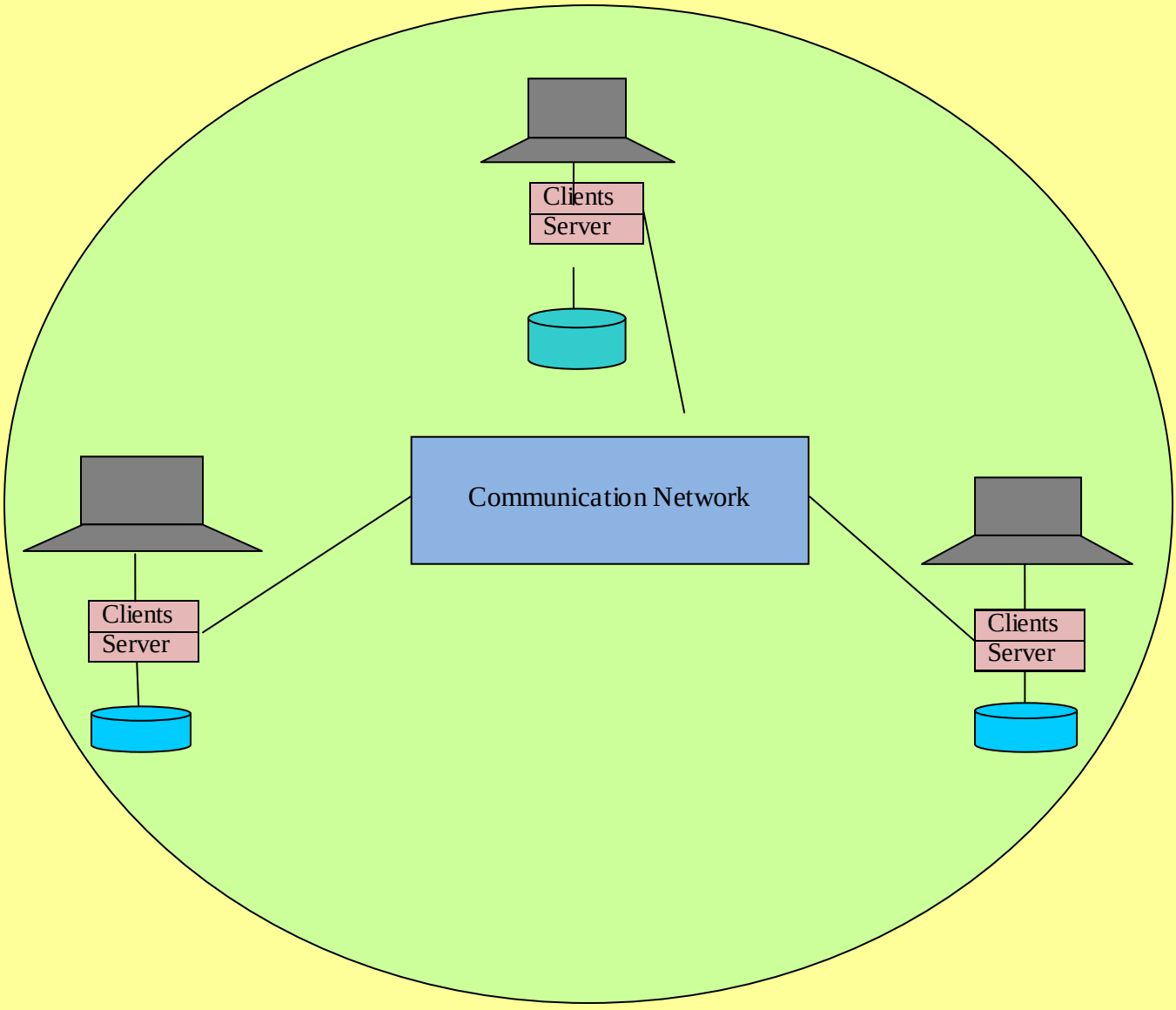
The Dummies' Guide to Database Systems: An Assembly of Information

An Easy to Understand Guide Even for Laypeople



Rosina S Khan

Dedicated to:

You, the Valued Reader

http://www.rosinaskhan.weebly.com

Contents

Preface

Nowadays there is a growing need to organize, store and retrieve information of any organization real fast and convenient. Lots of paper files need manual work for updates and throwing away paper clutter, retaining the importanrt ones. All of this can be done smoothly and speedily by making the information computerized via databases. That is what this book is about and it covers a fundamental approach to this topic which can be taught and be useful for undergraduate courses in databases or any newbie who wants to grab the concepts behind databases.

While databases are themselves actually a collection/assembly of information, the content of this book is really an assembly of information collected from the following resources. **So the title of the book is intentionally ambiguous.** I must admit many of the chapters in this book are written in such an easy-to-understand matter that even laypeople can grasp the concepts.

[1] Database System Concepts, 5th Edition, Abraham Silberschatz, S. Sudarshan, Mc Graw Hill International Editions, 2006

[2] Professor Dorothee Koch's Lecture notes, Stuttgart University of Applied Sciences, Germany, 2005

[3] An Introduction to Database Systems, C. J Dates, Addison Wesley, 8th Edition, 2003

[4] http://planetcassandra.org/what-is-nosql, 2014

Organization

Let me now explain the organization of this book.

Chapter 1 is an introductory chapter starting with the idea of introducing backend and front end of database systems and covers stuff from [1] about why databases are at all useful.

Chapter 2 contains basics of Entity Relationship model (ERM), the fundamental step in designing databases at the backend and the resources are mainly from [1] and [2].

Chapter 3 covers how to convert ERMs to relational models (table schemas) [2], and introduces relational algebra, a pure and procedural form of query language [1].

Chapter 4 introduces Structured Query Language (SQL), the most widely used language used for querying relational databases and retrieving info. [mainly [2] and partly [1])

Chapter 5 covers examples of integrity constraints (conditions) that can be imposed in relational systems while creating tables in SQL [2].

Chapter 6 consists of functional dependencies of one attribute (field) on another attribute(s) in a table and based on these whether we have to split the tables or not according to violation or not of normal forms based on the concept of normalization. [2]

Chapter 7 mainly covers query processing which is the series of activities involved in extracting data from a database. [1]

Chapter 8 explains why the need to map databases to files may arise and how. [1]

Chapter 9 is a short explanation of a data dictionary and what info about databases it contains. [1]

Chapter 10 includes sophisticated indexing techniques for files. Just as words or phrases in a text book index appear in a sorted order, an index for a file in a database works in a similar way. [1]

Chapter 11 introduces the concept of transactions which are a sequence of data access operations that transfers the database from one consistent state to another consistent state. [2]

Chapter 12 explains recovery in transactions and how to preserve correctness and consistency of data over time, allowing for parallel access (transactions) of multiple users after transaction failures. [2]

Chapter 13 covers concurrency control which rectifies the problems occurring if two or more transactions using the same data items are executed in parallel. [2]

Chapter 14 includes advanced databases covering introductions to distributed databases and their possible architectures, data warehouses, multimedia databases and data mining in brief, as well as introduction to NoSQL environment. [1,2,3,4]

There is also a miscellaneous database project at the very end which students can work on through out the whole semester in parallel with theory lectures. The project is equally useful for even those who are not students and can work on it in their own interests.

Acknowledments

I am deeply indebted to the authors and publisher of Database System Concepts, Professor Dorothee Koch, C.J Dates and publisher, and Planet Cassandra website for collecting their resources, paraphrasing and combining them with mine and hence the creation of this very book.

Last but not the least I am thankful to my mom and Dr Manzur for all their help and support while writing out this book.

--Rosina S Khan

C H A P T E R 1

INTRODUCTION

A database system is defined to be a collection of interrelated data and a set of programs to access those data.

Why *interrelated* data? Because some sort of relationship exists among the data. (It will become clearer when we come to the topic *Normalization of tables.)* This actually occurs in the backend. A backend is nothing but a collection of interrelated data in different tables. A table has some fields in a row. These fields are also called attributes. Corresponding to the attributes are some data in rows. These are called records or tuples. The entire table corresponds to an entity. We will cover more on entities and attributes in the next chapter.

Why *set of programs*? Programs here refer to application programs at the front end or *interfaces* connected to data at the backend. A typical layout for backend and front end is given as an example below:

Table 1.1: Jone's Account Info (*Backend)*

Accnt _A	Accnt_B
150	200

Fields/attributes

Tuple/record

Table 1.2: Jone's Updated Account Info after transaction

Accnt _A	Accnt_B
100	250

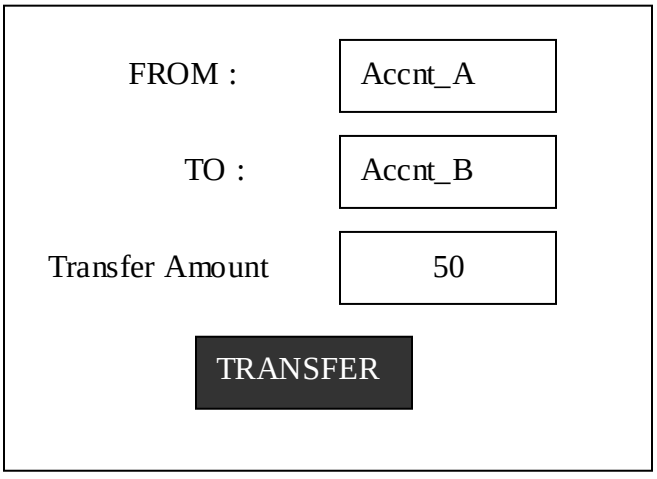


Fig 1.1: Jone's Account Interface (*Front end)*

A backend database for example, may consist of Jone's Account Info table. After a transaction, Jone's Account Info may be updated as shown in Table 1.2. An interface to the backend database may be depicted as shown in Fig 1.1. Given the values of source and destination accounts as well as the transfer amount in textboxes, hitting the transfer button will enable to make the transaction, which will in turn be updated in Jone's Account Info table in the backend.

The backend may be developed using software tools such as MS SQL Server, MySQL etc. while the front end may be developed using C#, or PHP, JavaScript and HTML, etc.

Before the advent of databases, organizations stored information using a typical file-processing system. In this system, permanent records were stored in various files and different application programs were written to extract records and add records to the appropriate files.

File-processing systems have a number of disadvantages. These are outlined as follows:

- Data redundancy and inconsistency: Different programmers may write the files and application programs over a passage of time. As a result, files may have different structures and the programs may be written in different several programming languages. Also, the same information may appear in different files. Data inconsistency results when the same information is updated in one place but not in another place in addition to higher storage and access cost.

- Difficulty in accessing data: Data retrieval may be problematic. Suppose a bank officer needs to find out the list of customers who live in a certain postal code. Such an application program does not exist. There is, however, an application program to generate the list of customers. The data processing department based on the demand of the bank officer has either to generate the list of customers and extract the needed info manually. The other alternative is to write a new application program to meet the demand. Both the alternatives are unsatisfactory. After a week or so, the bank officer needs to trim down the list of customers with bank balances more than Tk50,000. Again the data processing department is left with two alternatives both of which are unsatisfactory. The point here is that file processing systems do not allow data to be retrieved in a convenient and efficient manner.

- Data Isolation: As data lie in different files and files may be in different formats, it is difficult to retrieve the appropriate data by writing new application programs each time.

- Integrity Problems: The data values stored in the database must satisfy certain conditions called integrity or consistency constraints. For example, the bank balance of a customer must never fall below Tk200. Developers impose these constraints by writing appropriate code in the various application programs. To enforce a new constraint such as the bank balance of customers should not exceed

10 crore taka, it becomes difficult for the developer to enforce the new constraint and change the programs. The problem worsens when constraints involve several data items from different files.

- Atomicity Problems: Atomic transactions mean they either occur completely or none at all. For example, consider a program to transfer Tk500 from Account A to Account B. If a system failure occurs during the transaction, it is possible Tk500 was debit from Account A but not credited to Account B, resulting in an inconsistent database state. It is important that to maintain database consistency, either both the debit and credit occur, or that neither occurs. It is difficult to ensure atomicity in conventional file processing systems.

- Concurrent-access anomalies: Multiple users may update a system simultaneously. This facilitates faster response and overall performance of the system. Interaction of concurrent updates may result in inconsistent data. Consider bank account A, containing Tk500. If two customers withdraw funds say, Tk50 and Tk100, respectively from Account A at about the same time, the result of concurrent transactions may leave the account in an inconsistent state. If the two transactions occur concurrently, they may both read the value Tk500, and write back Tk450 and Tk400, respectively. Depending on which one writes the value last, the account may contain either Tk450 or Tk400, rather than the correct value of Tk350. To prevent this from happening, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by various different application programs that have not been coordinated previously.

- Security problems: Every user of the database system should not be able to access all the data. For example, in a banking system, payroll personnel (tax officer) needs to see only that part of the database that has information about the various bank employees. They do not need to access information about customer accounts. As another example, bank tellers see only that part of the database that has information on customer accounts. They cannot access information about salaries of bank employees. Enforcing such security constraints on a file-processing system is difficult because application programs are added to the system in an ad hoc manner.

1.1 View of Data

A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

1.1.1 Data Abstraction

Since many database-system users are not handy with computers, developers hide certain complexity details through several levels of abstraction in order to make it easier for users' interaction with the system.

- Physical level: The lowest level of abstraction describes how data are actually stored. It describes complex low-level data structures in detail.

- Logical level: The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. It describes the entire database in terms of a small number of relatively simple structures. Database administrators use the logical level of abstraction who must decide what information to store in the database.

- View Level: The highest level of abstraction describes only part of the entire database. In this level users need to access only part of the entire database. The level simplifies the interaction of users with the system. The system may provide many views for the entire database.

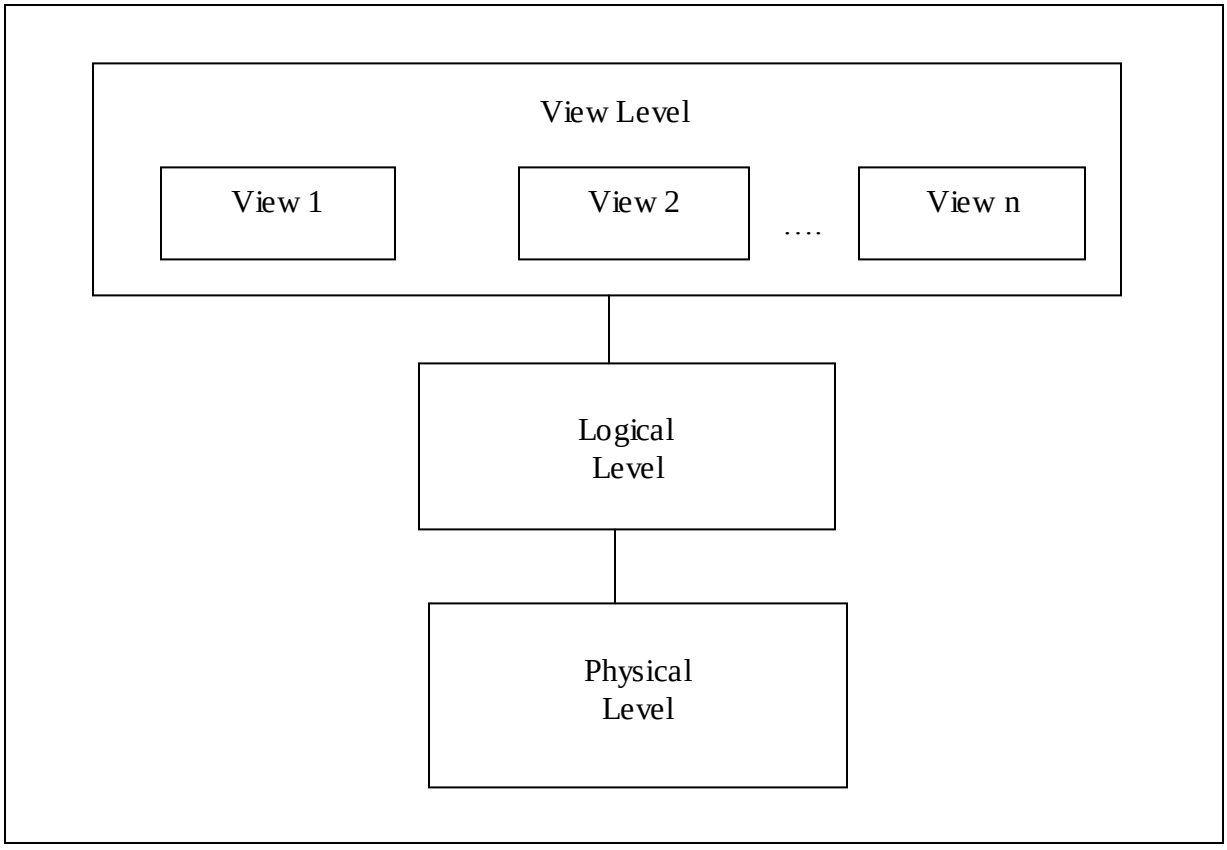


Fig 1.2: The three levels of data abstraction

Distinction among levels of abstraction may be compared to the concept of data types in programming languages. Most high-level programming languages support the notion of a structured type. For example, in a Pascal-level language, we may declare a record as follows:

```
type  customer = record
                    customer_id: string;
                    customer_name: string;
                    customer_street: string;
                    customer_city: string;
                 end;
```

Code 1.1: Pascal-like definition of record structure

Code 1.1 defines a record type called customer with four fields. Each field has a name and a type associated with it. A banking enterprise may have several such record types, including

- account, with fields account_number and balance
- employee, with fields employee_name and salary

At the physical level, a customer, account, or employee record can be described as block of consecutive storage locations or bytes. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrator, however, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition as shown in Code 1.1, and the level also defines the interrelationships among the different record types. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction too.

Finally at the view level, computer users see a set of application programs or front end interfaces that hide details of the data types. At this level several views are defined and database users see and access these views. The views also provide a kind of security mechanism to allow users only to access certain parts of the database. This has been explained in detail in the subsection **Security problems** on page 3. Some examples of views can be:

- View1: customer_name|account_number|balance
- View2: employee_name|account_number|balance|salary

1.2 Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema.

Consider the following database table:

Employee Table:

employee_name	salary
Jones	10000
Jim	20000
Jack	15000
David	25000
Henry	30000

} Schema

} Instance

In the above Employee Table, schema and instance are clearly shown.

Database systems have several schemas, partitioned according to the levels of abstraction. The physical schema describes the database design at the physical level, while the logical schema describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

The logical schema is the most important among all the schemas since programmers construct application programs or front end interfaces by using logical schemas. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting logical level. Hence, application programs do not need to be rewritten if physical schema changes and are said to exhibit physical data independence.

1.3 Database Languages

A database system provides a data-definition language to specify the database schema and a data-manipulation language to express database queries and updates. In practice, the data-definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL (Structured Query Language).

a) Data-Manipulation Language

A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- Retrieval of information stored in the database
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

b) Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL). The DDL is also used to specify additional properties of the data. We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

1.4 Database Users

- **Application programmers** are computer professionals who write application programs or front end interfaces. They can choose tools such as **Rapid application development (RAD)** to develop front end interfaces such as forms and reports with minimal programming effort.

- **Sophisticated users** interact with the system without writing programs. They form their requests to the system using a database query language e.g, SQL.

- **Specialized users** are sophisticated users who write specialized database applications such as computer-aided design (CAD), knowledge-base and expert systems that store complex data such as graphics and audio data and environment-modeling systems.

- **Naive users** are unsophisticated users who interact with the system by invoking one of the application forms that have been written previously. An example of an application program can be that of Jone's Account Interface in Fig. 1.1 where the naïve users fill in the fields of the form and hit the button. They may also simply read reports generated from the database

1.5 Database Administrator

A person who has central control over the whole database system is a database administrator (DBA). His functions may be summarized as below

- **Schema definition**: The DBA creates the original database schema or the overall design of the database.

- **Storage structure and access-method definition:** The DBA is aware of certain details of physical organization of the data i.e. how records occupy storage locations or bytes and the methods to access those data.

- **Schema and physical-organization modification:** The DBA can change the schema and physical organization according to the demands of the organization, or simply to improve performance. This can be done without affecting application programs at logical level.

- **Granting of authorization for data access:** The DBA can grant different types of authorization to different users and hence control which user has the right to access which parts of the database. This enhances to keep the system secure as has been explained earlier.

- **Routine maintenance:** Some examples of DBA's routine maintenance activities are as follows:

  i) Backup the database periodically to tapes or remote servers so that they can be recovered in case of disasters such as database sabotage.
  ii) Ensure enough disk space is available for normal operations and upgrade disk space as required.
  iii) Monitor jobs running in the database and ensure overall performance is good.

C H A P T E R 2

ENTITY RELATIONSHIP MODEL

2.1 Entity, Entity set and Attributes

An entity is a thing or object in the real world that is distinguishable from other objects. An entity can be concrete, such as a book or a person or it may be an abstract, such as a loan, holiday or a concept. An entity in a database system actually represents a table.

The properties or parts of an entity are called attributes. For example, a person has the attributes person_id, name, occupation, salary etc. A book has the attributes book_id, author, publisher, category, number of copies etc. Attributes are actually the fields of a database table or entity.

An entity set is a set of entities of the same type that share the same properties or attributes. For example, the set of all students who take a class can represent a student entity set in which each entity is a student sharing similar attributes with other students such as student_id, name, contact_no, address etc.

2.2 Types of attributes

An attribute can be of the following types:

- **Simple and Composite** attributes: Simple attributes are those as the name implies that is they are not divided into subparts. For example, the attribute *student_id* for the entity student has no subparts and therefore, is a simple attribute. On the other hand, an attribute *name* can be structured as a composite attribute consisting of first_name, middle_name and last name. As another example, the attribute *customer_street* for the entity customer can be composite consisting of street_number, street_name and apartment_number. (Fig 2.1)

- **Single-valued and mutivalued** attributes: The loan_number attribute for a specific loan refers to one loan_number. Such attributes are said to be single valued. On the other hand there may be instances where an attribute has a set of values for a specific entity. For example, consider the employee entity set with the attribute phone number. An employee may have zero, one or several phone_numbers and different employees may have different number of phone numbers. This type of attribute is said to be multivalued. As another example, the attribute dependent name of the employee entity set would be multivalued, since any particular employee may have zero, one or more dependent(s).

  Upper and lower bounds may be placed on the values in a multivalued attribute as needed. For instance, a bank may limit storing the number of phone_numbers for a customer to two. Placing bounds in this way means that the phone_number attribute for the customer entity set may have the range 0<=phone_number<=2.

- **Null** attributes: An attribute takes a null value when that value is missing, not applicable or unknown. For example, a person may have no middle name (not applicable). If the name for a particular customer is null we assume that the value is missing since every customer must have a name. A null value for an apartment_number could mean the address does not include an apartment number (not applicable), that an apartment number exists but we do not know what it is (missing), or that we don't know whether an apartment number is part of the customer's address (unknown).

- **Derived** attributes: The value for this type of attribute can be derived from the values of other related attributes or entities. For example, a customer entity has the attribute customer_age. If the customer attribute also has an attribute date_of_birth, we can calculate his age from data_of_birth and the current date. Therfore, customer_age is a derived attribute. As another example, the entity employee can have employment_length as an attribute. If this entity has another attribute start_date of his employment, then we can calculate the employee's employment_length from the start_date and current_date. Hence, in this case employment_length is a derived attribute.

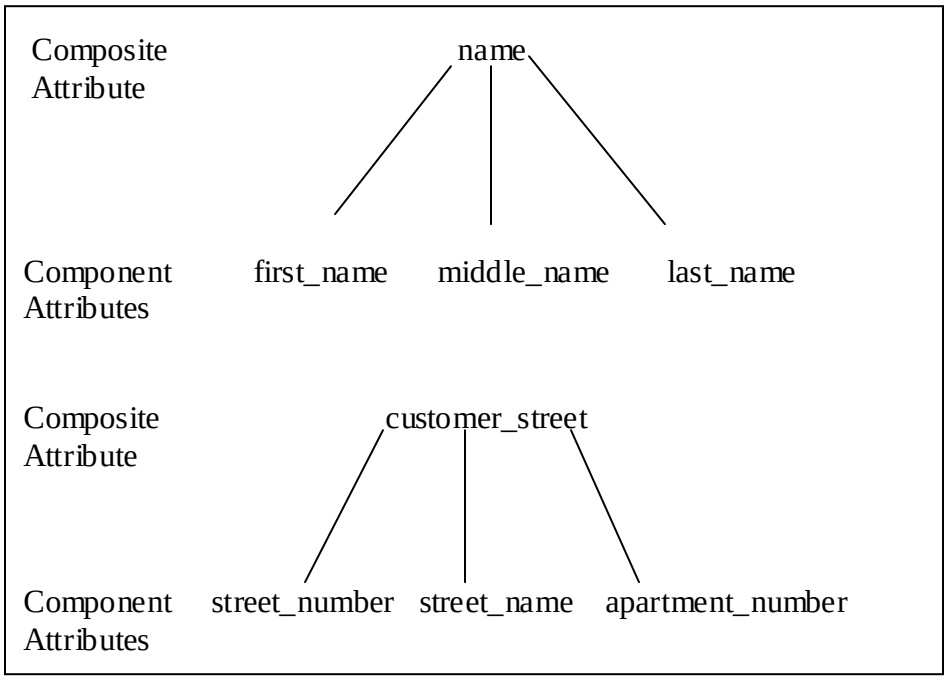


Fig. 2.1: Composite Attributes *name* and *customer_street*