictionary?
Answer: A database scheme is specified by a set of definitions which are expressed by a special language called a data definition language (DDL). The result of complication of DDL statements is a set of tables which are stored in a special file, called data dictionary. It is data about data, also known as meta-data.

Q.3.1.2.11 Represent the following
(i) entities using conceptual schema: *Student*, *Faculty*, *Course*
(ii) relationships using conceptual schema: *Teaches*, *Enrolled*
Answer: (i) *Student* (rno : *string*[10], *sname* : *string*[30], *class* : *string*[10])
Each student has a registration number (rno), name (*sname*). Also, each student is associated with a *class* (e.g, first year)
Faculty (fid : *string*[10], $fname$: *string*[30], did : *string*[30], *spec* : *string*[40])
Each faculty has name ($fname$) and a dependent identification (did). Each faculty has specialization (*spec*) in some subjects and a faculty

3. Entity Relationship Models

Q.3.1.3.1 Give three examples of each of the following relationship types: one-to-one, one-to-many, many-to-many.
Answer: Three examples of one-to-one relationship are given below. (i) relationship "capital of" between country and capital city, (ii) relationship "social security of" between citizen and security card, (iii) relationship "ID card of" between employee and ID card.
Examples of many-to-one relationship are (i) relationship $works\_for$ between employee and department, (ii) relationship *published_by* between book and publisher, (iii) relationship $from$ between car model and car company.
Some examples of many-to-many relationships are (i) relationship *book_author* between book and author, (ii) relationship *employee_project* between employee and project, (iii) relationship *course_student* between student and course.
Note: An one-to-many relationship can also be viewed as a many-to-one relationship.

Q.3.1.3.2 Explain different participation constraints with the help of an example.
Answer: Consider binary relationship *works_on* (see Fig. 3.1) between Employee and Project. We note that every project has some employee(s) to work on. Thus, the participation of Project entity with this relationship is total. On the other hand, each employee may not be assigned to work on a project. Thus, participation of Employee entity with this relationship is partial. A total participation is represented by a parallel lines, where as a partial participation is shown using a single line.

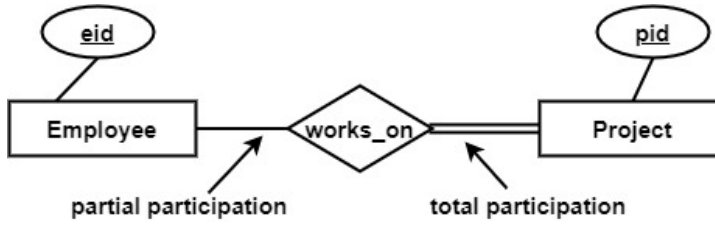

Figure 3.1: ERD showing total and partrial participations

Only key fields of entities are shown, and they are underlined.

Q.3.1.3.3 In a hotel reservation system, hotel room and person are connected through relationship *Booking*.
(i) What is the cardinality of relationship *Booking*?
(ii) Price of a room may change over the seasons. How do you model the situation in your ERD?
(iii) Describe the relation *Booking* created after mapping the ERD into tables.
Answer: (i) One room can be booked by multiple people over time, and again one person can book multiple rooms. Thus, *Booking* is a many-to-many relationship.
(ii) Each room can be associated with standard price, and this price can be applicable to normal seasons. During slack time, the price may come down and the price may go up during peak-time. This variation can be managed by adding an attribute *priceChange* with the relation *Booking*.
(iii) Relation *Booking* can be described by the following attributes: $personName$, $roomId$, $bookingDate$, $priceChange$, $startDate$, $endDate$. *Booking* is a many-to-many relationship. So, there will be a table corresponding to *Booking* relationship in the ERD (not shown here). The primary keys of "Hotel room" ($roomId$) and "Person" ($personName$) are brought into this table.

Q.3.1.3.4 Give examples of three ternary relationships.
Answer: Examples of three relationships are given below.
(i) One can define a ternary relationship "contract" between main actor, movie studio and movie. The cardinalities of these entities are n, 1 and n respectively.
(ii) A ternary relationship "subject_taught" can exists between course, subject and teacher. The cardinalities of these entities are 1, n and n respectively.
(iii) We could define a ternary relationship "prescription" between doctor, medicine and patient. The cardinalities of these entities are 1, n and n respectively.

Q.3.1.3.5 Define the terms: degree of a relation, cardinality of a relation instance.
Answer: The *degree of a relation* is the number of columns (attributes) in the relation. The *cardinality of a relation instance* is the number

of tuples in the relation at a particular point of time.

Q.3.1.3.6 Choose the appropriate one.
ER diagram is used to model database design at (i) view level, (ii) conceptual level, (iii) physical level
Answer: (ii)

Q.3.1.3.7 What are composite, derived and multi-valued attributes? How are they shown in ERDs?
Answer: An attribute that is composed of more than one attribute is known as *composite attribute*.

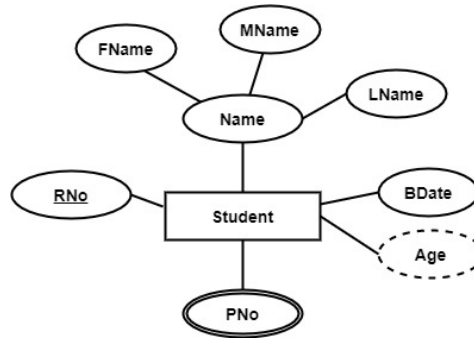

Figure 3.2: Entity Student is represented with a composite attribute (Name), a derived attribute (Age), and a multi-valued attribute, phone number (PNo)

For example, $Name$ is a composite attribute. Attribute $Name$ is composed of first name (FName), middle name (MName) and last name (LName). A composite attribute is divided into a tree-like structure in an ERD.
An attribute whose value can be obtained from other attribute is called a *derived attribute*. Here Age is a derived attribute, since it can be obtained from the date of birth ($BDate$). Derived attributes are depicted by dashed ellipse.
An attribute that may assume multiple values at a time is called a $multi-valued\ attribute$. Pnone number (PNo) can have multiple values. Even, it may not have a value also. Multivalued attributes are depicted by double ellipses.
All these attributes are shown in Fig. 3.2.