B.B.A.
(Computer Application)
Semester-I

NEW SYLLABUS
CBCS PATTERN

DATABASE MANAGEMENT SYSTEMS

Dr. Ms. MANISHA BHARAMBE
ABHIJEET MANKAR

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

DATABASE MANAGEMENT SYSTEMS

For B.B.A.(CA) : Semester - I

[Course Code 104 : Credit - 3]

CBCS Pattern

As Per Revised Syllabus, Effective from June 2019

Dr. Ms. MANISHA BHARAMBE
M.Sc. (Computer Science), M.Phil. Ph.D. (Comp. Sci.)
Associate Professor,
Department of Computer Science,
MES's Abasaheb Garware College,
PUNE 4.

ABHIJEET D. MANKAR
M.C.S., SET
Assistant Professor,
Department of Computer Science,
Tuljaram Chaturchand College of Arts,
Science and Commerce (Autonomous),
BARAMATI.

Price ₹ 200.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N4939

DATABASE MANAGEMENT SYSTEMS **ISBN 978-93-89108-89-7**

First Edition : **July 2019**

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041
Mobile No. 9404233041/9850046517

➢ DISTRIBUTION CENTRES

PUNE

Nirali Prakashan (For orders within Pune) : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan (For orders outside Pune) : S. No. 28/27, Dhayari, Near Asian College Pune 411041
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976, Fax : (022) 2386 9976
Email : niralimumbai@pragationline.com

➢ DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra
Tel : (0257) 222 0395, Mob : 94234 91860; Email : niralijalgaon@pragationline.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : niralikolhapur@pragationline.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : niralinagpur@pragationline.com

DELHI

Nirali Prakashan : 4593/15, Basement, Agarwal Lane, Ansari Road, Daryaganj
Near Times of India Building, New Delhi 110002 Mob : 08505972553
Email : niralidelhi@pragationline.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: niralibangalore@pragationline.com

Other Branches : Hyderabad, Chennai

Preface ...

We take an opportunity to present this Text Book on **"Database Management System"** to the students of First Semester B.B.A. (Computer Applications). The objective of this book is to present the subject matter in a most concise, compact, to the point and lucid manner.

This book aims at clarifying and explaining the fundamental concepts of designing and implementing databases, which is helpful throughout the life time as a student and professional.

Extensive exercises and numerous case studies are provided at the end of each chapter, which helps to develop the skills of database application design and implementation. Without normalization database is of no use. This book clears the concept and enhance the ability of designing normalized database.

We are very much thankful to our Publisher **Shri. Dineshbhai Furia** and **Shri. Jignesh Furia** for publishing this book in minimum possible time.

Any suggestions for improving the content shall be gratefully received and highly appreciated.

Authors

Syllabus ...

1. File Structure and Organization **(6 Lectures)**

- 1.1 Introduction
- 1.2 Logical and Physical Files
  - 1.2.1 File
  - 1.2.2 File Structure
  - 1.2.3 Logical and Physical Files Definitions
- 1.3 Basic File Operations
  - 1.3.1 Opening Files
  - 1.3.2 Closing Files
  - 1.3.3 Reading and Writing
  - 1.3.4 Seeking
- 1.4 File Organization
  - 1.4.1 Field and Record Structure in File
  - 1.4.2 Record Types
  - 1.4.3 Types of File Organization
    - 1.4.3.1 Sequential
    - 1.4.3.2 Indexed
    - 1.4.3.3 Hashed
- 1.5 Indexing
  - 1.5.1 What is an Index ?
  - 1.5.2 When to use indexes ?
  - 1.5.3 Types of Index
    - 1.5.3.1 Dense Index
    - 1.5.3.2 Sparse Index

2. Database Management System **(14 Lectures)**

- 2.1 Introduction
- 2.2 Basic Concept and Definitions
  - 2.2.1 Data and Information
  - 2.2.2 Data Vs Information
  - 2.2.3 Data Dictionary
  - 2.2.4 Data Item or Field
  - 2.2.5 Record
- 2.3 Definition of DBMS
- 2.4 Applications of DBMS
- 2.5 File processing System Vs DBMS
- 2.6 Advantages and Disadvantages of DBMS
- 2.7 Users of DBMS
  - 2.7.1 Database Designers
  - 2.7.2 Applications Programmer
  - 2.7.3 Sophisticated Users
  - 2.7.4 End Users
- 2.8 Views of Data
- 2.9 Data Models
  - 2.9.1 Object Based Logical Model
    - a. Object Oriented Data Model
    - b. Entity Relationship Data Model
  - 2.9.2 Record Base Logical Model
    - a. Relational Model

b. Network Model
c. Hierarchical Model

2.10 Entity Relationship Diagram (ERD)
2.11 Extended Features of ERD
2.12 Overall System Structure

3. Relational Model **(8 Lectures)**

3.1 Introduction
3.2 Terms
a. Relation
b. Tuple
c. Attribute
d. Cardinality
e. Degree of Relationship Set
f. Domain
3.3 Keys
3.3.1 Super Key
3.3.2 Candidate Key
3.3.3 Primary Key
3.3.4 Foreign Key
3.4 Relational Algebra Operations
a. Select
b. Project
c. Union
d. Difference
e. Intersection
f. Cartesian Product
g. Natural Join

4. SQL (Structured Query Language) **(12 Lectures)**

4.1 Introduction
4.2 History of SQL
4.3 Basic Structure
4.4 DDL Commands
4.5 DML Commands
4.6 Simple Queries
4.7 Nested Queries
4.8 Aggregate Functions

5. Relational Database Design **(8 Lectures)**

5.1 Introduction
5.2 Anomalies of an Normalized Database
5.3 Normalization
5.4 Normal Form
5.4.1 1NF
5.4.2 2NF
5.4.3 3NF
5.4.4 BCNF

Contents ...

1... File Structure and Organization

Learning Objectives...

- To get familiar with the fundamental concept of File and File Organization.
- To know about basic File Operations.
- To know about the indexing of Files and its types.

1.1 INTRODUCTION

- A file organization is a method of arranging the records in a file.
- The file is stored on secondary storage device called hard disk.
- A file can be accessed or modified in different ways.
- This is done to perform some basic operations on the records available in the file.
- For example: sort the records in ascending order on employee's name.
- But if we want to sort salary in increasing order then sorting records by name is not a good file organization. It should be sorted on salary.
- This chapter deals with logical and physical files and different types of file organization techniques.

1.2 LOGICAL AND PHYSICAL FILES

1.2.1 What is File?

- In any information system, we deal with data.
- This data has to be arranged in a proper way to accept, process and communicate operations and results.
- For arranging the data, we need files.
- A manual file stores all the information relating to a particular activity.
- For example: inventory activities in an inventory file, payroll activities in a payroll file and so on.

- The basic unit of information for computer and manual files is a record.
- A collection of related data items form a record. For example: each employee's record will contain data items such as Employee number, Employee name, Basic pay, Allowances, Deductions, Gross pay, Net pay.
- A set of logically related records form or constitute a file.
- Fig. 1.1 shows how a file is constituted.

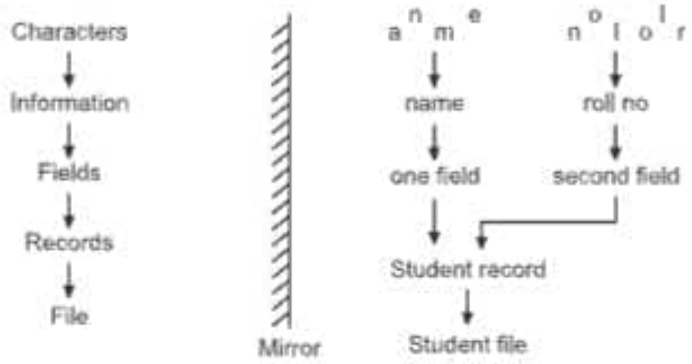


Fig. 1.1: How to constitute a file

1.2.2 File Structure

- To learn file structure, one must understand the basic terms used to describe the file hierarchy; the terms are explained below:

(a) Character or byte:

- A bit is the smallest unit of data representation (value of a bit may be a 0 or 1). Eight bits make a byte which can represent a character or a special symbol in a character code.

1 character = 1 byte.

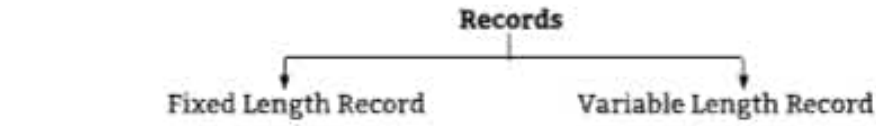
(b) Data Item:

- One or more characters combined may form a data item.
- It is used to describe an attribute of an object or entity.
- For example: student_no, student_name age, etc. are data items.
- A data item is also referred to as a *field*. However, there is a slight difference between data item and field.
- A field is a physical space on a magnetic disc whereas a data item is the data stored in the field.

(c) Record:

- The data items related to an object or entity are grouped into a record.
- Record can also be defined as a set of logically related fields.
- There are two types of records: 1. Fixed length. 2. Variable length.

- In a fixed length record, every occurrence of the record must have each of the fields present and a given field need to be the same length from record to record. This means each occurrence of a record in a file is the same or of a fixed length.
- In variable length record, every occurrence of a record need not have each of the fields present and a given field need not be the same length from record to record.
- This means, each occurrence of a record in a file is not the same. Fig. 1.2 shows types of records.



Student record

Roll no	Name
01	Samir
04	Ramesh

Customers record

Cust name	Loan taken	Loan details
Sandeep	Y	*
Leena	N	–

* Loan detail includes the loan amount, bank from which loan is taken, EMI, no. of years.

Fig. 1.2: Types of Records

(d) File

- File is a set of logically related records. Almost all information stored in a computer must be in a file. There are many different types of files: data files, text files, program files, directory files, and so on.

1.2.3 Logical and Physical Files (S-18)

- Files can be viewed as logical files and physical files.
- Logical file is a file, viewed in terms of what data items contains its record and what processing operations may be performed on the file. The user of the file will normally adopt such a view.
- Physical file is a file, viewed in terms of how the data is stored on a storage device and how the processing operations are made possible.
- In short, files can be considered to have a multilevel structure as shown in Fig. 1.3 (a) and Fig. 1.3 (b) shows the logical and physical files.

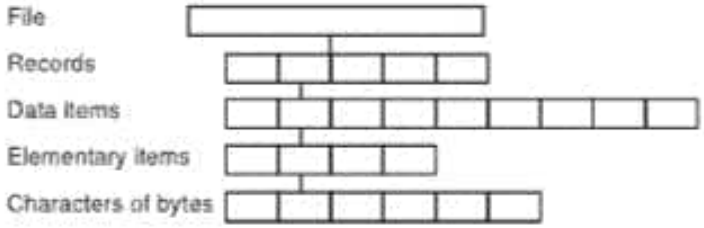


Fig. 1.3 (a): File Structure

- From Fig. 1.3 (a), we have seen that file consist of records, records consists of data items (fields).
- Data items may contain elementary items.
- For example: If *Date* is a data item then its elementary items are month, date and year.

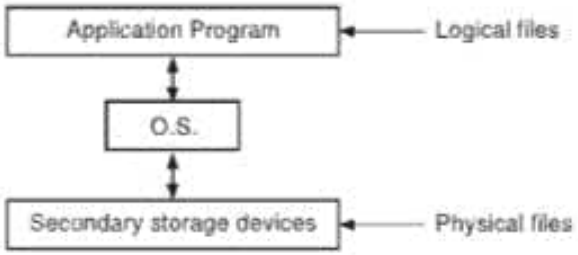


Fig. 1.3 (b): Logical and Physical Files

- The physical files are stored in secondary storage devices.
- The operating system makes a connection between logical and physical files for the application program.
- Application programs read or write the bytes from physical files that are stored on secondary storage like a disk.

1.3 BASIC FILE OPERATIONS (S-17)

- Once, we have logical file name, we need to declare what we want to do with the file. Basically, only two operations can be done, these are:
  (1) Open an existing file.
  (2) Create a new file, deleting the contents of the existing file.
- Once a file is created or opened, we can do operations like reading, writing and seeking. Let's see all these operations in detail.

1.3.1 Open File Operation (S-18)

- We can open the existing physical file. The program pointer is positioned at the beginning of the file and stored contents are not changed. (See Fig. 1.4).
- For example: picking up a phone to receive a call.

Fig. 1.4: File Pointer Position in Opened File

1) If we want to create a new file, it will open a blank file with the program pointer. Then we can write data into the file (See Fig. 1.5 a).

2) But if the same name of an available physical file is given, then the contents are removed first and the file opened is blank again (See Fig. 1.5 b).

- For example: picking up a phone to place a call.

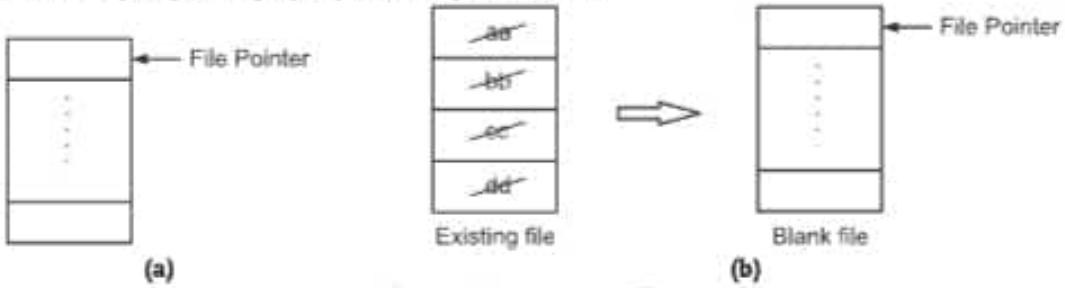


Fig. 1.5: Create a new file

1.3.2 Close File Operation

- In terms of telephone example, hanging up the phone is the most important job.
- Unless the phone hangs up, the phone line is not available for placing another call or receiving a call.
- This is as good as closing a file.
- Closing of file ensures that the buffer for that file has been flushed of data to a physical file.
- Operating system itself closes a file when program terminates normally.
- Closing a file is most important criteria for the protection of data against data loss.
- When you open a file, it is duplicated and the contents are taken into primary memory (Fig. 1.6) and when you close the file, the buffer area and memory allocated to that file is freed (See Fig. 1.7).

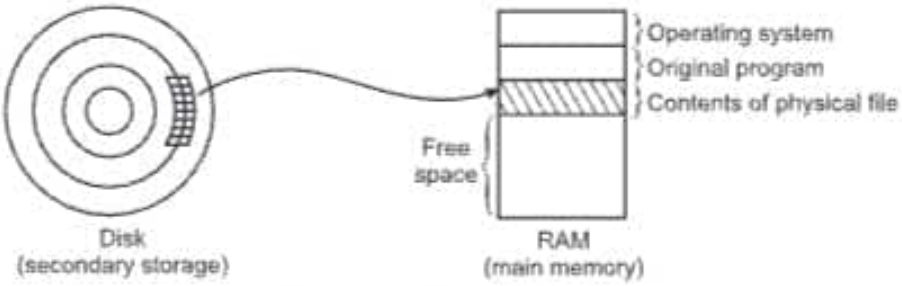


Fig. 1.6: Opening existing file

Fig. 1.7: Closing existing file

1.3.3 Reading and Writing File Operation

- Reading and writing are the fundamental operations performed on any file.
- For reading any record or content of a file, we need to open an existing file.
- For writing a record in a file, we can open a new file in write mode or an existing file in append mode.
- In a read operation, contents of a file are taken into the buffer area from a secondary storage device and then from buffer to user's working area of RAM i.e. primary memory. See Fig. 1.8.
- In a write operation, contents of user area or primary memory is taken into buffer and then wrote onto secondary storage device. See Fig. 1.9.

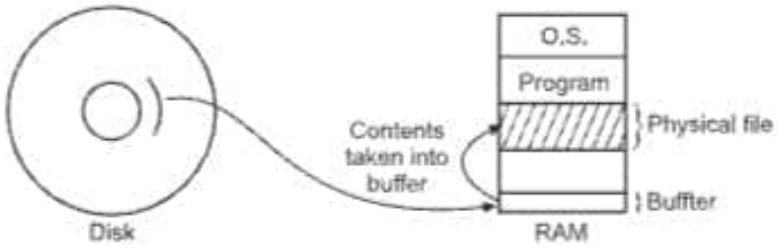


Fig. 1.8: Read operation

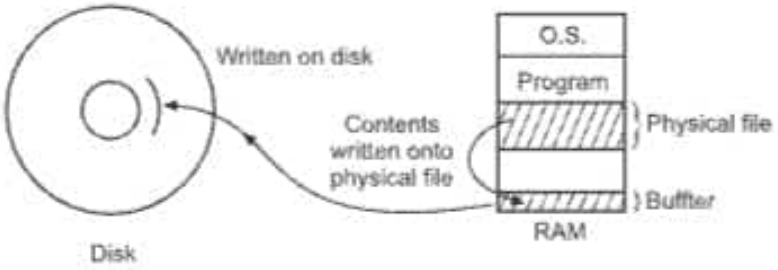


Fig. 1.9: Write operation

- In COBOL language, read and write statements are available for these operations whereas in C, C++ language, fread ( ) and fwrite ( ) functions are available to do the same.

1.3.4 Seeking File Operation

- Generally, we read through the file sequentially i.e. reading one byte after another until we reach the end-of-file.
- Every time a byte is read, the read/write pointer (file pointer) moves ahead to read the next byte.
- This is done by the operating system.
- Suppose there are 20 thousand bytes and after reading first byte we want to read say 19 thousand bytes without wasting time through sequential read.

- For this, we must be able to control the file pointer so that it is moved according to our need.
- The action of moving directly to a certain position in a file is called as Seeking.
- Generally, with seek, we require two arguments; source-filename and offset.

  seek (source-file, offset)
- Here, source file is logical filename and the offset will give number of position where a file pointer should move from the start of the file. In C, we have fseek( ) function to do seeking.
  - Operations on files are usually grouped into retrieval (locate certain records) operations and update (insertion or deletion of records or modification of field values) operations.
  - Actual operations for finding and accessing file records change from system to system. Following are a set of representative operations.
    (a) **Find next:** It searches for the next record in the file satisfying the search condition. Main memory buffer is checked for the existence of the block containing that record. If it is not in main memory buffer, then the block containing that record is transferred into main memory buffer. The record is located in the main memory buffer, which becomes the current record.
    (b) **Delete:** It deletes the current record and makes changes in the file on disk to reflect the decision.
    (c) **Modify:** It modifies some field values for the current record. It also makes changes in the file on disk to reflect the modifications.
  - All the above operations are known as record-at-a-time operations as they are applied to an individual record. Additionally, set-at-a-time higher level operations which may be applied to a file are given below:
    (a) **Find all:** It finds all the records in the file satisfying a search condition.
    (b) **Find ordered:** It retrieves all the records in the file in some specific order.
    (c) **Reorganize:** It starts the reorganization process. For example, by sorting the file records on a specified field we can reorder the file records.

1.4 FILE ORGANISATION [S-17]

1.4.1 Fields and Record Structure in File

- Data is usually stored in the form of records. Each record consists of a collection of related data *values* or *items*, where each value is formed of one or more bytes and corresponds to a particular *field* of the record.
- Record usually describes entities and their attributes.
- For example, an EMPLOYEE record represents an employee entity and each field value in the record specifies some attribute of that employee such as NAME, BIRTH-DATE, SALARY, etc.

- A collection of field names and their corresponding data types consist of a record type or record format definition.
- A *data type* associated with each field, specifies the type of values a field can take.
- The data type of a field is usually one of the standard data types used in programming. These include numeric (integer, long-integer or real number), string of characters (fixed- length or varying), Boolean (having 0 and 1 or True and false values only), and sometimes specially coded data and time data types.
- The number of bytes required for each data type is fixed for a given computer system. An integer may require 4 bytes, long integer 8 bytes, real number 4 bytes, a Boolean 1 byte, a date 4 bytes (to code the date into an integer), and a fixed length string of K characters K bytes.
- Variable length string may require as many bytes as there are characters in each field value.
- **BLOB:** In recent database applications, the need may arise for storing data items that consist of large unstructured objects, which represent images, digitized video or audio streams or free text. These are referred to as BLOBs (Binary Large Objects). Normally, a BLOB data item is stored separately from its record in a pool of disk blocks and a pointer to the BLOB is included in the record.
- There are four common methods to add fields into the file:
  1. Force the fields into a predictable length.
  2. Begin each field with a length indicator.
  3. Place a delimiter at the end of each field to separate it from the next field.
  4. Use a "keyword = value" expression to identify each field and its contents.
- **For example**, in C Programming:

```
Struct Person {
        char last [10];
        char first [10];
        char addr [15];
        char city [15];
        int zip [6];
};
```

- In this example, each field is a character array that can hold a string value of some maximum size.
- This is fixed-size field structure where structure Person can be stored in (10 + 10 + 15 + 15 + 6) 56 bytes.
- Another way to make it possible to count to the end of a field is to store the field length just ahead of the field (Fig. 1.10 (b)).
- If the fields are not too long, then it is possible to store the length in a single byte at the start of each field.

- The choice of delimiter is another way to separate the fields.
- We can use white-space characters (blank, newline, tab) as a delimiters because they provide a clean separation between fields, which is shown in the Fig. 1.10 (c).
- Fig. 1.10 (d) shows the structure in which a field provides information about itself. Such self-describing structures can be very useful tools for organizing files in many applications.

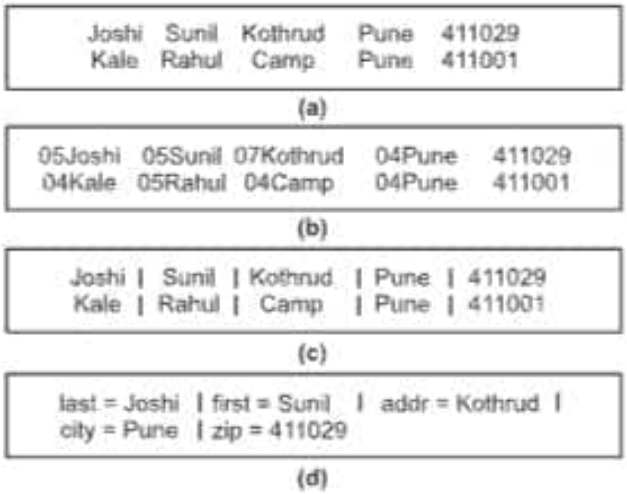


Fig. 1.10: Each Field is Identified by Keyword

1.4.2 Record Types

- We have already introduced two types of record in section 1.2.2. In this section, we will see it in more detail.
- A file is a collection of records. Mostly all records in a file are of the same record type.
  - The file is said to be made up of **fixed-length records**, if every record is equal in size (in bytes).
  - The file is made up of **variable-length records**, if different records in the file do not match in terms of size (in bytes).
- Reasons for having variable length records in a file:
  - The file records belong to one record type, but one or more of the fields may have multiple values for individual records, such a field is called a repeating field. A group of values for the repeating field is called a repeating group.
  - The file records belong to one record type, but one or more of the fields are optional.
  - The file contains records of heterogeneous record types. This will happen if related records of heterogeneous types are placed together on disk blocks. For example, the Sales_Report records of a particular Product may be placed following the Product's record.

- Fig. 1.11 shows different record storage formats.

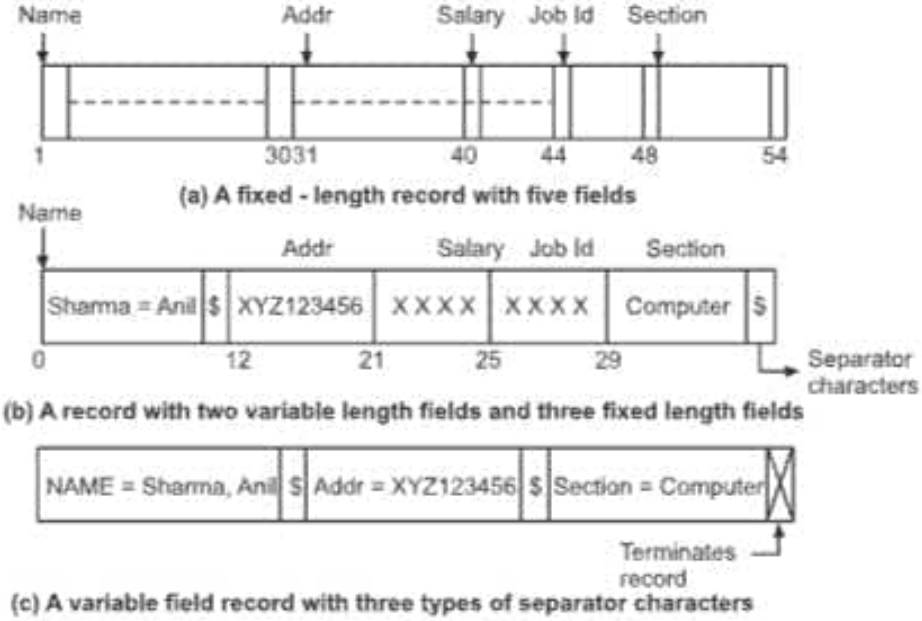


Fig. 1.11: Different Record-Storage Formats

- Fig. 1.11 (a) shows fixed-length Employee records with a record size of 54 bytes. Because of the fixed field lengths, we can identify the starting byte position of each field relative to the starting position of the records as shown in Fig. 1.12.

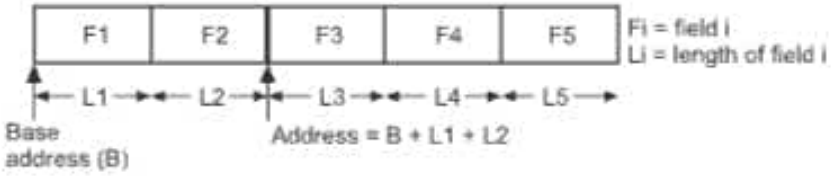


Fig. 1.12

- A file having variable length records can be represented as a fixed-length records file.
- For example, in every file records optional fields are part of it with a special null value stored in it, if there is no value for that field. A null value is used to denote that the value is unknown or inapplicable or missing. A null value is different than a zero or a field containing spaces.
- In fixed-length record, space is wasted when some records do not have values for all the physical spaces given for each record.
- For variable-length fields, we cannot guess the exact length of some field values. To calculate exactly the bytes of a field, we use special separator characters called delimiters. (e.g. ? or % or $). These characters are just used to indicate end of variable length fields [Fig. 1.11 (b)].

- Some fields in the records are optional. It means that they have certain value when it is required, otherwise they have null value.
- A file or records with optional fields can be formatted in different ways.
- A record with optional fields can be represented as a sequence of <field-name, field-value> pair rather than field values.
- As shown in Fig. 1.11 (c), three types of delimiters are used: (i) '=' is used to separate field name from field value, (ii) '$' is used for separating the fields, (iii) ⊠ is used for terminating record.
- A more practical option is to give field type to each field and include in each record as a sequence of *<field-type, field-value>* pair.
- A repeating field needs two separator characters:
  (1) To separate the repeating values of the field,
  (2) To indicate termination of the field.
- Finally for a file having records of heterogeneous types, before each record a record type indicator is added.

1.4.3 Types of File Organizations

- In this section, we present simplified analysis of *three* basic file organizations: files *sorted* on some field, files that are *hashed* on some fields and indexed file organization.
- Our objective is to emphasize the importance of choosing an appropriate file organization.

1.4.3.1 Sequential Files [S-18]

- We can physically arrange the records of a file on disk based on the values of one of their fields – called the *ordering field*.
- This leads to *an ordered or sequential file*. If the ordering field is also a *key field* of the file a field definitely to have a unique value in each record then the field is also called the *ordering key* for the file. Fig. 1.13 shows an ordered file with NAME as the ordering key field (assuming that employee has distinct names).

Ordered records have some advantages over unordered files as follows:

1. For reading the records in order of the ordering field values becomes efficient, since no sorting is required.
2. For finding the next record from the current one in order of the ordering field usually requires no additional block accesses, as the next record is in the same block as the current one (unless the current record is the last one in the block).
3. For using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used.

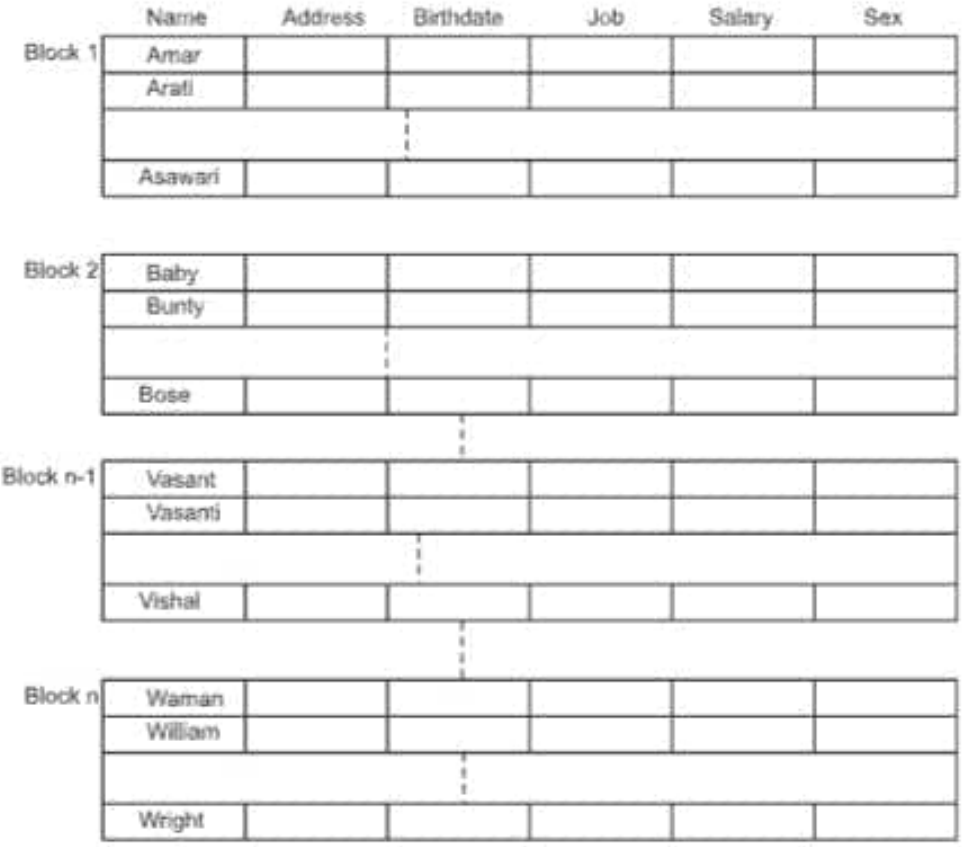


Fig. 1.13: Some blocks of an ordered (sequential) file of EMPLOYEE record with NAME as the ordering field

Insertion and Deletion:

- Inserting and Deleting records are expensive operations for an ordered file because the records must remain physically ordered.
- To *insert* a new record, we must find its proper position in the file, based on its ordering field value. Then make space in the file to insert the record in that position.
- For a large file, this can be very tedious task because on average, half the records of the file must be moved to make space for the new record.
- This means that half the file blocks must be read and rewritten after records are moved among them.
- If we use deletion markers, then record deletion problem is less difficult and file is reorganized periodically.
- One option for making insertion more efficient is to keep some unused space in each block for new records. However, once this space is used, the original problem again occurs.
- Another method is used to create a temporary unordered file termed as an **Overflow** or **Transaction file**. With this technique, the actual ordered file is termed as the **Main** or **Master file**.

- New records are inserted at the end of the transaction file rather than in main file. Periodically, the transaction file is merged with the master file during file reorganization. Insertion becomes very efficient, but at the cost of increased complexity in the search algorithm.
- The overflow or transaction file must be searched using a Sequential search if (after the binary search) the record is not found in the main file.
- Modifying a field value of a record depends on two factors, the search condition to find the record and the field to be modified.
- If the search condition involves the ordering key field, we can find the record using a binary search, otherwise we must do a Sequential search.
- A non-ordering field can be modified by changing the record and rewriting it in the same physical location on disk by assuming that records are fixed-length records.
- Modifying the ordering fields signifies that the record can change its position in the file, which needs deletion of the old record followed by the insertion of the new modified record.
- Reading the file records in order of the ordering field is quite efficient. To include the records in overflow, we must merge them in their proper positions. In this case, we can first reorganize the file, and then read its blocks sequentially.
- To reorganize the file, we first sort the records in the transaction file, and then merge them with the master file. The records marked deleted are removed during the reorganization process.
- Ordered files are rarely used in database applications, unless an additional access path, called a *primary index* which is included with the file.

Disadvantages of Sorted File Organization:

- If the file is very large, then insertion is time-consuming.
- **Difficulties in searching:** If file grows, then searching becomes inefficient, for any search, we need to start reading a file from the beginning and continue till the end or until desired record is found.
- **Problem with record deletion:** To understand this problem, let us consider a situation where the librarian must maintain a number of cards equivalent to the number of books, eventhough the book is lost. Thus, if there are 5000 books in the library, there must be 5000 cards. Thus, the card must be maintained even if the book is lost. Perhaps, the librarian put deletion marker (say *) on the card to indicate card is lost. In any case, the librarian must not remove the card. This happens in a sorted file organization. When we want to remove a record from a file, the file cannot crunch itself automatically or we cannot reduce a file size by one record. Also we cannot reclaim the space freed by the deletion of this record.
- In general, sorted file organization is used if the number of records is small. However, if the number of records as well as the number of queries is large, it becomes inefficient.

1.4.3.2 Indexed Files

- An index is a data structure that organizes data records on disk to optimize certain file operations.
- An index allows us to efficiently search or retrieve all records. Using an index, we can achieve a fast search of data records.
- **Example:** Suppose we have books records. We can store the records in a file organized as an 'index on book_no'.
- In addition to that, we can create additional index file based on bookname, to speed up queries involving bookname. First Index file contains all data entries of book_no. and second index file contains all data entries that allow us to locate book records satisfying a query on bookname.
- Using index we can find the desired entry and then use these to obtain data records. A data entry with search key value k contains enough information to locate (one or more) data records with search key value k.
- In order to create and maintain index files, a computer creates a data file and an index file. The data file contains the actual contents (data) of the record and the index file contains the index entries. The one field in a file is the primary key, which identifies a record uniquely.
- In the following ways, the files are organized:
  1. The data file is stored in the order of the primary key values.
  2. The index file contains two fields: the key value and the pointer to the data record.
  3. One record in the index file thus, consists of a key value and a pointer to the corresponding data record. The pointer points to the first entry within the range of data records.

Example:

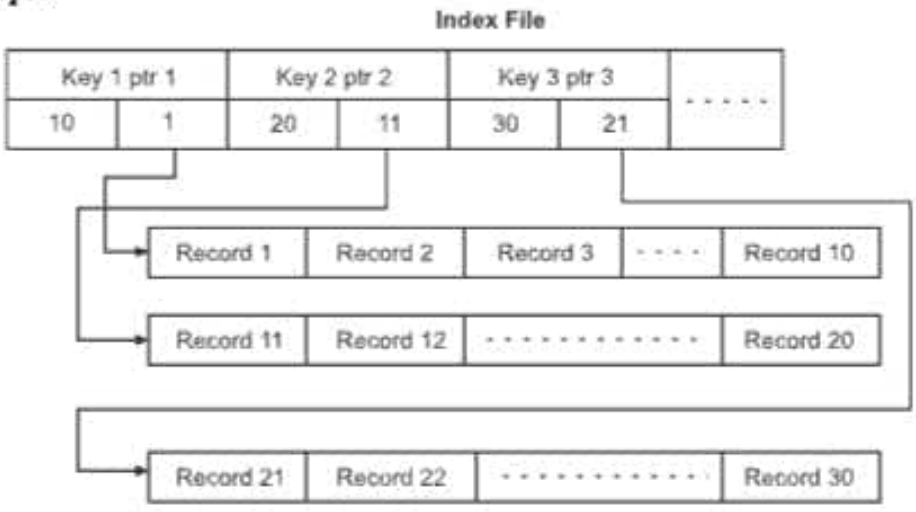


Fig. 1.14: Index File Organization

- Generally the key value is the largest primary key value in a given range of records.
- In Fig. 1.14, the first index entry is 10, which is highest primary key value in the first data block of 1 to 10. The pointer from this index entry points to the start of this range i.e. 1.

Advantages:

1. Data can be accessed directly and quickly.
2. Data maintained centrally and it kept up-to-date.
3. Primary and secondary index can be used to search the data.

Disadvantages:

1. If we want to insert new index values between any two existing values, then it becomes difficult.
2. If index values become too high, then searching becomes slow.
3. The use of an index lowers the computer efficiency.
4. Hardware required for these systems is expensive as data is stored on disk.
5. File is updated directly.
6. Backup should be taken regularly.

- We can solve this problem using multi-level index. Multi-level index contains multiple levels of indexes, the final level pointing to the data items.
- The advantage of multi-level index is that we can create multiple levels of indexes, accommodating as many index entries as desired.
- This leaves enough space for future insertions.

1.4.3.3 Hashed Files

- In hashed files, the record number itself becomes an equivalent of the key value or primary key.
- The term hash indicates splitting of a key into pieces. The computer cuts the key into pieces so as to avoid wastage of space.
- Hash file organization provides very fast access to records on certain search conditions. This is usually called a **hash** or **direct file**.
- The idea behind hashing is to provide a function h, called a **hash function** or **randomizing function**, i.e. applied to the hash field value of a record and yields the address of disk block in which the record is stored.
- A search for the record within the block can be carried out in the main memory buffer.

(i) Internal Hashing:

- For internal files, hashing is typically implemented through the use of an array of records. Suppose that the array index range is from 0 to M - 1 [Fig. 1.15 (a)], then we have M *slots* whose addresses correspond to the array indexes.

- We choose a hash function that transforms the hash field value into an integer between 0 and M – 1. One common hash function is the h(K) = K mod M function, which returns the remainder of an integer hash field value K after division by M, this value is then used for the record address.
- Non-integer hash field values can be transformed into integers before the mod function is applied.

  For example:

  N = Number of records in the file

  K = Set of keys that can uniquely identify all the records in the file

  Hash function H(K) = K mod M

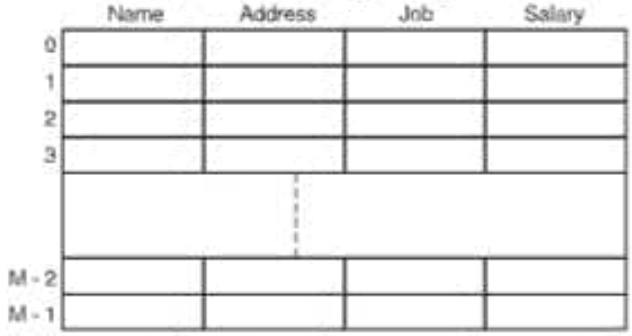
  If K is 9875, N is 58 and M is 99, then we have,

  $$H(K) = 9875 \bmod 99 = 74$$

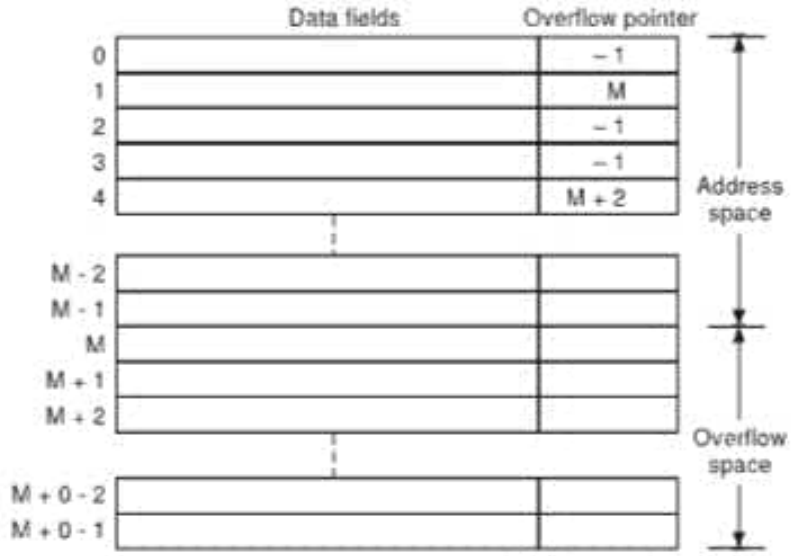
- A *collision* occurs when the hash field value of a new record that is being inserted hashes to an address that already contains a different record.
- In this situation, we must insert the new record in some other position, since its hash address is occupied.
- The process of finding another position is called *collision resolution*. There are numerous methods for collision resolution, including the following:
  - **(a) Open addressing:** Proceeding from the filled position specified by the hash address, the program checks the following positions in sequence until an vancant (empty) position is found.
  - **(b) Chaining:** In this method, various overflow locations are kept, usually by expanding the array with a number of overflow positions. In addition, a pointer field is added to every record location. A collision problem is handled by placing the new record a vacant overflow location and setting the pointer of the occupied hash address location to the address of that overflow location. A linked list of overflow records for each hash address is maintained, as shown in Fig. 1.15 (b).

(ii) Multiple hashing:

- The program applies a second hash function if the first finally results in a collision. If again collision results, the program either uses open addressing or applies a third hash function and then uses open addressing if required.



(a) Array of M positions for use in Internal Hashing



(b) Collision Resolution by Chaining Records

Fig. 1.15: Internal Hashing Data Structure

- The goal of a good hashing function is to distribute the records uniformly over the address space such that collisions are minimized while not leaving many vacant locations.

(iii)External Hashing:

- Hashing for disk files is known as **External Hashing**. The address space of disk is divided into buckets, each of which holds multiple records.
- A bucket is either one disk block or cluster of contiguous blocks. The hashing function maps a key into a corresponding bucket number, rather than assigning an absolute block address to the bucket.
- A table is maintained in the file header which transforms the bucket number into the corresponding disk block address as shown in Fig. 1.16

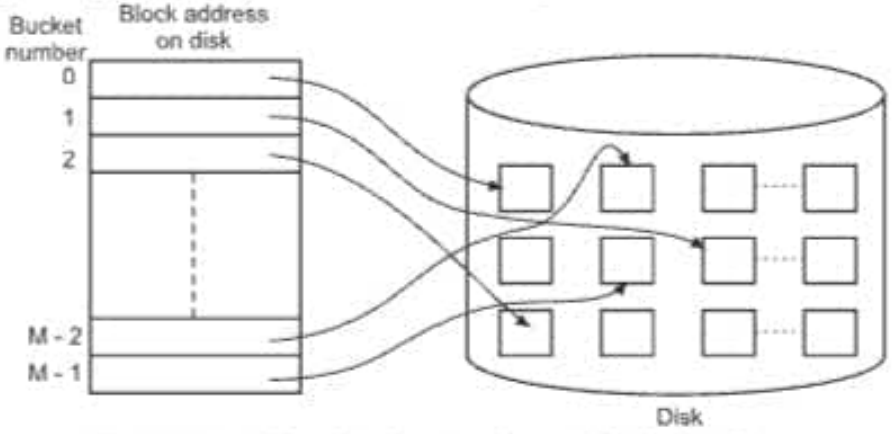


Fig. 1.16: Matching of Bucket Number to Disk Block Address

- In every main bucket, there must be 3 entries.

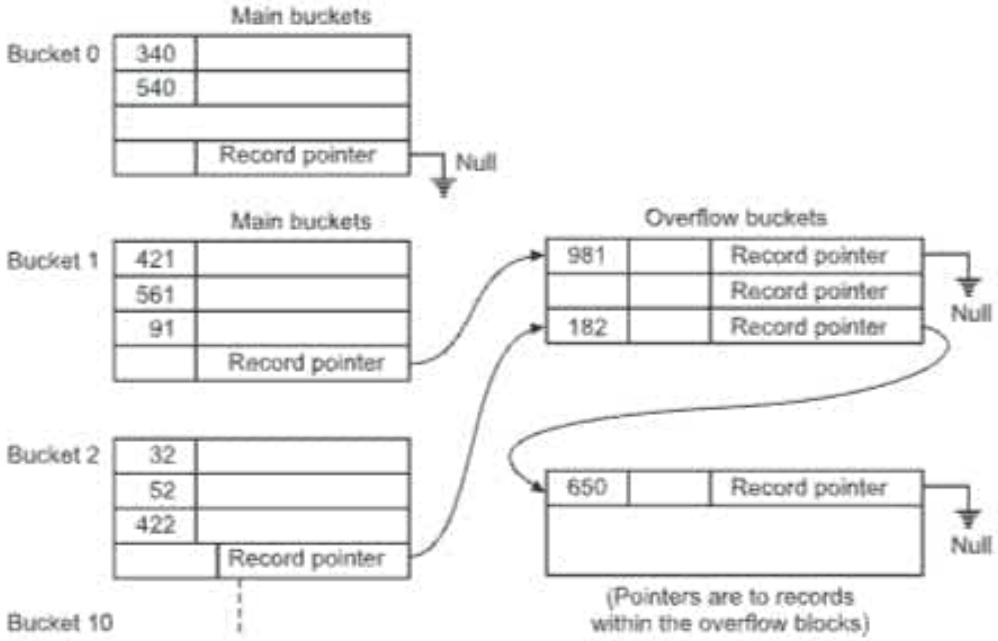


Fig. 1.17: Handling Overflow for Buckets by Chaining

- The collision problem is less with buckets, because as many records as will occupy in a bucket can hash to the same bucket without causing problems.
- If bucket is full, we can use a variation of chaining in which we maintain a pointer in each bucket to a linked list of overflow records for the bucket as shown in Fig. 1.17.
- The pointers in the linked list must be record pointers, which contains both a block address and a corresponding record position within the block.
- The drawback of hashing is the fixed amount of space allocated to the file.

1.5 INDEXING

- An index for a file in database system works in the same way as the index of the textbook.
- A textbook index lists important terms at the end of the book in an alphabetical manner. If we are interested in a topic, we can search for it in the index at the end of the book, find the pages where it occurs, and ultimately read the pages to find the information.
- Indexes are used to improve the performance of the database system. E.g. catalogue of a library.
- If we are searching for a book written by a particular author, we search it in the author catalogue. The card in the catalogue gives us the exact location of book we are searching for.

- The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value.
- The index file is comparatively smaller than the data file; hence the use of the binary search method is efficient.

1.5.1 When to use Indexes ?

1. Indexes are used when the database is very large.
2. Searching is very fast if indexes are present.
3. Sorting data is also easy when indexes are used.
4. Storage space required at runtime (in Main Memory) is very less if we have indexes.

1.5.2 Types of Indexes

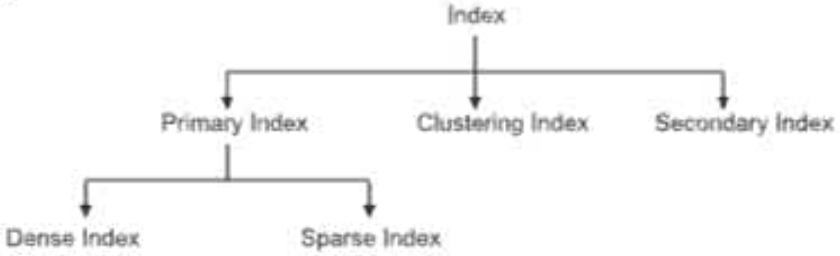


Fig. 1.18: Types of Indexes

- There are several types of ordered indexes. Let us see them in detail.
- **Primary Index:** A *primary* index is an index specified on the ordering key field of an ordered file of records. Every record has unique value for that field.
- **Clustering Index:** If the ordering field is not a key field (i.e. if several records in the file can have the same value for the ordering field) then a *clustering index*, can be used. Remember that a file can have at most one primary index or one clustering index, but not both indexes.
- **Secondary Index:** *Secondary index*, can be specified on any non-ordering field of a file. A file can have several secondary indexes in addition to its primary access method.
- Here we assume that all files are sequentially ordered and thus have a primary search key.
- Such files, together with a primary index are called *index-sequential files*.

  They are one of the oldest index schemes used in database systems.
- They are designed for applications that requires both sequential processing of the entire file and random access to individual records.

1.5.2.1 Primary Index (S-18, W-16)

Fig. 1.19 shows a sequential file of *deposit* records for the bank and the *branch-name* of a bank is the search key.

	branch_name	acc_no	cust_name	balance
	Blackbum	A217	Johnson	750
	Derby	A101	Jack	500
	Derby	A110	Alex	935
	Kirkcaldy	A215	Lalonde	415
	Paris	A102	Roger	527
	Paris	A201	Smith	635
	Paris	A218	Nill	815
	Sweden	A222	Sam	240
	Sydney	A305	Van	751

Fig. 1.19: Sequential File for Deposit Record

- The above file contains records having fields **branch_name, account_number, customer_name and balance**.
- In order to allow fast random access, an index structure is used. There are two types of index that may be used for index files.
  1. **Dense index:** An index record appears for every search-key value in the file. The record contains the search-key value and a pointer to the first data record with that 'search-key' value.
  2. **Sparse index:** Index records are created for only some of the records. To locate, we find the index record with the largest search-key value that is less than or equal to the search-key value we are looking for. We start at the record pointed by that index record and follow the pointers in the file until we find the record that we are interested in.
- Fig. 1.20 and 1.21 show dense and sparse indices, respectively, for the deposit file.

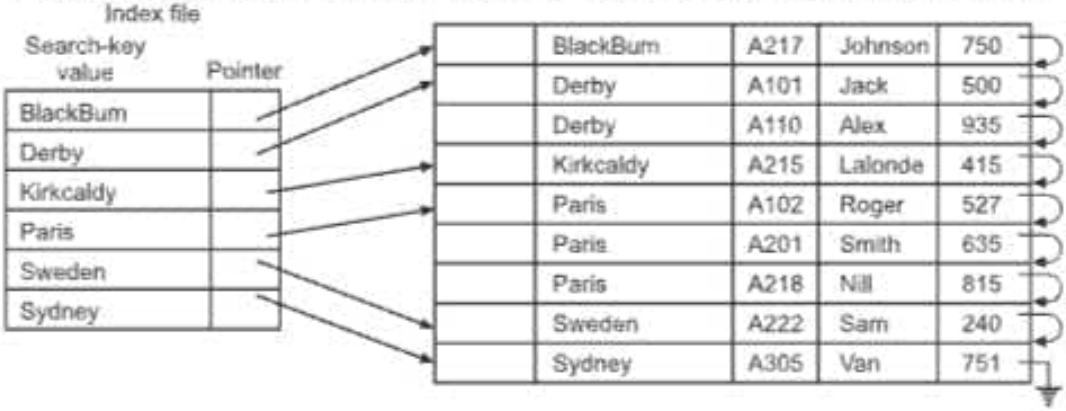


Fig. 1.20: Dense Index

- For example, we are looking up records for the *Paris* branch. Using the dense index of Fig. 1.20, we follow the pointer directly to the first *Paris* record.
- We process this record, and follow the pointer in that record to find the next records in search-key (*branch_name*) branch other than Paris.

- If we are using the sparse index (Fig. 1.21), we do not find an index entry for *Paris*.
- Since, the last entry (in alphabetic order) before *Paris* is "*Kirkcaldy*".
- We follow that pointer then read the *deposit* file in sequential order until we find the first *Paris* record and begin processing at that point.

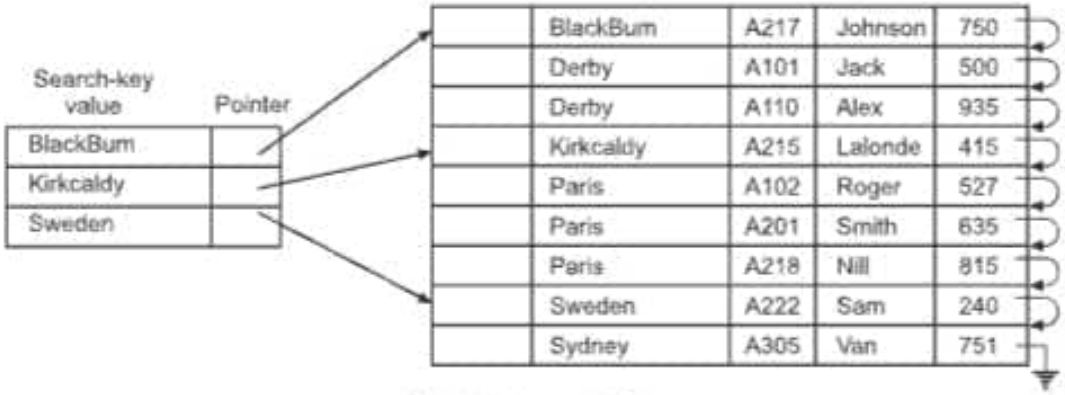


Fig. 1.21: Sparse Index

- As we have seen, it is generally faster to find a record if we have a dense index rather than a sparse index.
- Sparse indices have an advantage over dense indices in that they require less space.
- Even if we use a sparse index, the index itself may become too large for efficient processing.
- It is not unreasonable, in practice, to have a file with 100,000 records, with 10 records stored in each block.
- If we have one index record per block, the index has 10,000 records. Index records are smaller than data records, so let us assume 100 index records fit on a block. Thus, our index occupies 100 blocks.
- If an index is sufficiently small to be kept in main memory search time is low. However, if the index is so large that it must be kept on disk, a search results in several disk block reads.
- If the index occupies b blocks and binary search is used, we may read as many as $1 + \log_2(b)$ blocks. For our 100-block index, this means 7 block reads.
- Note that if overflow blocks have been used, binary search will not work. In that case, a sequential (linear) search is typically used which requires 'b' block read. Thus, the process of searching the index may be costly.
- To handle with this problem, we construct a sparse index on the primary index, as shown in Fig. 1.22. To search a record, we first use binary search on the outer index to find the record for the largest search-key value less than or equal to the one we are looking for. The pointer points to a block of the inner index. We search this block until we find the record which has the largest search-key value less than or equal to the one we are looking for. The pointer in this record points to the block of the file that contains the record for which we are looking.

- By using two levels of indexing, we have read only one index block instead of 7 blocks, if we assume that the outer index is already in main memory.
- If file is huge in size, even the outer index may grow too large to fit in main memory. In such a case, we can create another level of index.
- We can repeat this process as many times as required. In practice, two levels are sufficient. Frequently, each level of index corresponds to a unit of physical storage.
- Every index must be updated whenever, a record is either inserted into or deleted from the file. We describe algorithms for updating single-level indices below:

(i) Deletion: In order to delete a record, it is necessary to see the record to be deleted. If the deleted record was the last record with its specific search-key value, then we delete the search-key value from the index. For dense indices, we delete a search-key value for record deletion. For sparse indices, we delete a key value by replacing its entry in the index (if one exists) with the next search-key value (in search-key order). If the next search-key value already has an index entry, we delete the entry.

(ii) Insertion: First see the search-key value appearing in the record to be inserted. If the index is dense, and the search-key value does not appear in the index, insert it. If the index is sparse, no change needs to be made to the index unless a new block is created. In this case, the first search-key value (in search-key order) appearing in the new block is inserted into the database.

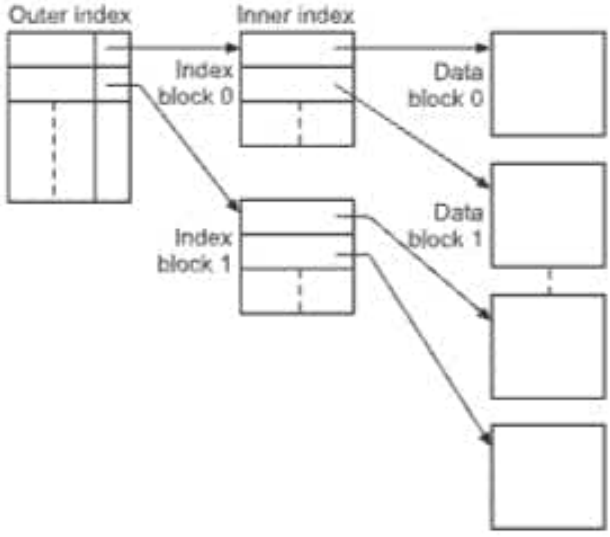


Fig. 1.22: Two-level Sparse Index

1.5.2.2 Clustering Index

- If records of a file are physically ordered on a non-key field that does not have a different value for each record, that field is called the **clustering field**.
- We can create a different type of index, called a **clustering index**, to speed up retrieval of records that have the same value for the clustering field.

- A clustering index is also an ordered file with two fields; the first field is of the same type as the clustering field of the data file and the second field is block pointer.
- There is one entry in the clustering index for each distinct value of the clustering field. Insertion and deletion cause a problem since, the data records are physically ordered.
- So, it is common to reserve a complete block for each value of the clustering field, all records with that value are placed in the block. If more than one block is required to store the records for a particular value, additional blocks are allocated and linked together.

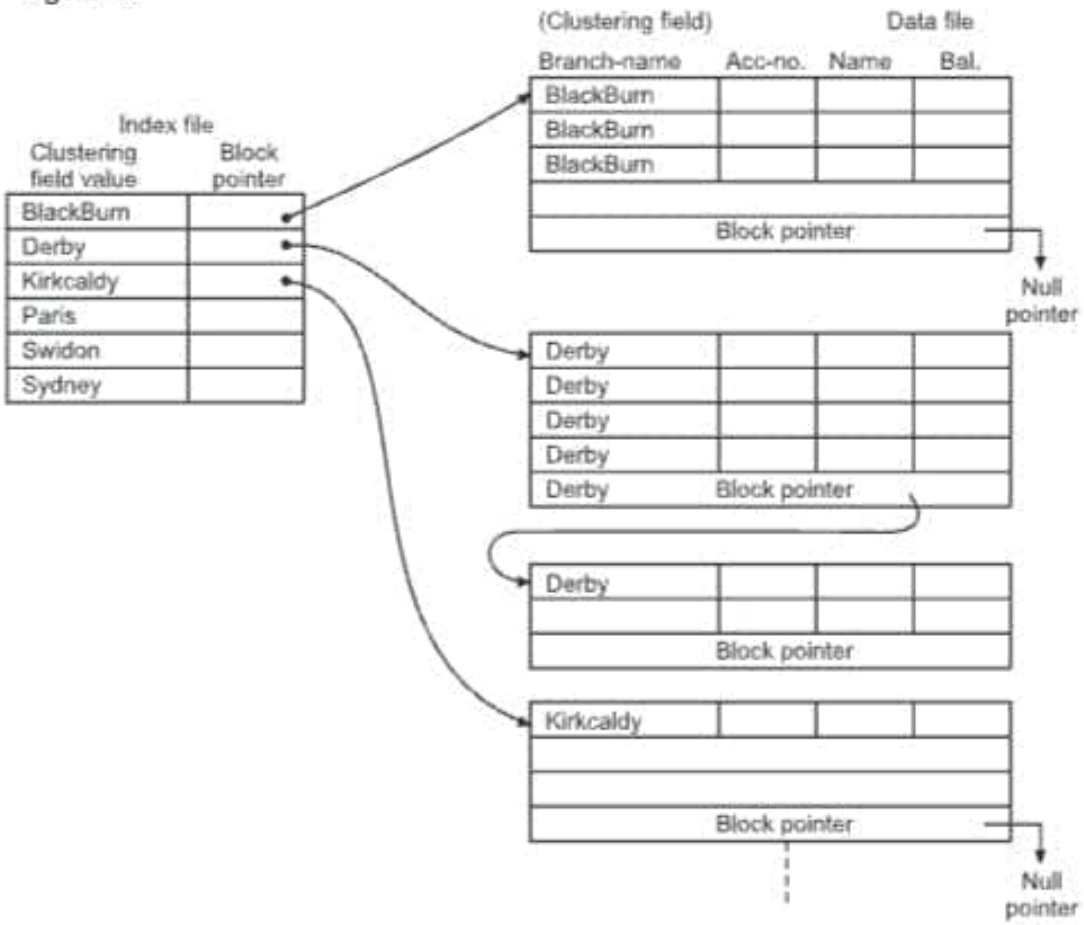


Fig. 1.23: Clustering Index

- In above figure clustering index with separate blocks for each group of records that share the same value for the clustering field
- A data file can be clustered on at most one search key, which means that we can have atmost one clustered index on a data file.

- An index that is not clustered is called an *unclustered* index; we can have several unclustered indexes on a data file.
- Suppose that student records are sorted by age, an index on age that stores data entries in sorted order by age in a clustered index. If in addition, we have an index on the birth-date field, the latter must be an unclustered index.

1.5.2.3 Secondary Index

- In a standard index-sequential file, only one index is maintained. If we choose to include several indices on different search keys, the index whose search key specifies the sequential order of the file is the primary index.
- The other indices are called secondary indices. The search key of a primary index is usually the primary key.

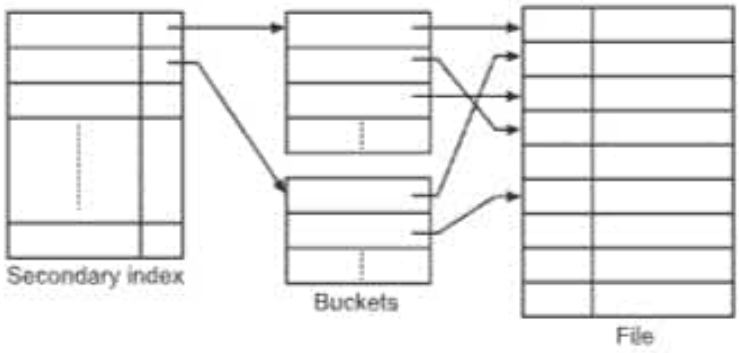


Fig. 1.24: Secondary Index

- Secondary indices may be structured differently from primary indices. Fig. 1.24 shows the structure of a secondary index. It uses an extra level of indirection.
- The pointers do not point directly to the file. They point to a bucket which contains pointers to the file. A bucket is a collection of one or more blocks chained together.
- This approach allows all the pointers for one secondary *search-key* value to be stored together. This is useful in certain types of queries for which we may do considerable processing using only the pointers. For primary keys, we can obtain all the pointers for one primary search-key value using a sequential scan.
- A sequential scan in primary-key order is efficient because records are stored physically in an order that approximates primary-key order.
- However, we cannot store a file physically in both primary-key order and in key order based upon a secondary key. If we attempt to scan the file sequentially, secondary-key order and physical-key order will differ,. In secondary-key order, the reading of each record is likely to require the reading of a new block from disk.
- By storing pointers in a bucket as shown in Fig. 1.25, we eliminate the need for extra pointers in the records themselves and eliminate the need for sequential scans in secondary key-order.

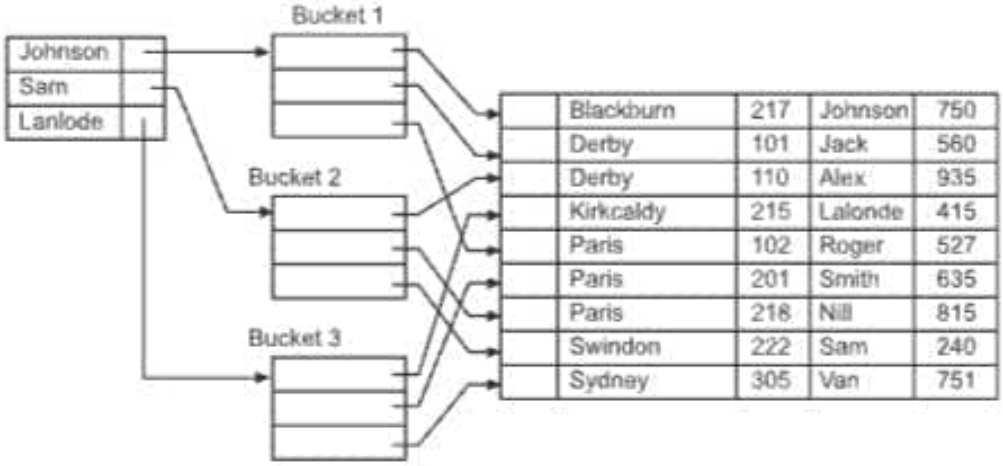


Fig. 1.25: Secondary Index on 'Customer-Name'

- It is desirable to use dense rather than sparse secondary indices. Consider Fig. 1.25 index on secondary-key *customer-name*.
- If we are performing retrieval on Alex, we must read all three records pointed to by entries in bucket 2. Only one entry points to a record for which customer_name value is Alex, get the records needed to be read. Since, the file is not ordered physically by customer_name, we expect this retrieval to require three disk block reads.
- By using a dense secondary index rather than a sparse index, we eliminate the need to read records with a secondary search-key value other than the one on which we are performing a lookup. Whenever, the file is modified, every index must be updated.
- Hence, secondary indices must be dense, with index entry for each search key value and a pointer to every record in the file.
- Secondary indices improve the performance of queries that use keys other than the search keys of the primary index.

Summary

- File organization refers to the way the data is stored in a file. File organization is very important because it determines the methods of access, efficiency, flexibility and storage devices to use.
- Logical file is a file viewed in terms of what data items its record contains and what processing operations may be performed on the file.
- Physical file is a file viewed in terms of how the data is stored on a storage device and how the processing operations are made possible.
- There are various operations which can be implemented on a file. For Example, Open, close, update file, Search record in the file, sort records in the file etc.
- An indexed file is a computer file with an index that allows easy random access to any record given its file key.

- In file organization, hash function is used to calculate the address of the block to store the records. The hash function can be any simple or complex mathematical function.

Check Your Understanding

Multiple Choice Questions:

1. A collection of related records is called .....
   (a) file (b) record (c) byte (d) field
2. Which of the following file operation modifies the file ?
   (a) insert (b) delete (c) modify (d) all
3. Student rollno, doctors name, part number, etc. are example of:
   (a) fields (b) primary key (c) unique (d) record
4. For quality selection i.e. selection of record on specified key, ... file organization is useful.
   (a) hash (b) sorted (c) heap (d) all
5. For range selection, ... file organization is useful.
   (a) hash (b) sorted (c) heap (d) all
6. In case of automatic teller machine (ATM) which file organization is suited best ?
   (a) indexed (b) sequential (c) both (d) none

Ans.: (1) a (2) c (3) a (4) a (5) b (6) c.

Practice Questions

Q.1 Answer the following questions in brief.

1. What is the effect of file size on the choice of File Organization ?
2. What are different types of records?
3. What is an index?
4. What is Primary Index.
5. Define logical and physical files.

Q.2 Answer the following questions.

1. Explain index file organization.
2. Explain field structure in brief.
3. Compare variable-length record and fixed length record.
4. Explain hashed file organization in brief.
5. Write short note on sequential file.
6. Write advantages and disadvantages of index file organization.
7. Write difference between dense and sparse index ?
8. Explain different types of indexes with example.

Q.3 Define the Terms:

1. Chaining, Open Addressing, Collision, File, Record, Delimiter, Ordering Field, Indexing, Clustering Field.

Previous Exams Questions

Summer 2015

1. What are logical and physical files. **4 M**

Ans. Refer to Section 1.2.3.

2. Explain object and physical files. **4 M**

Ans. Refer to Section 1.2.3.

3. Explain basic file operation. **4 M**

Ans. Refer to Section 1.3.

Winter 2015

1. What is file organization ? Explain the sequential file organization. **4 M**

Ans. Refer to Sections 1.4 and 1.4.4.

2. What do you mean by Indexing ? Explain sparse Indexing. **4 M**

Ans. Refer to Sections 1.5 and 1.5.3.1.

Summer 2016

1. What is file organization ? Explain the indexed file organization. **4 M**

Ans. Refer to Sections 1.4 and 1.4.4.2.

2. What is file ? Explain file structure. **4 M**

Ans. Refer to Sections 1.2.1 and 1.2.2.

3. What do you mean by indexing ? Explain dense indexing. **4 M**

Ans. Refer to Sections 1.5 and 1.5.1.

4. Explain the basic operations of files.

Ans. Refer to Section 1.3.

Winter 2016

1. Explain the Dense and Sparse Index. **4 M**

Ans. Refer to Section 1.5.3.1.

Summer 2017

1. Explain basic file operations. **4 M**

Ans. Refer to Section 1.4.3.

2. What are logical and physical files ? **4 M**

Ans. Refer to Section 1.2.3.

3. Explain sparse index. **4 M**

Ans. Refer to Section 1.5.3.1.

4. Explain Sequential file organization. **4 M**

Ans. Refer to Section 1.4.4.1.

Summer 2018

1. What are logical and physical files ? **4 M**

Ans. Refer to Section 1.2.3.

2. Explain sparse index. **4 M**

Ans. Refer to Section 1.5.3.1.

3. Explain Sequential file organization. **4 M**

Ans. Refer to Section 1.4.4.1.

❖❖❖

2... Database Management System

Learning Objectives...

- To understand basic concept of data.
- To know about DBMS and its applications and users.
- To study different views of data and data relationships.
- To learn data models.

2.1 INTRODUCTION

- In typical business environment, it is always essential to be able to produce the right information at the right time with minimum efforts.
- The database shows a change in storage, access and management of data.
- A DBMS (Database Management System) is a complex software system that is used to manage, store and manipulate data in the database.
- It is utilized by a large variety of users, to retrieve and manipulate data under its control.
- The users could be utilizing that database concurrently from on-line terminals and/or in a batch environment via application programs written in a high-level language.

2.2 BASIC CONCEPT AND DEFINITIONS

2.2.1 Data and Information

- Data can be defined as a representation of facts, concepts or instructions.
- Information is organized or classified data so that it has some meaningful values.
- Information is the processed data on which few decisions or actions can be taken.
- The processed data must have the following characteristics:
  - **Timely:** Information should be available when required.
  - **Accuracy:** Information should be accurate.
  - **Completeness:** Information should be complete.

2.2.2 Data vs Information

- The difference between Data and Information is shown here.

Data	Information
1. Data is in raw format.	1. After processing data we obtain information.
2. Data can be simple and at the same time unorganized.	2. Information is a set of data which is processed in a meaningful way.
3. Data is always interpreted to extract meaning. So data is meaningless.	3. When we obtain information after processing on data, it does not contain meaningless details.
4. It doesn't play any role in decision making.	4. It plays an important role in decision making.
5. **Example:** Sales done by a supermarket on a day	5. **Example:** Regionwise sales report. It gives an idea about the most profitable region.

2.2.3 Data Dictionary

- A data dictionary is called as metadata repository. As data dictionary is a collection of the descriptions of data objects, the database administrators use this to develop and maintain the database. Following is an example of data dictionary entry.

Table: Student

Field Name	Data Type	Other information
Roll_No	AutoNumber	Primary Key
Name	Text	Field width = 15
DtBirth	Date	Not Null

2.2.4 Data Item or Field

- A data item is a single unit of data in a stored record. Each data item or field has a certain data type.
- This term refers to the smallest possible unit of information.
- For example: In the previous table field names represent different data items Example: Roll_No.

2.2.5 Record

- A collection of different data items that provide information on an entity or a relationship is called a *record.*
- For example: 10, John, 12-4-2000 is a record of Roll_No = 10, Name = John & DtBirth = 12-4-2000.

2.3 DEFINITION OF DBMS

- In any organization, data is the basic resource needed to run the organization. This data is required by decision makers for processing and retrieving information.
- **Data:** A data is collection of information or real fact which can be recorded and have implicit meaning. Customer_name, item-price, balance etc. can be considered as data.
- **Database:** The database stores the information useful to an organization. A database is a shared collection of inter-related data, which is designed to fulfil the information needs.
- For example: Consider the name, telephone number and addresses of the people you know. This data is recorded in an indexed address book, or stored on a disk, or using a personal computer. This is a collection of related data with an implicit meaning and hence is a database.
- A database uses the following *implicit properties*.
  (i) A database represents some aspect of the real world, sometimes called the *miniworld*. Changes to the miniworld are reflected in the database.
  (ii) A database is a logical collection of data with some meaning. A random mixture of data cannot be termed as a database.
  (iii) A database is designed, built and populated with data for a specific purpose. It has an intended group of users.
- So, finally we can say that a database has some source from which data are derived, some degree of interaction with events in the real world, and an audience that is actively interested in the contents of the database.
- A **Database Management System** (DBMS) is a software system that allows the user to define, manipulate and process the data in a database, in order to produce meaningful information. The basic functions of DBMS are:
  1. To store data in a database.
  2. To organize the data.
  3. To control access of data.
  4. To protect data i.e. provide security.
- *Defining* a database involves specifying the data types, structures and strict control over the data to be stored in the database.
- *Constructing* the database means storing the data itself on some storage medium that is controlled by the DBMS.
- *Manipulating* a database includes such functions as querying the database to fetch specific data, modifying the database to reflect changes in the miniworld, and creating reports from the data.
- So in general, user can write programs or queries; while DBMS use the database stored on storage devices and gives meaningful information. The database system is illustrated in Fig. 2.1.

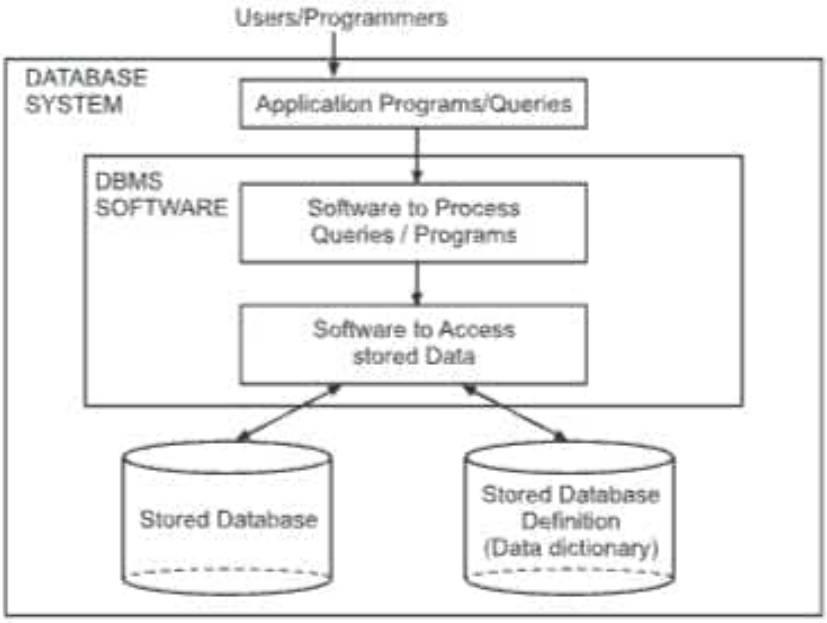


Fig. 2.1: Simple Database System

2.4 APPLICATIONS OF DBMS (S-17)

- DBMS is used for organizing, analyzing and modifying the information stored in a database.
- There are various applications available. These are listed below:
  - Computerized Library System.
  - Automated Teller Machines (ATMs).
  - Airlines (flight) Reservation System.
  - Inventory Management System.
  - Fashion designing.
  - Banking.
  - Universities – Registration, Grades.
  - Sales – Customer, Product, Purchases.
  - Manufacturing – Production, Inventory, Orders, Supply Chain and many more.

2.5 FILE PROCESSING SYSTEM VS DBMS

- In file processing system (Refer Fig. 2.2), the records are stored in separate files. Each file is called a *flat file*.
- To access the data from these flat files, various programs are written.
- If user wants simple change in the resultant database, lot of changes are needed in the application programs.

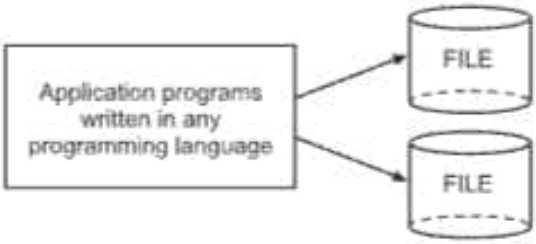


Fig. 2.2: File Processing System

- As system became more complex, file processing system lost its flexibility.

2.5.1 Limitations of File System

1. **Separated and Isolated data:** To make certain decision, a user might need data from different files. Because of the involvement of system analyst and programmer, files may have different formats.

   For example, salary of an employee stored as integer in one file and float in other file. Therefore, files were evaluated first and then relationships were determined. After that programs could be written to extract data from these files.

2. **Data Redundancy:** Suppose same type of information was stored in more than one file. This repetition of data caused loss of data integrity. Integrity means accurate and consistent data. For example, Address of an employee was stored in three different files. If change of address occurs and updated in one file only, then mismatch of information gives inconsistency to data.

3. **Difficulty in Data Access:** If the format of a certain record was changed, the code must be immediately updated, otherwise system provides incorrect data.

4. **Concurrent Access Anomalies:** In file processing system, once a file is opened by a user, it can not be used by another user, till first closes a file. It means sharing a file by various users at the same time is not possible.

5. **Security Problem:** One user 'A' does some changes and store them in a database. Second user also does some new changes and stores them in a database, and then the changes done by 'A' are lost. This is the security problem in file processing system.

- We can try to deal with this data management problem by storing the data in a collection of operating system files. This approach has the following drawbacks:

1. We probably do not have required memory to hold all the data; we must therefore store data in a storage device such as a disk or tape, and bring relevant parts into main memory for processing as needed.
2. Operating system provides only a password mechanism for security. This is not sufficiently flexible to provide security in which different users have permission to access different subsets of the data.

3. We have to write special programs to answer each question; that users may want to ask about the data. It increases the complexity since, large volume of data is present.
4. We must ensure that data is restored to a consistent state if the system suddenly fails while changes are being made.

2.5.2 Characteristics of DBMS

- A *DBMS* is a piece of software that is designed to make all above tasks easier. By storing a data in DBMS, rather than in files, we can use the DBMS features to manage the data in a robust and efficient manner.
- The major *components of DBMS* are as follows:

Applications:

- The user can write the queries or PL/SQL statements to get some result.

Transaction Management:

- A transaction is a set of database operations in a particular order.
- A transaction can update a record, delete a record or modify a record or a set of records.
- To save(store) these changes permanently in database, commit the change.
- If you do not want to save(store) the change permanently, roll back it.

Concurrency Control:

- It is database activity which co-ordinates the action of database manipulation.

Recovery Management:

- This ensures that aborted or failed transaction does not create any unfavourable effect on the database.

Security Management:

- The data should be protected against unauthorized access.
- Security management ensures that only authenticated users should have access to the database.
- It provides access privileges to users.

Language Interface:

- DBMS provide a language support for definition and manipulation of data.

Storage Management:

- DBMS provide this facility/service to store the data permanently in secondary storage device.
- It also interfaces with operating system to access the physical storage.

Characteristics of DBMS:

1. **Users file approach:** In file processing, each user defines and implements the files needed for a specific application. Hence more storage space is required. In DBMS, a single database is maintained that is defined once and is accessed by various users. DBMS software is not written for specific application.

2. **Self-describing nature of DBMS system:** A DBMS contains database as well as a complete definition of the database, which is stored in the system catalog. It contains information such as structure of each file, the type and storage format of each data item and various constraints of the data. The catalog is used by DBMS software and by database users who need information about database. For example, a company databases, a banking database, a university database. The database definition is stored in the catalog.

   In File Processing, data definition is part of the application programs.

3. **Isolation between Programs and Data:** DBMS access programs are written independently of any specific files. The structure of data files is stored in catalog separately from the access program (program-data independence). In file processing, if any change in structure of file is made then, all programs that access this file have to be changed. Suppose we want to add any field in record of file (say birth_date in student file), then program has to be changed in file processing.

4. **Multiple Views:** DBMS supports multiple views of the data, which a file cannot support. For example: one user is only interested for student mark list, other user is interested for courses attended by that student, these multi-user views are satisfied by DBMS.

5. **Data sharing:** Sharing of data is possible only in DBMS not in file processing.

2.5.3 Comparison of File Processing System and DBMS

File Processing System	Database Management System
1. PC-based small systems.	1. Mini-mainframe based large systems.
2. Relatively cheap.	2. Relatively expensive.
3. Less number of files used.	3. More number of files used.
4. Single user system.	4. Multiple user system.
5. Data redundancy and Inconsistency occur.	5. Data is independent and non-redundant.
6. Data access is difficult.	6. Data access is efficient.
7. Data is isolated.	7. Data is integrated.
8. Concurrent access to a file is not possible.	8. Concurrent access and crash recovery possible.
9. Security problems	9. Better security.
10. Little preliminary design.	10. Vast preliminary design.
11. Examples: C++, COBOL, VB.	11. Examples: Oracle, Postgres.
12. File system data sharing is not powerful as compared to DBMS.	12. DBMS offers features of data sharing efficiently.
13. Transaction concept is not used. (Set of related operations.)	13. The concept of transaction is an important aspect of DBMS.

2.6 ADVANTAGES AND DISADVANTAGES OF DBMS (S- 18, 17; W- 17)

Using DBMS, we have following advantages:

1. Cost of software development is reduced.
2. User can get proper logical organization of dataset.
3. Centralization for multi-user is available.
4. Centralized data reduces management problems and duplication of data also.
5. Data is independent.
6. User can monitor database performance.
7. Data redundancy and consistency are controllable.
8. Program and data interdependency is decreased.
9. Data flexibility is increased.
10. Data integrity and its quality is maintained.
11. Inconsistency of data is avoided.
12. Same data can be shared by many application programs.
13. Security to data is provided.

- DBMS has some disadvantages:
  1. Cost of software and hardware is more.
  2. Problems with backup and recovery of database.
  3. Problems with centralized data.
  4. Complexity increases in case of multiuser DBMS.
- Because of data sharing by various users, response given by the system slower down or degrade and time taken by the system is more.

2.7 USERS OF DBMS (S-17)

- The users of database system can be classified on the basis of degree of expertise or the mode of their interactions with DBMS.
- There are 4 groups of users of DBMS:
  1. Database designers.
  2. Applications programmers.
  3. Sophisticated users.
  4. End users.

2.7.1 Database Designers

- There are two main database designers.
  (i) Database manager.
  (ii) Database administrator.

2.7.1.1 Database Manager

- A database manager is supervising authority over the Database Administrator.
- A database manager manages and updates a database. The database manager has some knowledge about the database, but need not be having the same expertise as the database administrator.
- Actually the database administrator takes care of the database. The database administrator can create, delete and update a database.

2.7.1.2 Database Administrator (DBA) (W-17)

- The DBA is an individual person or group of persons (a team) who must have a good knowledge in data processing. DBA must have sufficient technical knowledge to make decisions in the organization.
- The DBA must manage the staff to ensure orderly development of the database project, to satisfy database users and to plan for future database requirements.

Responsibilities of DBA:

1. Deciding information contents of the database i.e. decide conceptual and internal schema. The data dictionary is created by DBA, so users can query the dictionary.
2. DBA decides the storage structures and access strategy i.e. which user can retrieve what.
3. Defines authorization checks and validation procedures. DBA checks validity before giving access.
4. Define strategies for backups and recovery.
5. Define security and integrity rules.
6. DBA monitors the system performance.

2.7.2 Application Programmers

- They are computer professionals.
- They are responsible for the development of application programs.
- For the development of application program, they use a general purpose programming language like C, COBOL and Pascal.
- By using these programs, these people handle or manipulate the databases.

2.7.3 Sophisticated Users

- These people are also computer professional but they do not write programs. To interact with the system, they use query languages like SQL.

2.7.4 Specialized Users

- They are sophisticated users. They write specialized database application programs.
- For example: Computer-aided design system, knowledge base and expert systems, etc.

2.7.5 End Users

- End users are unaware of the presence of database system or any other system.
- These users are called as *unsophisticated users* who interact with the system through already written permanent programs.
- They are also called as *Naive users*.

2.8 VIEWS OF DATA (W-17)

- A database management system is a collection of interrelated files and a set of programs that allow users to access and modify these files.
- A main purpose of a database management system is to provide users with an abstract view of the data.

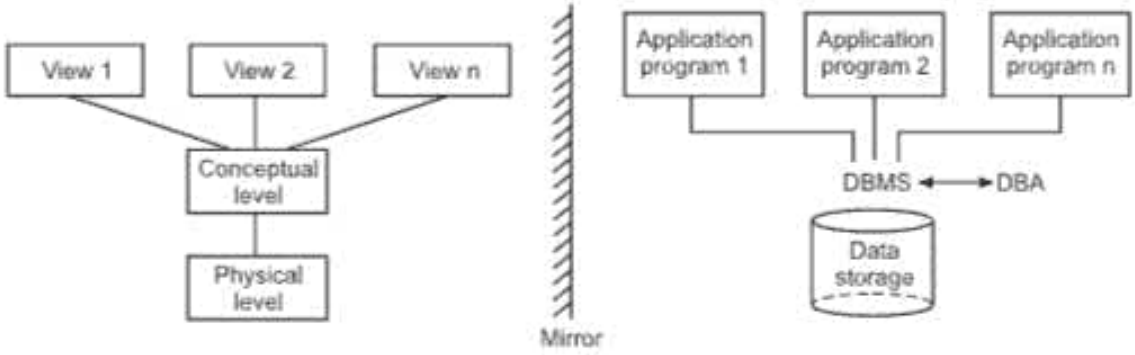


Fig. 2.3: Architecture of DBMS

[Note: (b) is Mirror Image of (a)]

- Data abstraction means hiding the implementation details from the end users. DBMS uses the same principle.
- It hides the complex details of how the data is actually stored on the secondary storage device in DBMS.
- There are three levels of data abstraction, Refer Fig. 2.3.

2.8.1 Physical Level

- Physical level is the lower level of Architecture of DBMS.
- This level describes how the data is actually stored in a database. This level defines the data structure used to store the data on secondary storage device.

2.8.2 Conceptual Level

- The conceptual level describes the structure of the whole database.
- The structure of each table is specified in terms of its attributes. Attribute type is specified in terms of logical structure of the database (Fig. 2.4). This level is decided by Database Administrator (DBA).
- This level also describes the relationships between various tables.

- For example

```
Struct Emp          →  defines data structure for a record in a table.
{
    int emp_id;
    char name [20];   } Shows type of attributes stored.
    float salary;
};
```

Fig. 2.4: Logical definition of database in 'C' Language

2.8.3 View Level

- In multi-user system, many users do not use entire database, but read some fields of it.
- For security purpose DBMS allows to create different views of single table.
- Because of this facility users will have separate logical structure (Fig. 2.5), so original database remains as it is. This access is provided by DBA only.

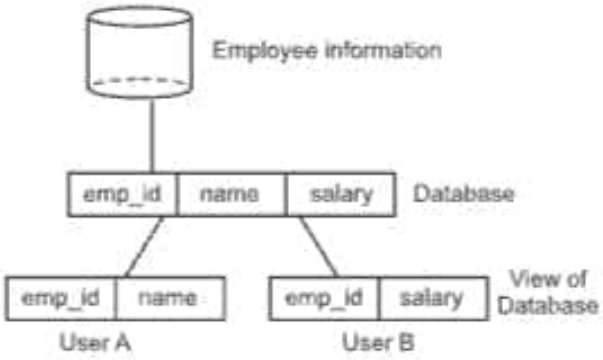


Fig. 2.5: View Level in detail

- At the physical level, employee record can be described as a block of consecutive storage location (For example: word or bytes).
- At the conceptual level, each record is described by a type definition and in view level, different views of the database are defined.
- For example, ATM machines in a bank see only that part of the database that has information on customer accounts.
- They cannot access information concerning salaries of employees.

Data Independence:

- Data independence means is the ability to change the overall logical or physical structure of the data without changing the application programs or views of data i.e. the views created for users does not have any change.
- There are two levels of data independence, physical and logical data independence.

(i) Physical Data Independence

- It is the ability to change the internal schema without having to change the conceptual or external schemas.
- For improving performance, we need to reorganize some physical files. This requires to implement changes to the internal schema.
- In this, the conceptual schema separates the users from the changes in the physical storage of the data.
- Physical data independence is easier to achieve as compared to logical data independence.

(ii) Logical Data Independence

- It is the ability to change the conceptual schema without having to change the application programs or external schemas.
- In this type of data independence, the users are protected from changes in the logical structure of the data.
- Only the view definition and the mapping need be changed in a DBMS that supports logical data independence.

2.9 DATA MODELS (W-17)

- DBMS allows us to store and retrieve data.
- Data consists of facts, which became information when they are processed. This information is conveyed to people.
- For example: if order and payments are the data, then balance due and the quantity in hand would be the information.
- While designing a database, developer has to make decisions about the data which is stored in a database.
- The structure and relationship of that data has to be decided.
- So developer uses some methods to create a database for the user and system implementation. These methods are called *Models*.
- A model defines the method of storing and retrieving data.
- A model is an abstraction process that hides the details which are not required while highlighting details required to the applications at hand.
- A data model is a collection of conceptual tool for describing data, its relationship, data semantics and data constraints.
- **A Data Model consists of:**
  1. A named logical unit i.e. record type and data item.
  2. Relationship among these logical units.
  3. Data item is logical unit of data and record type is collection of data items.
- Actually, it provides abstraction of database application.

- There are three types of data models available, [Refer Fig. 2.6].
  1. Object Based Logical Models.
  2. Record Based Logical Models.
  3. Physical Data Models.

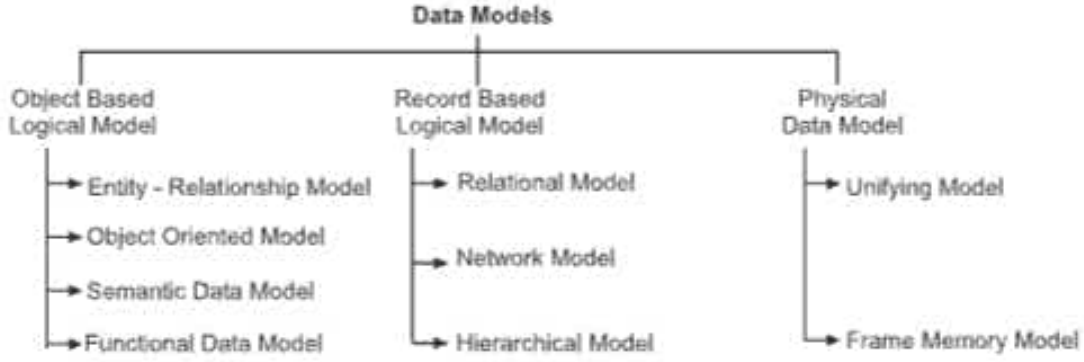


Fig. 2.6: Data Models and Types

2.9.1 Object Based Logical Models

- These models are used to describe entities in the real world, attributes of each entity and their relationship. It is used to describe logical data structure. There are four subtypes.
  - **(a) Entity Relationship Model:** It is high level data model based on the fact of real world. This model consists of collection of basic objects called as entities and the relationship among these objects.
  - **(b) Object Oriented Model:** This model is based on collection of objects. Each object has its own methods. Actually the value stored for a object and methods are grouped into a logical unit called as **class**. A user is allowed to define various objects of same class. This model actually talks about classes and objects. For example: C++. Unlike the traditional databases, this model does not have any inbuilt database structure.
  - **(c) Semantic Data Model:** The semantic data model is a method of structuring data in order to represent it in a specific logical way. Generally individual data or a word does not express any meaning, but if it is used with a context derives more meaning.
  - **(d) Functional Data Model:** The functional data model defines data objects, attributes and relationships as database functions. The central concept of this model is a definition of all components in the form of functions. A functional data manipulation language has a number of data manipulation functions that can be applied to database functions.

2.9.2 Record Based Logical Models (W-16)

- These models are also used to describe data at logical and view level.
- These are used to describe the logical database structure in terms of records.
- These model show the implementation part.
- These models have three types in it.
  1. Relational Model.
  2. Network Model.
  3. Hierarchical Model.

1. Relational Model:

- This model shows collection of tables to represent both data and the relationship.
- Each table consists of multiple columns and each column is recognized by a unique name.
- For example: In a database, Customer table and Account table are the two tables available.
- The relationship between these two tables has been shown by a third table, where one field or column name from both the table is taken together.
- So a new table called Customer_Amt is formed.

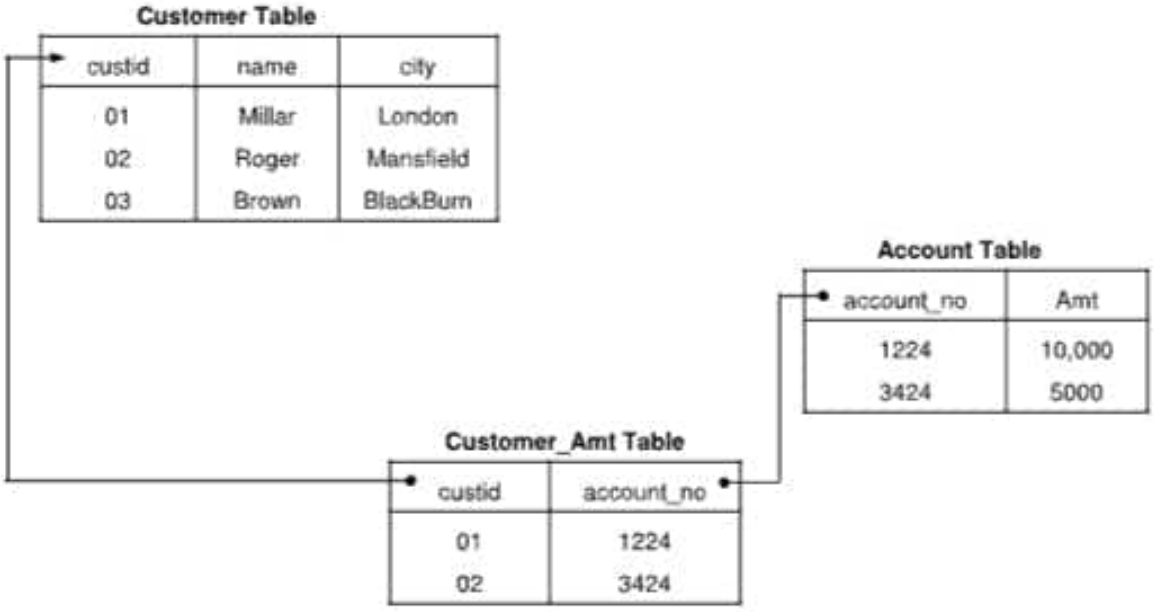


Fig. 2.7: Relational Model Representation

2. Network Model:

- Data is represented by collection of records and the relationship among the data is represented by links.
- The records are organized as a collection of arbitrary graphs.

- For example, in customer table extra field or column is considered, which consists of the address where account table information is stored.
- Information of each link represents the related record address.

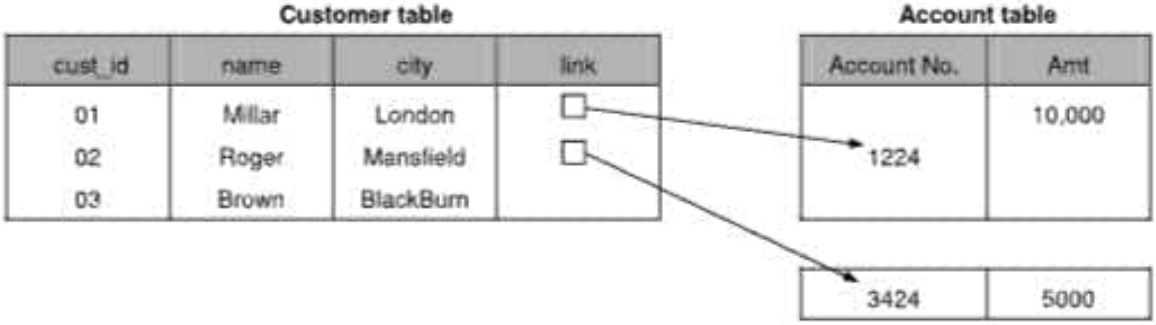


Fig. 2.8: Network Model Representation

3. Hierarchical Model:

- Data is similar to network model i.e. links or pointers used to show the relationships.
- Only change is that the records are organized as a collection of tree.

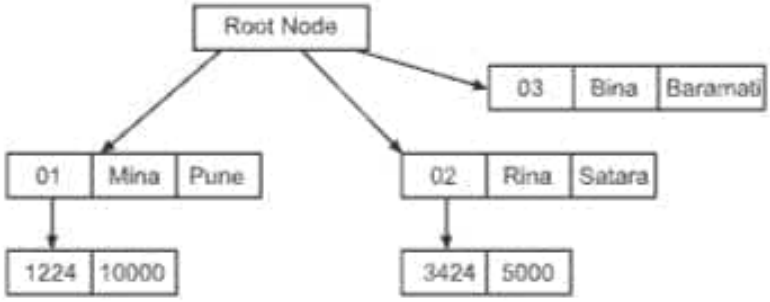


Fig. 2.9: Hierarchical Model

2.9.3 Physical Data Models

- This model is used to describe data at the lowest level.
- It describes about the way of data stored in secondary storage device by representing information such as record formats, record ordering, access path etc.
- It has two types of data models.
  1. Unifying model.
  2. Frame memory model.

2.10 ENTITY RELATIONSHIP DIAGRAM

- An entity-relationship diagram is a photocopy of data structure.
- E-R diagram was developed in 1976 by Dr. Peter Chen and others.
- They simplified the representation of large and complex data storage concepts using the E-R diagrams.

- An E-R model is based on a perception of a real world which consists of a collection of basic objects called *entities* and *relationship* among these objects.
- It represents the overall logical structure of a database. An E-R model is normally expressed as an E-R diagram which is a graphical representation of a particular database.

  Elements of the E-R model:

  1. Entities.
  2. Attributes.
  3. Entity sets.
  4. Relationships.
  5. Relationship sets.

2.10.1 Entities

- An entity is a real world object.
- This object has its own properties.
- This object is distinguishable from other objects.
- For example, student, employee, doctor, politician are the entities.
- Student has its own properties like his/her own name, address etc.
- An entity is represented by a rectangle in E-R diagram.
- For example,

- An entity has set of properties called *attributes* which holds value in it.
- An entity must have some attribute through which it is uniquely identified.
- For example: Employee code is the identity of employee.
- Roll number is the identity of a student.

2.10.2 Attributes

- Once, the entity is identified, then it is described in real terms or through its attributes.
- An *attribute* is description that is used to express the property possessed by each member of an entity set or a relationship.
- For each attribute, set of permitted values are available called as *domain*.
- The attributes which uniquely define the occurrence of an entity are called *primary keys*.
- Roll_no and Employee_ id is a primary key of above entities Student and Employee.

- For example, **Disease table**

Disease name	Causing organism
Typhoid	Salmonella Typhi
Dysentry	Shigella dysentriae

- No primary key is available. Then we can consider *disease_no* as extra field or attribute.
- An attribute must appear in only one place in the model. Either it may be required or optional.
- When the attribute is required, it holds a value. When it is optional, it may or may not have a value for it (null value).
- For example, Entity plant. It's attributes are plant name, plant type, date of acquisition and pot size.
- Pot_size is optional because some plants do not come in pots. They come in plastic bags. But other fields are required.
- In E-R diagram, an attribute is represented in two different ways.
  1. By ellipses attached to entities (Fig. 2.10).
  2. By listing the attribute as text. (Fig. 2.11).

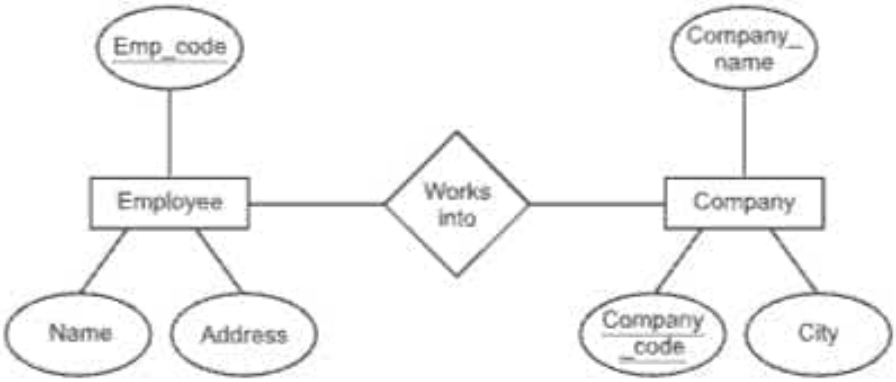


Fig. 2.10: Representation of attributes by Ellipses

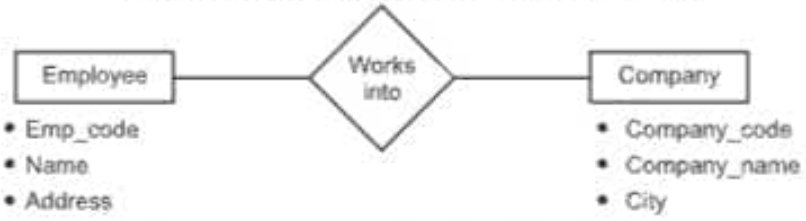


Fig. 2.11: Representation of attribute by text

- An attribute may be of following types:
  1. **Simple Attribute:** It represents a single property i.e. these are not divided into meaningful subparts. These are also called as atomic attributes. For example:

2. **Composite Attribute:** The attribute can be divided into meaningful subparts. This helps us to group together the related attributes. This may appear as a hierarchy.

   For example, Address is the composite attribute having street address, city, state, zip as simple attributes. This you can see in Fig. 2.12.

Fig. 2.12: Composite Attribute

3. **Single valued Attribute:** The attribute which contains one value for a specific entity is called single valued attribute. For example: Roll_no, Name, Amount, Age has only one value.
4. **Multi-valued Attribute:** The attribute which contains more than one value for a specific entity is called multi-value attribute.

   For example:

   1. Person has many phone numbers.
   2. Number of dependents on an employee.

   The multivalued attributes are represented by the following symbol.

Fig. 2.13: Symbol of Multi-valued Attribute

For example:

5. **Null Attribute:** The attribute without any value is called as **Null Attribute**. Null value is used when entity does not have a value for an attribute. There are two assumptions done for this null attribute.

   Sometimes when there is no value present we say "**not applicable**" or sometimes we may say "**value is missing**". Let's see the example.

   Suppose employee have no dependents and the name of dependent is null it means, this is not applicable.

   If say employee-code or social security number is null it means it is missing.

The attribute *phone_no* of employee can have value NULL, if employee does not have telephone connection.

6. **Derived Attribute:** The value of this attribute can be derived from values of other stored attributes of entities. Finding marks or grade or percentage of a student is the best example of derived attribute. Given an attribute *birth_date*, the *age* can be derived from this attribute for an employee. So age is derived attribute and birth_date is called stored attribute. It is represented as dotted ellipse or oval as shown in Fig. 2.14.

Fig. 2.14: Derived attribute

For example:

- The concept of an entity set corresponds to the programming language notion of type definition.
- A variable of a given data type has a particular value at a given instance of time. Thus, a variable in programming language corresponds to the concept of an entity in the E-R model.
- A database thus includes a collection of entity sets each of which contains any number of entities of the same type.
- For example, bank database that consists of five entity sets:
  1. **Branch**: the set of all branches of a bank. Each branch is described by attributes branch_name, branch_city and assets.
  2. **Customer**: the set of all people who have account at bank. Its attributes are customer_name, social_security, street and customer_city.
  3. **Employee**: the set of people who work in bank. Each employee has attributes employee_name and phone_number.
  4. **Account**: the set of all accounts maintained in the bank and attributes are account_number and balance.
  5. **Transaction**: the set of all account transactions executed in the bank. Attributes are trans_no, account_number, date, trans_type, amount.

2.10.3 Entity Sets

- Entity is the basic object of E-R model.
- An **entity** is an **object** in the real world which has its own properties and it is distinguishable from other objects.
- An **entity set** is a set of entities of the same type that share the same properties or attributes.

- For example,
  1. The set of all persons who are employees at the college and defined as an entity set employee.
  2. The set of all persons who are students at the college and defined as an entity set student.
  3. If a bank has number of branches then branch is also a entity set. So in each branch entity-set may be described by the attributes branch_name, branch_city and assets.
- For example, employee of a college [employee]
  student of a college [student]
- A person entity may be an employee entity or student entity or both or sometimes no one from this set.
- An entity may be physical like a flower or a book or abstract like a concept or holiday.

Strong and Weak Entities:

- Sometimes, there is no attribute in an entity which shows uniqueness of entity.
- An entity set may not have sufficient attributes to form a primary key is termed as Weak Entity Set.
- The entity set which has a primary key is known as Strong Entity Set.
- A weak entity set is indicated by double lined rectangle.

- **For example:** In a bank customer has account. Therefore, *customer* and *account* are related with each other.
- Customer does some transactions in a bank on the account.
- In transaction entity if transaction_no is considered as primary key, for one customer deposit and withdrawal is done and key value starts from one because it is typically in a sequential number [Fig. 2.15].

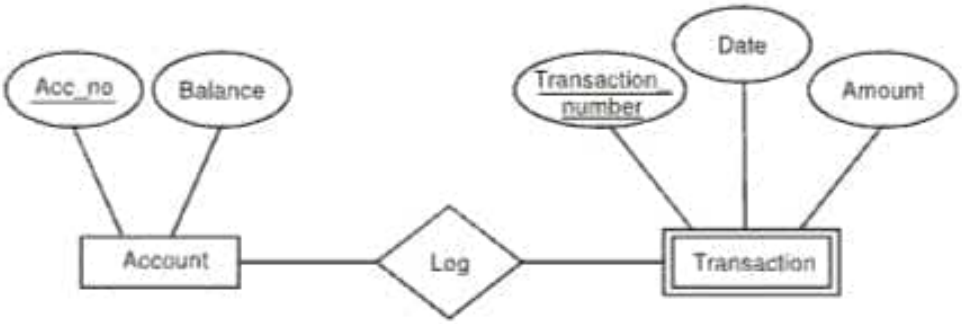


Fig. 2.15: Account keeps Log of Transaction

- For each customer the *transaction_no* starts at 1. See following table, [Fig. 2.16].

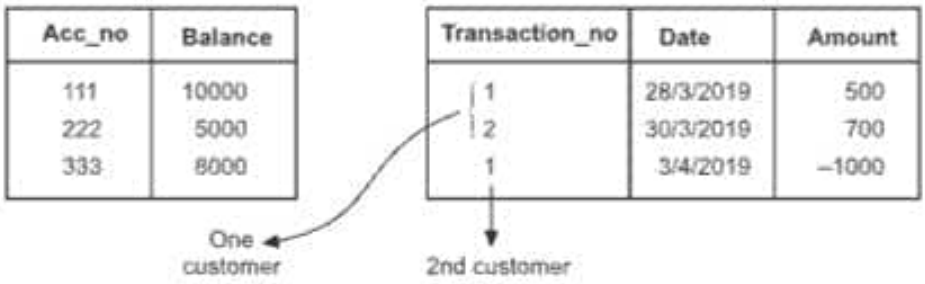


Fig. 2.16: Table consists of records

- Eventhough transactions are distinct, same transaction number is used for deposit and withdrawal for different customers.
- Thus transaction_no is not a unique value. So it is referred to as weak-entity.
- Therefore the relationship is also weak. This is shown in the Fig. 2.17.

Fig. 2.17: Weak Relationship

- To make a weak-entity set meaningful it must be associated with another entity set called as owner entity set. If we consider account_no with transaction_no, it will show a healthy relationship between account and transaction, [Fig. 2.18].

Acc_no	Transaction_no	Date	Amount
111	1	28/3/2019	+500
222	1	3/4/2019	–1000

Fig. 2.18: Strong relationship through Weak Entity

2.10.4 Relations and Types of Relationship (W-17)

- Relationship shows the association between different entities.
- There are four types of relationships:
  1. **One-to-One (1 : 1):** An entity in A is associated with at the most one entity in B and an entity in B is also associated with at the most one entity in A. [Refer Fig. 2.19].

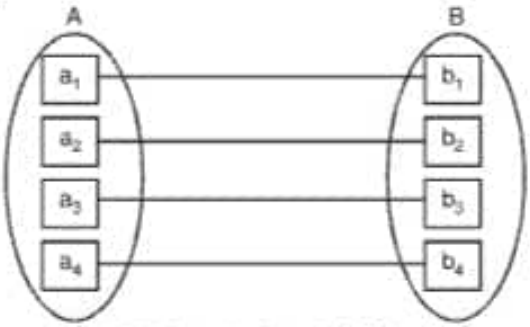


Fig. 2.19: One-to-One Relationship

For example,

- One Class has one Roster [Refer Fig. 2.20].
- One Patient is assigned to one bed.

Fig. 2.20: Representation of One-to-One Relationship

2. **One to Many (1 : M):** An entity in A is associated with any number of entities in B and entity in B is associated with at the most one entity in A. The 1: M relationship is represented using 'Parent-Child' concept. Here 'One' side of relationship is called parent and 'Many' side of relationship is called as child. See Fig. 2.21.

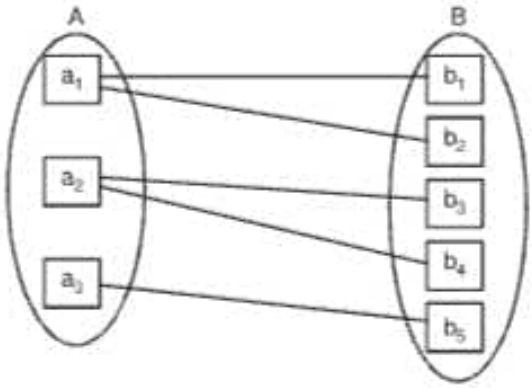


Fig. 2.21: One-to-Many Relationship

For example: One student takes many subjects [see Fig. 2.22 (a)]. Similarly one manager manages many employees [See Fig. 2.22 (b)].

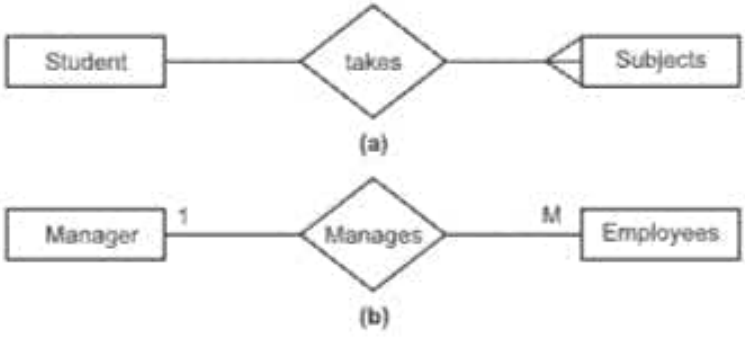


Fig. 2.22: Representation of One to Many Relationship

3. **Many to One (M : 1):** An entity in A is associated with at most one entity in B and an entity in B is associated with any number of entities in A. (Fig. 2.23).

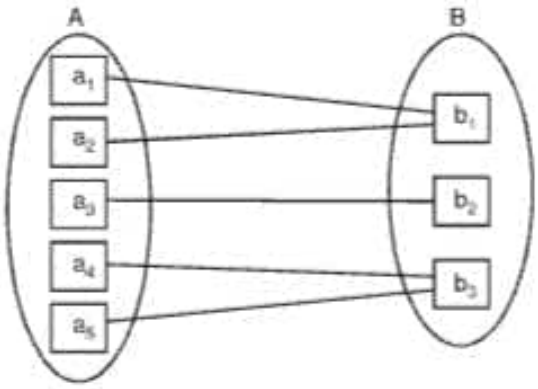


Fig. 2.23: Many-to-One Relationship

For example: Many politicians are present in a party [See Fig. 2.24 (a)]. Many employees are working in a department [Refer Fig. 2.24 (b)]. It means one department can have many employees and one employee is working in one department.

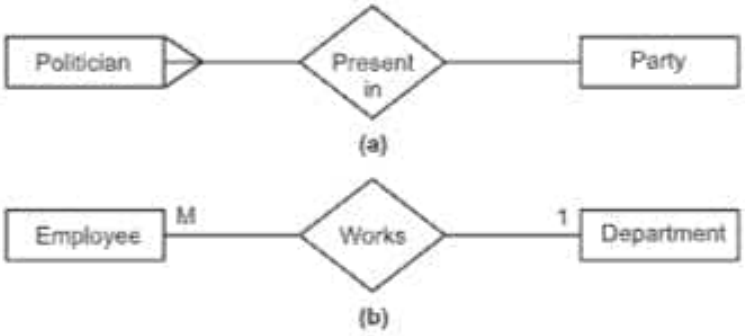


Fig. 2.24: Many to One Relationship

4. **Many-to-many (M : M)**: Entities in A can be associated with any number of entities in B and vice versa. [Fig. 2.25].

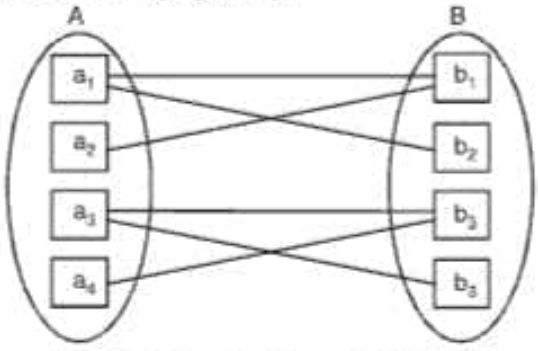


Fig. 2.25: Many-to-Many Relationship

For example: Many teachers conduct many tests [see Fig. 2.26 (a)]. Similarly many doctors work in many Hospitals [see Fig. 2.26 (b)]. Means one doctor work

in many hospitals and one hospital has many doctors. Therefore, relationship is Many-to-Many.

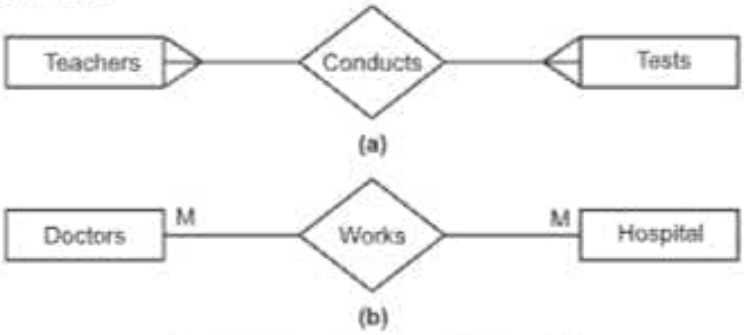


Fig. 2.26: Many to Many Relationship

- All above are the types of relationship depending upon cardinality.
  There are three more types of relationship available.
  (a) Binary relationship
  (b) Unary relationship
  (c) Ternary relationship
- Let us learn something about these types.
  1. **Binary Relationship:** It has involvement of two entities only. Therefore the degree of relationship is 2. One entity may be related with 1 : 1, 1 : M, M : M cardinalities with another entity.

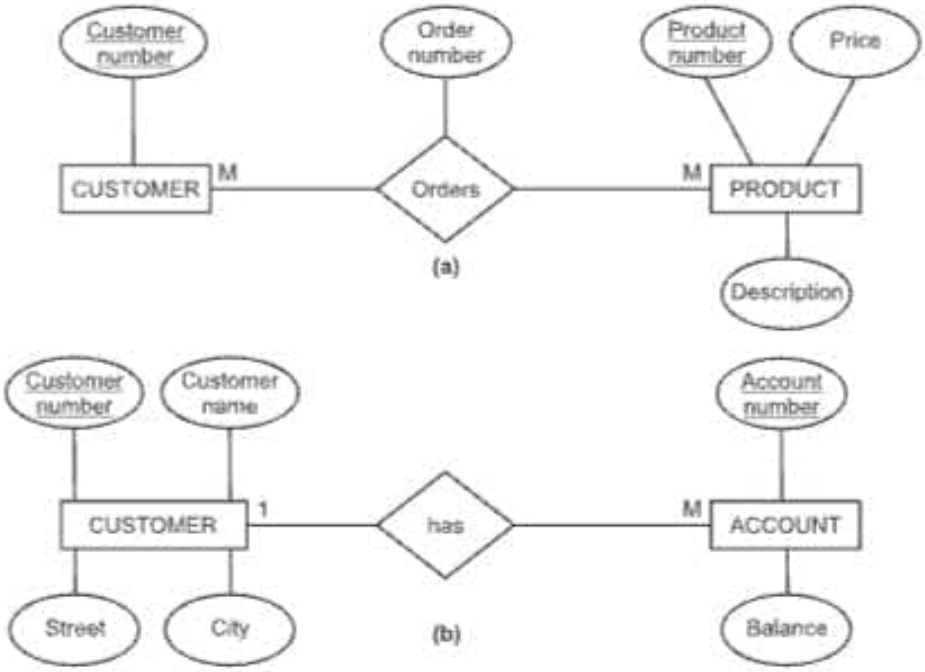


Fig. 2.27: Binary Relationship

For example: Many products can be ordered by many customers. Here Product and Customer are two entities. [See Fig. 2.27 (a)]. Similarly, one customer can have many accounts so again two entities involved [Fig. 2.27 (b)] and therefore referred to as binary relationship.

In E-R diagram primary key is represented by the symbol shown in Fig. 2.28.

Fig. 2.28: Primary Key Representation

2. **Unary Relationship:** This represents the association between the occurrence of *single* entity. Therefore the degree is 1. [Fig. 2.29].

   For example: 1. Unary person 1: 1 person (marriage).

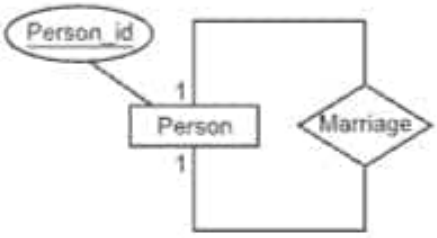


Fig. 2.29 (a): Unary Relationship (1 : 1)

2. Unary item M : M item (Bill of material)

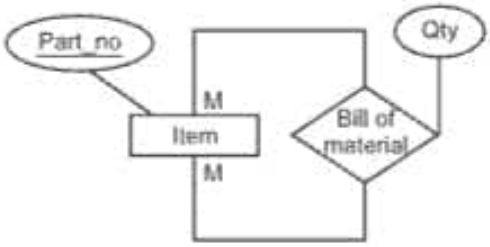


Fig. 2.29 (b): Unary relationship (M : M)

3. **Ternary Relationship:** It refers to the occurrence of three entities at the same time and degree is three.

- For example:
  1. An employee works into a Branch and his job is say manager. If we write it as binary relationship, then employee_branch and employee_job will create some problems for validity. [See Fig. 2.30 (a)].
  2. Ternary relationship is available with Item, Vendor and Warehouse entities (M : M : M).

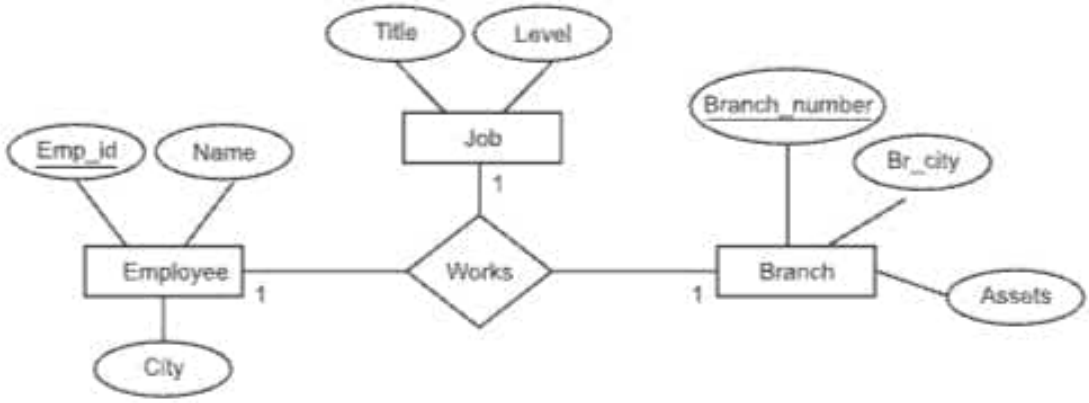


Fig. 2.30 (a): Ternary Relationship

3. Ternary relationship is available with Customer, Branch and Account entities as shown in the Fig. 2.30 (b).

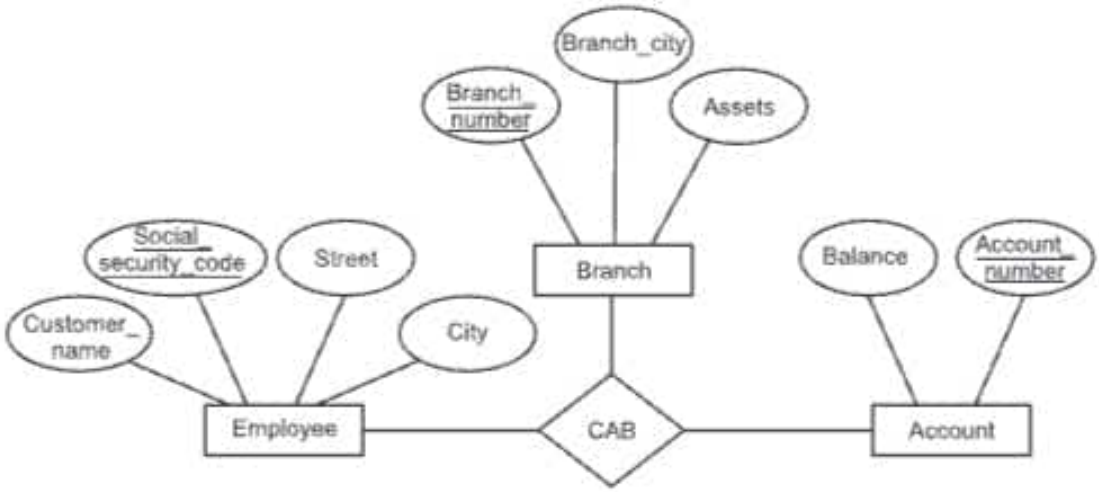


Fig. 2.30 (b): Ternary Relationship

2.10.5 Relationship Sets

- A *relationship* is association or linkage among several entities.
- For example: The relationship between Employee and Department is shown in the Fig. 2.31 (a).
- The relationship between Customer and Sales order is shown in the Fig. 2.31 (b).
- Similarly the relationship between Customer and Account is shown in the Fig. 2.31 (c).
- The relationships are represented by diamond shapes.
- A *Relationship Set* is a set of relationships of the same type.
- Formally speaking, it is a mathematical relation on $n \geq 2$ possibly non-distinct sets.

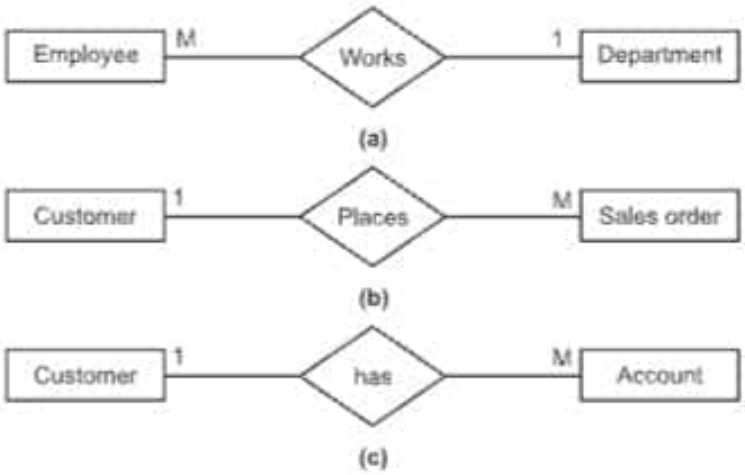


Fig. 2.31: Representation of Relationship

- If $E_1, E_2, E_3 ..., E_n$ are entity sets, then a relationship set S is a subset of
  $\{(e_1, e_2, ..., e_n) \mid e_1 \in E_1, e_2 \in E2, ..., e_n \in E_n\}$
  where $(e_1, e_2, ..., e_n)$ is a relationship.
- For example: Consider the two entity sets doctor and patients. [See Fig. 2.32]

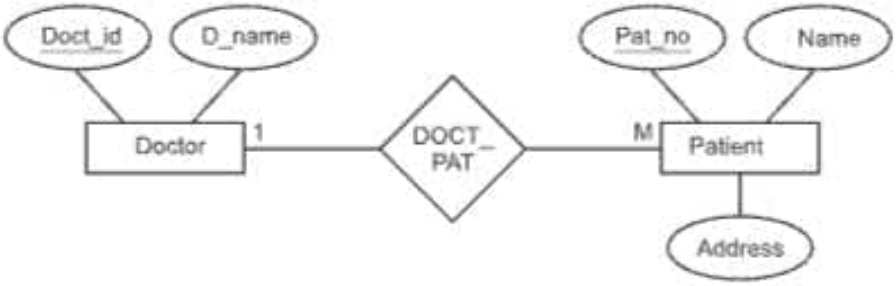


Fig. 2.32: Entity Doctor and Patient Relationship

- We can define the relationship set DOCT_PAT to denote the association between above named entities.
- This represents real-world entities.

Doct_Id	D_Name
01	Smith
02	Johanson
03	Williams

Pat_no	Name	Address
101	Anderson	London
102	Thomas	Newyork
103	Jackson	Sydney

Fig. 2.33: Relationship DOCT_PAT

- From Fig. 2.33, we can say DOCTOR Smith handles 2 patients i.e. Anderson and Thomas.

2.11 EXTENDED FEATURES OF ERD

- The basic E-R concept can model most of the database features.
- But same aspect of database can be expressed by adding some extra features called *extended features*.
- There are 3 extended features:
  1. Specialization
  2. Generalization
  3. Aggregation

1. Specialization (↓):

- It is the abstracting process of introducing new characteristics of objects to create one or more new classes of objects.
- In simple words, specialization is a result of taking a subset of higher level entity set to form a lower level entity set, [Fig. 2.34].
- Entities are expanded in Specialization.
- This line implies that Doctor can also be anything else apart from permanent doctor and Consulting doctor.

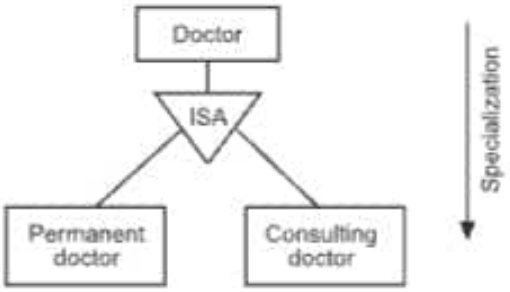


Fig. 2.34: Specialization Process

2. Generalization (↑): (S-18)

- It is abstracting process of viewing set of objects as a single general class. It concentrates on the general characteristics of consequent set while ignoring their differences.
- It means generalization is the result of taking the union of two or more entity sets to produce a higher level entity set, [Fig. 2.35].
- This line implies that a doctor has to be either a permanent doctor or a consulting doctor, nothing else. Therefore the entities are limited in Generalization.

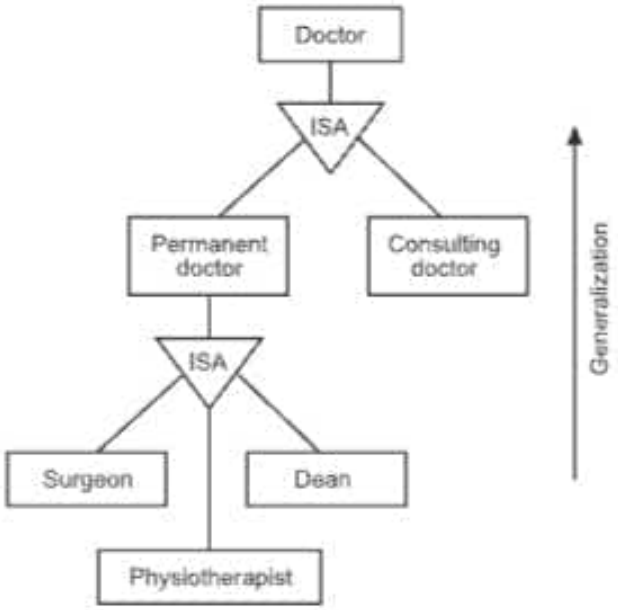


Fig. 2.35: Generalization Process

The difference between Specialization and Generalization: (W-16, 17)

Generalization	Specialization
1. It is bottom-top design process.	1. It is top-down design process.
2. It is union of two or more lower level entity sets to produce higher level entity set.	2. It is subset of higher level entity set to form lower level entity set.
3. In this, lower level entity sets are also described by attributes and relationship of higher level entity sets.	3. Lower level entity sets may have different attributes and may participate in relationship that do not apply higher level entity set.
4. It is used to emphasize similarities among lower level entity types.	4. It is used to emphasize distinction between higher level and lower level entity set.
5. In this, every higher level entity must also be a lower level entity.	5. It allows for the possibility of more low level entity sets also.

3. Aggregation:

- It is the process of combining information on objects so that the higher level objects can be abstracted.
- In any entity relationship diagram, we cannot form a relationship among the entities and their relationship.
- Aggregation is the process which allows to do so. (See Fig. 2.36).

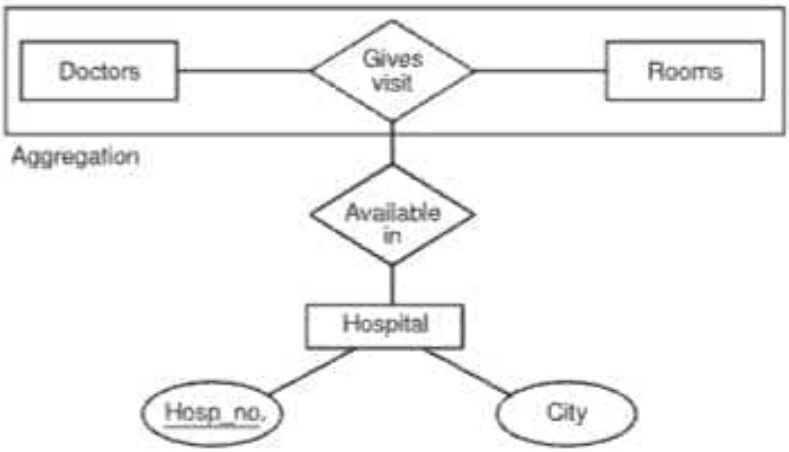


Fig. 2.36: Aggregation Process

- In above example, many doctors gives visits to many rooms and everything is done into the same hospital.

ERD

- An ERD is a pictorial representation of the entities and the relationship between them. From this, user can use the information structure of the application at a glance.
- Afterwards these ERDs are used to design tables and databases.
- E-R diagrams consist of the following major components:
  1. Rectangle – represents entity set.
  2. Ellipses – which represents the attributes.
  3. Diamonds – which represents relationship sets.
  4. Lines – used as connectors between entities and relationship sets.
  5. Double ellipses – which represents multi-valued attributes.
  6. Dashed ellipses – represent derived attribute.
  7. Primary key is represented by underlining the text in an ellipse.
  8. Double rectangle represents weak entity.

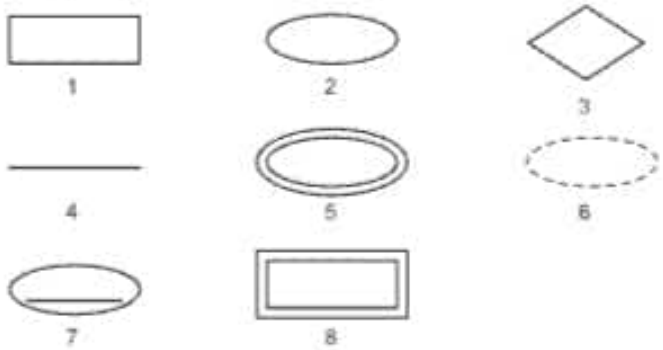


Fig. 2.37: Symbols used in ER diagrams

Case Study:

1. Let's consider "General Hospital" where the operations are as follows:

(a) In the hospital, many doctors are working. Their personal information is maintained because they get fixed salary per month.

(b) The patients are admitted to the hospital into the room. They are treated by various doctors.

(c) Sometimes, patients require certain pathological tests which are actually carried out into the labs.

- This information is shown in the E-R diagram which is shown in the Fig. 2.38 with primary keys.

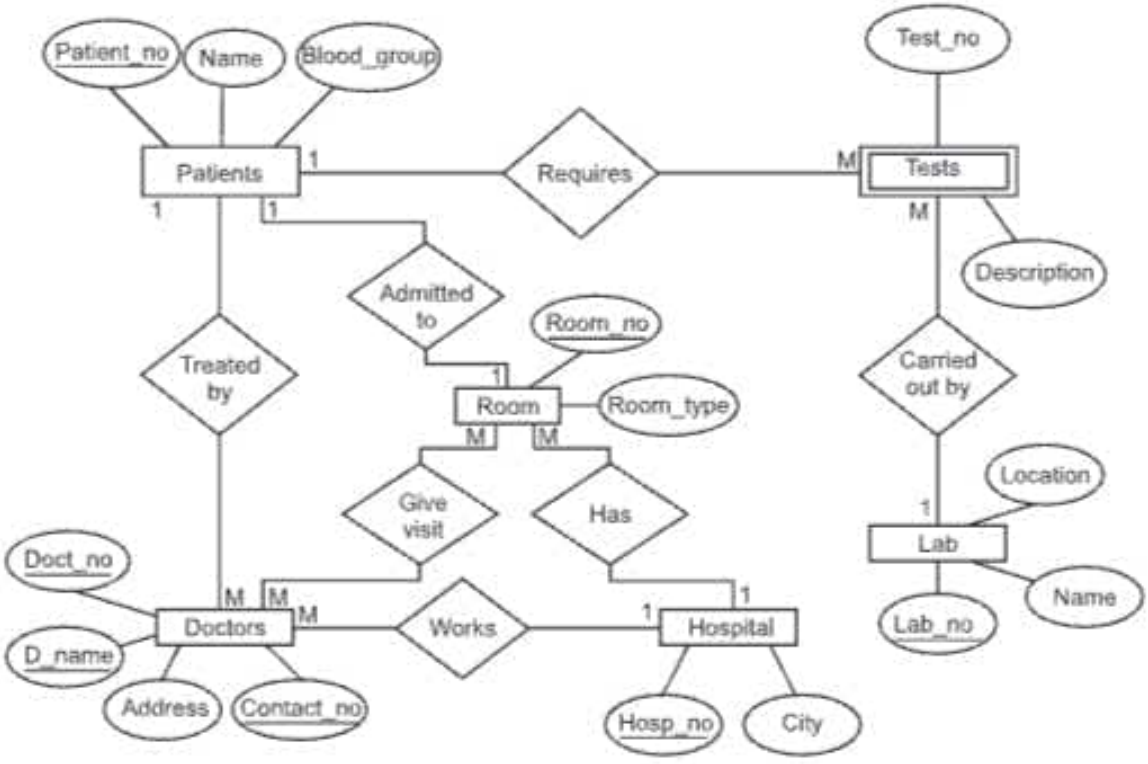


Fig. 2.38: ERD of Hospital system

2.12 OVERALL SYSTEM STRUCTURE

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system. In most cases, the computer's operating system provides only the most basic services and the database system must build on that base. Thus, the design of a database system must include consideration of the interface between the database system and the operation system.

- The basic components of DBMS can be divided into three subsystems.
  1. **Design tool:** It allows to create database form and reports.
  2. **Runtime facilities:** It process the application created by the design tool.
  3. **DBMS engine:** It works as a translator between design tool and runtime facilities. It is divided into two main parts. (Refer Fig. 2.39).
     (a) Query processor components.
     (b) Storage manager components.

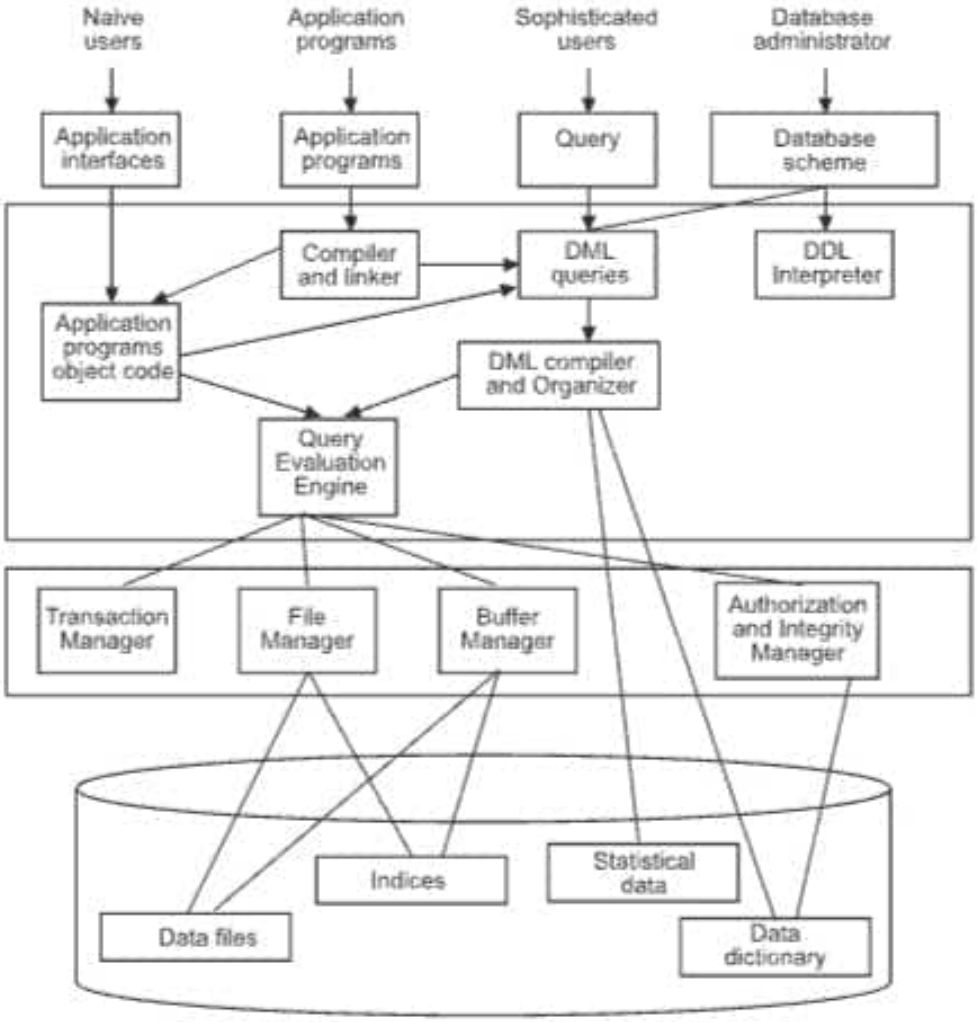


Fig. 2.39: Overall system structure or DBMS engine

- Let's us see these main parts of DBMS Engine in detail.

(a) Query processor components: It has following sub-components.

(i) DML compiler and Organizer: When query is given i.e. DML statement, it translates that DML statement to low level instruction which a query evaluation

engine understands. It also translates the users request into more efficient form so that query engine finds a good strategy to execute that query.

(ii) **Compiler and Linker:** It converts the DML statements embedded in application program to normal procedure call in host language. So that it interacts with DML compiler.

(iii) **DDL Interpreter:** It translates Data Definition Language statement into the set of tables in the form of metadata. The metadata is data about data and called data dictionary.

(iv) **Query Evaluation Engine:** It executes the low level instruction passed by DML compiler to fetch the necessary data.

(b) **Storage Management Components:** These components provide an interface between low level data stored in database and application program and the queries submitted to the system.

Following are the subcomponents:

(i) **Authorization and Integrity Manager:** When a query is submitted to a system, it is first checked for authorized data access i.e. person who executes query has a right or not. Then query is checked for integrity constraint.

(ii) **Transaction Manager:** In spite of system failure, this ensures about the consistency of database. It also maintains the atomicity.

(iii) **File Manager:** It manages the allocation of disk space and data structure used to represent information stored on disk.

(iv) **Buffer Manager:** It is responsible to fetch data from disk into main memory and also decides which data to cache in memory.

- Other than these two main parts, there are some data structures available which resides in DBMS.

(i) **Data files:** This stores the database.

(ii) **Data dictionary:** This stores the information about data in database called as **metadata**.

(iii) **Indices:** This provides fast access to the data items that hold particular files.

(iv) **Statistical data:** This stores statistical data about the data.

2.13 CASE STUDIES (S-15,16,18 W-15,16, 17

- Solved case studies are included to show how E-R model is used for conceptual design.
- To learn the effective way of drawing and designing E-R diagram.

Case Study 1: Construct an E-R diagram for a car insurance company that has a set of customers. Each customer owns one or more cars. Each car be associated with zero to any number of recorded accidents.

Solution: Refer Fig. 2.40.

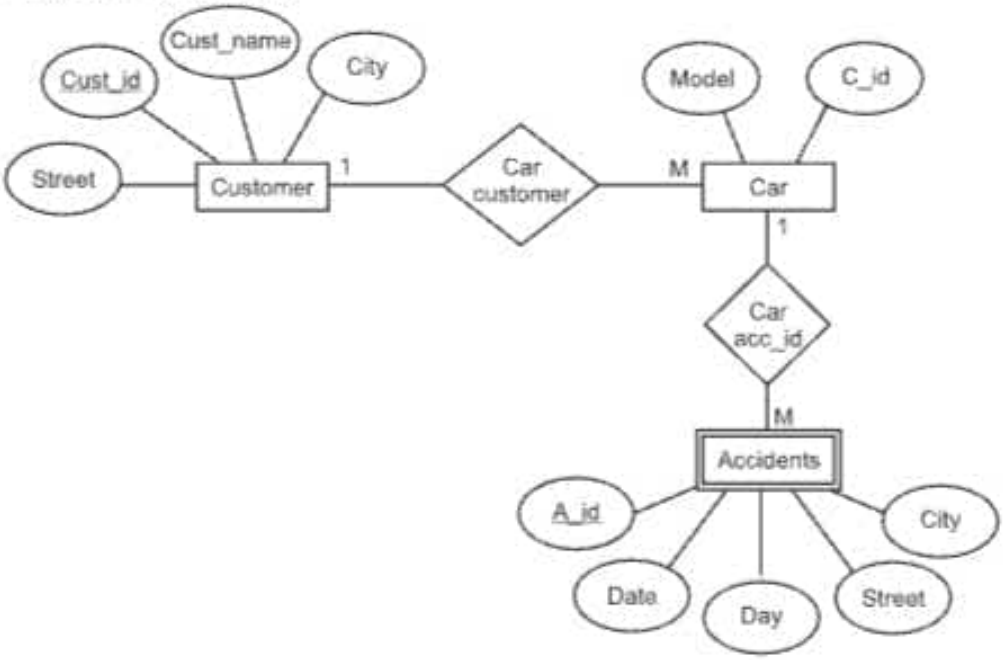


Fig. 2.40: Car Insurance System

Case Study 2: Draw an E-R diagram for order processing system where a person can give order for many items by specifying its quantity. And also an item can be ordered by many persons.

Solution: Refer Fig. 2.41.

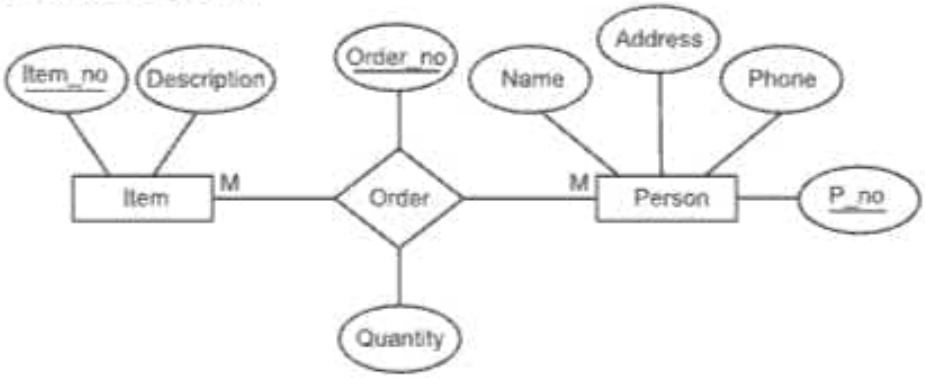


Fig. 2.41: Order Processing System

Case Study 3: A movie studio wishes to institute a database to manage their files of movies actors and directors. The following facts are relevant.

(i) Each actor has appeared in many movies.

(ii) Each director has directed many movies.

(iii) Each movie has one director and one or more actors.

(iv) Each actor and director may have several addresses.

Draw E-R diagram.

Solution: Refer Fig. 2.42.

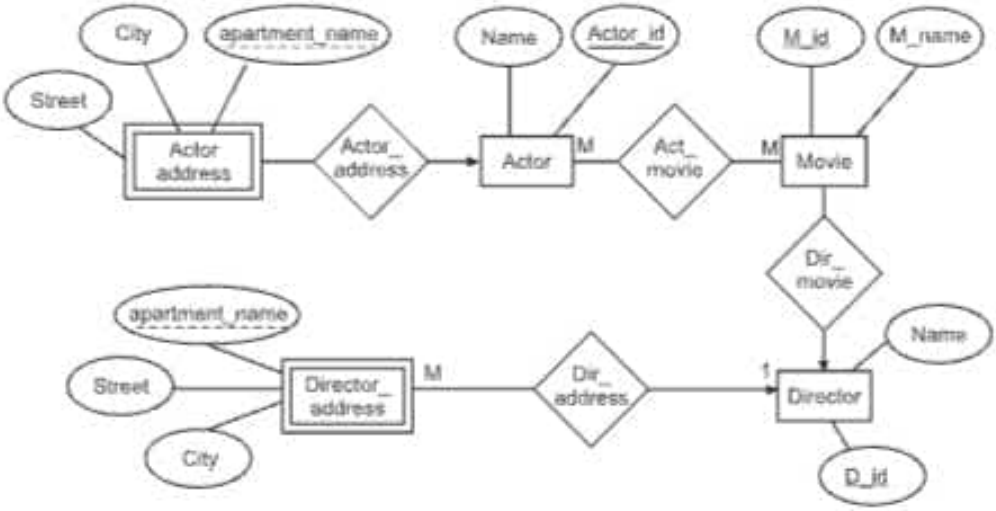


Fig. 2.42: Film Industry System

Case Study 4: In a nursery, the plants are sold to the customers. These plants are Flowering and Non-flowering only. Nutrients are given to the plant with some quantity. Nutrients includes Pesticides, Watering and Manure.

Solution: Refer Fig. 2.43.

Fig. 2.43: Plant Nursery System

Case Study 5: Study the following case study and

(1) Find out the entities.

(2) Draw E-R diagram.

The management of Life Hospital has decided to computerize their operations. The following information is provided by the management. There are resident, fulltime and consulting doctors, with various specialization. Consulting doctors visit hospital at a fixed time everyday or some days of week which varies from doctor to doctor. Patients are admitted to hospital and their main cause of admission is recorded. For accident cases, additional information such as police booklet number, name of police and accident description is recorded. A patient is admitted to a room which has certain category having fixed change per day.

The given case study is of Life Hospital which has decided to computerize their operations.

From the specifications listed above, we identify the entity sets and their attributes, also.

Thus,

Entity Sets Designation of 'Life Hospital'

(i) Doctor (Dr_no, Dr_name, Dr_add, Drspecialization)

(ii) Patient (P_id, P_name, P_gender, P_address, P_contact)

(iii) Visiting Schedule (Record_id, Day, from_time, to_time)

(iv) Category (C_id, C_name, charge_per_day)

(v) Room (R_no, R_description)

(vi) Admission details (admission_id, date, cause_adm)

(vii) Accident info (Accrecord_id, Police_booklet_no, police_name, Acci_descri).

Relationship Set Designation of "Life Hospital"

(i) Doctor has many visiting schedules.

∴ Doctor and visiting schedules are related by 1-M relationship.

(ii) Doctor looks after patients.

∴ Doctor and patients are related by M-M relationship.

(iii) Patient has admission details.

∴ Patient and admission details are related by 1-1 relationship.

(iv) Admission details has accident information.

∴ Admission detail and accident info are related with 1-1 relationship.

(v) Room uses admission details.

∴ Room and admission details are related by M-M relationship.

(vi) Room has category.

∴ Room and category are related by M-1 relationship.

Using the above entity sets and relationship sets, we draw the following **E-R diagram.** Primary/candidate keys are shown in the diagram with underline. (Section 2.44)

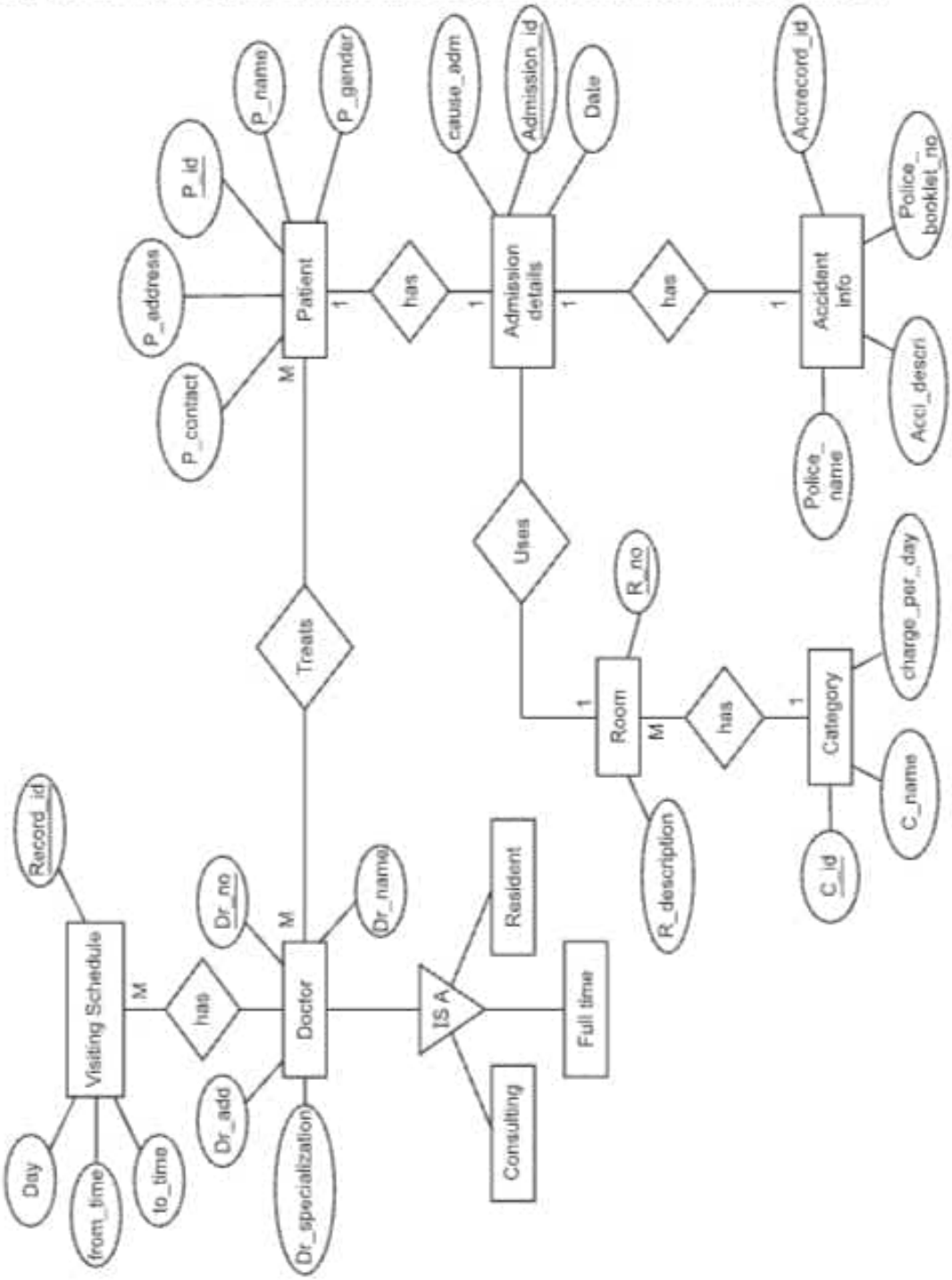


Fig. 2.44: Life Hospital System – ER diagram

Case study 6: "Star" is an agency for flat booking and it has number of builders and agents who are jointly working. A customer can get a flat for residential or commercial purpose. If customer is approached through an agent the agency and builders are giving some commission to the agent. Agent shows various flats and sites within various location.

Study the case and do the following: **(S-17)**

1. Draw an E-R diagram.
2. Design the relational database.

Ans. 1. Members are builders or agents.

2. Here agency and members are related to 1 to many relation.
3. Agency and customer relates with one to many.
4. Customer and flat related with 1 to 1.
5. Agent and commission relates with 1 to 1.

E-R diagram: (Fig. 2.45)

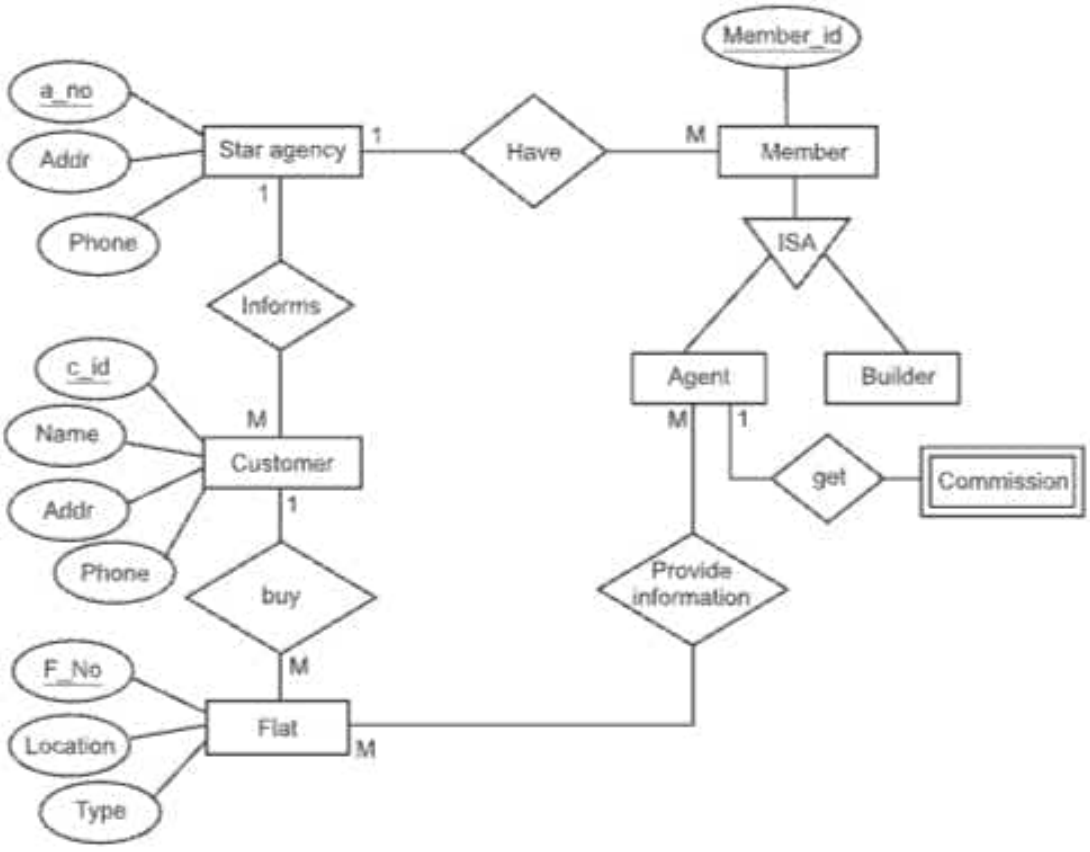


Fig. 2.45: Star Agency –Flat Booking System

Tables:

1. Agency (a_no, addr, phone) a_no is primary key.
2. Customer (c_id, name, addr, phone) c_id is primary key.
3. Flat (F_no, location, site_name, type, c_id) F_no is primary key, c_id is foreign key.
4. Agent (member_id, name, year_experience, a_no, c_id) (member_id, a_no, c_id) is candidate key.
5. Builder (member_id, a_no, name) (member_id, a_no) is candidate key.
6. Commission is a weak entity. Commission (member_id, comm_amount).

Case Study 7: Consider a trucking company which is responsible for picking up shipments for warehouses of a retail chain and deliver the shipments to the individual store location. A truck may carry several shipments in a single trip and deliver it to multiple stores.

Draw an ER-diagram and convert it into relational model.

Solution:

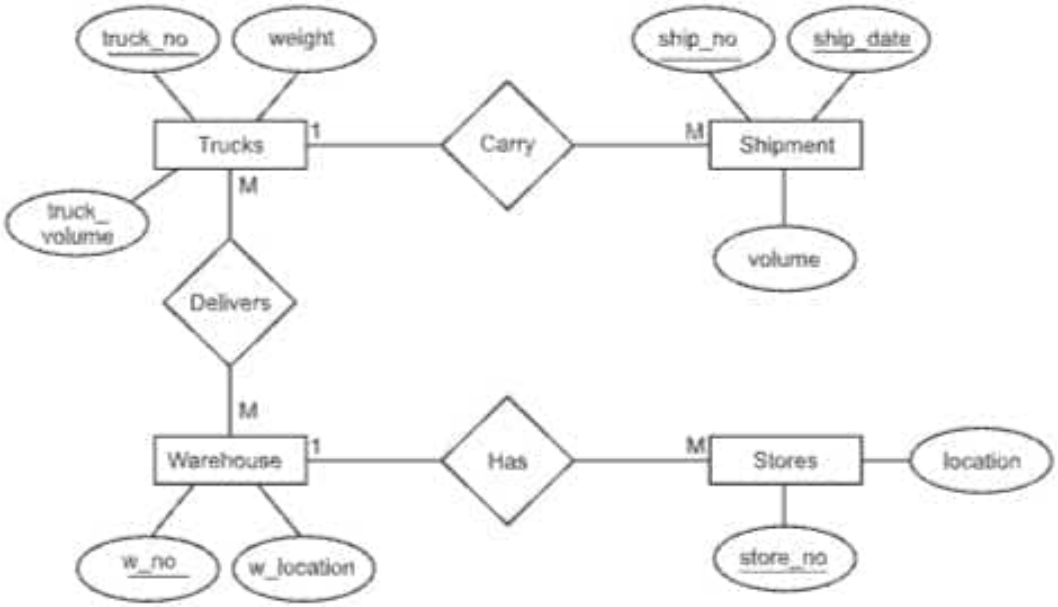


Fig. 2.46: Truck-Shipment System

The relational model consists of the following tables:

- Trucks (truck_no, truck_volume, weight).
- Shipment (ship_no, ship_date, volume, truck_no).
- Warehouse (w_no, w_location).
- Stores (store_no, location, w_no).
- Ware_trucks (truck_no, w_no).

Case Study 8: In a car sales management system, customer purchases a car. Many cars has similar model. Services are provided to each model. Each car gets a job card and

which contains spare types. These spare types has spare parts. Job cards were prepared by the advisor. Draw ER diagram and design the relational database.

Solution:

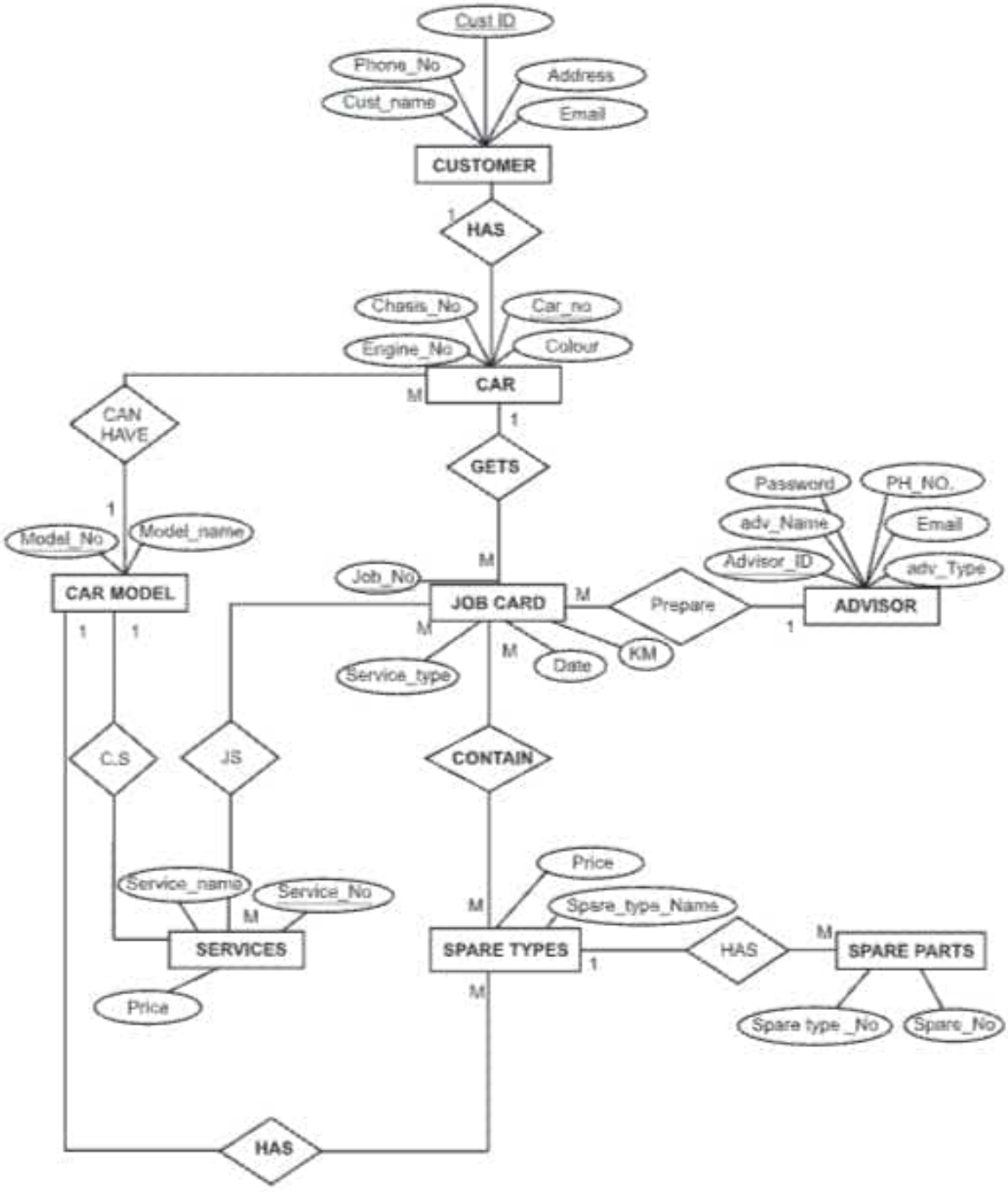


Fig. 2.47: Car sales Management System

The following are the tables available here:

1. Customer (cust_id, cust_name address, phone_no, email)
2. Car Model (model_no, model_name)
3. Car (car_no, chasis_no, engine_no, colour, cust_id, model_no)
4. Advisor (advior_id, adv_name, password, ph_no, emial, adv_type)
5. Job Card (job_no, servcie_type, date, km, car_no, advisor_id)
6. Services (service_no, service_name, price, model_no)
7. JC_Service (job_no, service_no)
8. Spare types (spare_type_no, spare_type name, price, model_no)
9. Spare parts (spare_no, spare_type_no)
10. Contain (job_no, spare_type_no)

Case Study 9: In airport, airlines reservation is done. Airplane are owned by airline which have flight system. Passenger visits employee and ask for reservation. Employee checks flight system and provide reservation to the passenger. Draw ER diagram.

Solution:

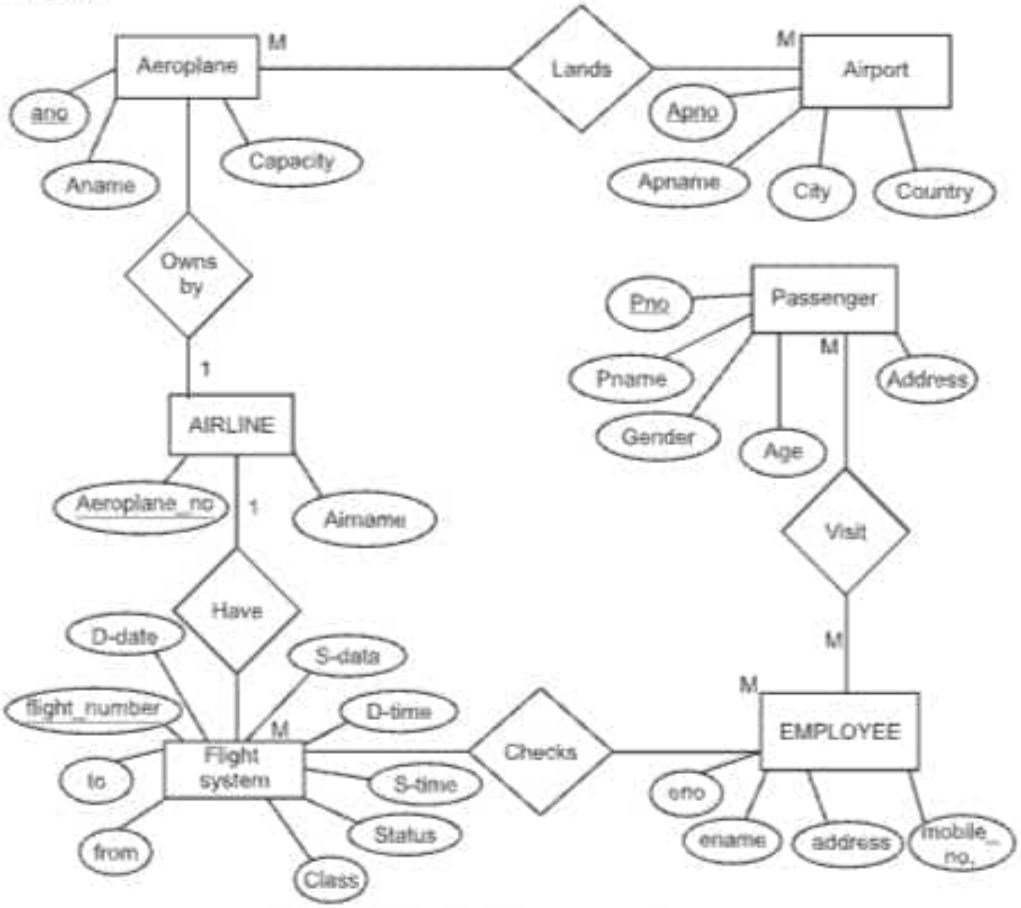


Fig. 2.48: Airlines Reservation System

Case Study 10: In online hotel reservation system, customer makes reservation and pays the payment. Through reservation customer books the rooms and services are provided by the employees of hotel to the customer. Draw ER diagram.

Solution:

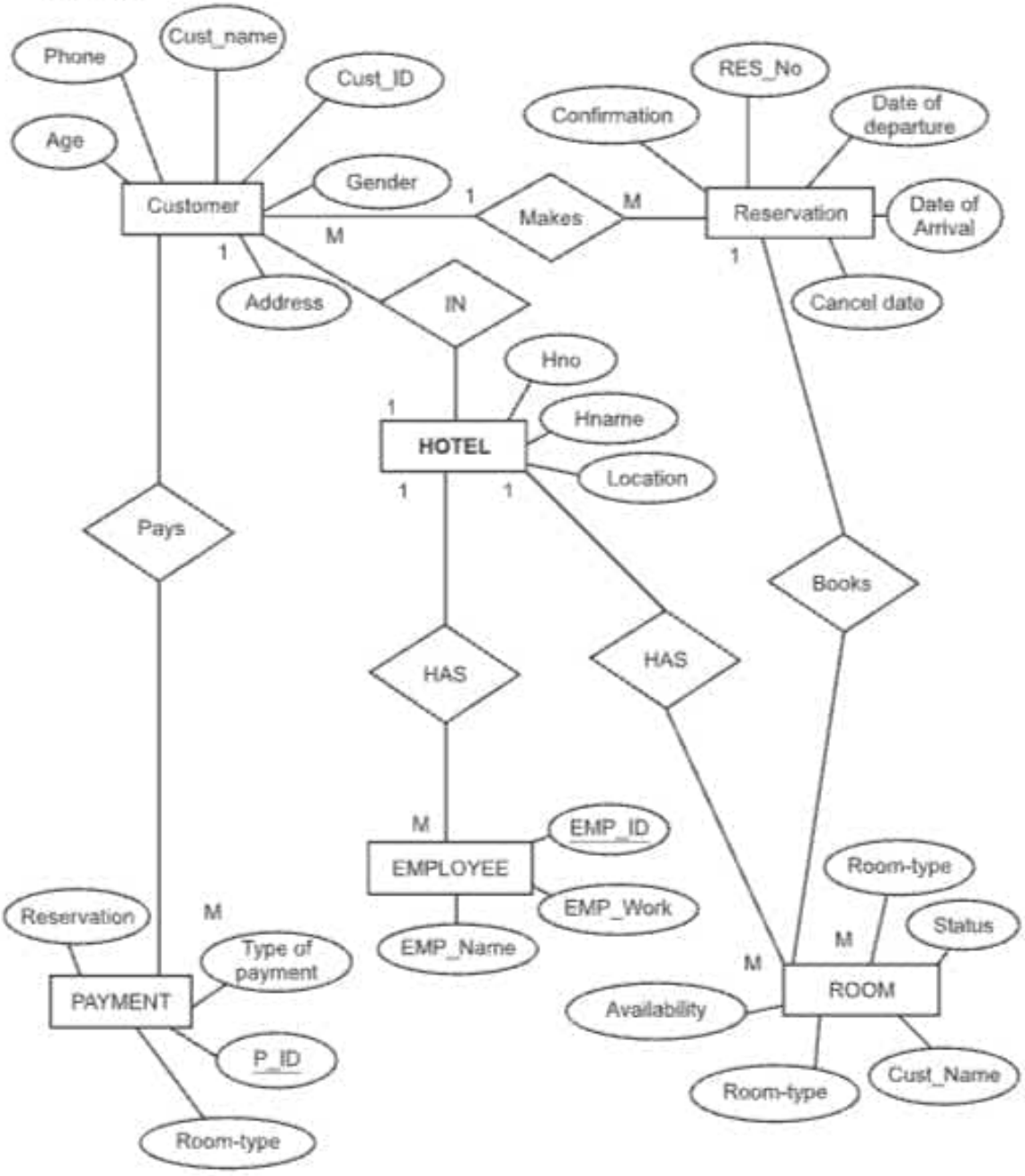


Fig. 2.49: Online Hotel Reservation System

Summary

- A DBMS (Database Management System) is a complex software system that is used to manage, store and manipulate data.
- Data can be defined as a representation of facts, concepts or instructions. Information is the processed data on which few divisions or actions can be taken.
- Data dictionary is centralized repository of information about data such as meaning, relationships, other data, origin, usage and format.
- The users of database system can be classified on the basis of degree of expertise or the mode of their interactions with DBMS.
- Data abstraction means hiding the implementation details from the end users. DBMS uses the same principle.
- Data independence means the property to change the overall logical or physical structure of the data without changing the application programs view of data.
- *Relation* means reference and relationship means connection. There are four types of relationship: One-to-One, One-to-Many, Many-to-One, Many-to Many.
- An E-R model is normally expressed as an E-R diagram which is a graphical representation of a particular database.
- Specialization is the abstracting process of introducing new characteristics of objects to create one or more new classes of objects.
- Aggregation is the process of combining information on a object so that the higher level objects can be abstracted.

Check Your Understanding

1. Data is better organized and is easily accessible by using .....
   (a) DBMS (b) FMS
   (c) EMS (d) all
2. Entity-relationship model is widely used in .....
   (a) Database implementation (b) Database direction
   (c) Database processing (d) Database design
3. The entity relationship set is represented in E-R diagram as .....
   (a) Diamond (b) Double diamonds
   (c) Dashed lines (d) Rectangles
4. Which of the following is an advantages of database ?
   (a) reduction in redundancy (b) avoidance of inconsistency
   (c) security enforcement (d) all above
5. In ..... databases we have a strict parent_child relationship only.
   (a) hierarchical (b) network
   (c) relational (d) all

Ans.: (1) a (2) d (3) a (4) d (5) a

Practice Questions

Q.1 Answer the following questions in brief.

1. What is DBMS ?
2. What is data abstraction ?
3. What are different types of Database users ?
4. What are the data models ?
5. What is E-R modeling ?

Q.2 Answer the following questions.

1. What are the functions of DBMS ?
2. State the advantages of DBMS.
3. What is data abstraction ? What are different levels of data abstractions ?
4. What do you mean by view of data ? What are its different types ?
5. What is data independence ? What are its different types ?

Q.3 Define terms

Instance, Primary key, Data dictionary, Data independence, Data Model, Entity, Attribute, Unary Relationship, Aggregation, Generalization

Previous Exams Questions

Summer 2015

1. List various users of DBMS and specify their jobs. **4 M**

Ans. Refer to Section 2.7.

2. Explain advantages and disadvantages of DBMS. **4 M**

Ans. Refer to Section 2.6.

3. Explain the term Entity. **4 M**

Ans. Refer to Section 2.10.1.

4. Explain entity and attributes and explain its types. **4 M**

Ans. Refer to Section 2.10.1 and 2.10.2.

5. A reputed general hospital has decided to computerized its operations. In hospital many doctors are working. The patients are admitted to the hospital into the room. There they are treated by various doctors. Sometimes patients have to undergo certain pathological tests, which are carried out in the labs. **8 M**

   A database should provide following details:

   (i) Identify all entities.

   (ii) Identify all relationship.

   (iii) Draw E-R diagram.

Ans. Refer to Section 'Case-Studies' Section.

Summer 2016

1. Explain the various advantages disadvantages of DBMS. **4 M**

Ans. Refer to Section 2.6

2. What do you mean by data model ? Explain entity relationship data model with example. **4 M**

Ans. Refer to Sections 2.9 and 2.10.

3. What do you mean by Data Model ? Explain network model with example. **4 M**

Ans. Refer to Sections 2.9 and 2.9.2.

4. Explain the following terms: **(1 M each Q)**
   - Record - Refer to Section 2.2.5.
   - Data dictionary - Refer to Section 2.2.3.

5. Explain the overall structure of system. **4 M**

Ans. Refer to Section 2.12.

6. Explain the following terms with example: **4 M**
   (i) Generalization.
   (ii) Specialization.

Ans. Refer to Section 2.11.

7. Attempt the following: **8 M**
   (a) An insurance agent sells insurance policies to clients. Policies can be of different types such as Vehicle Insurance, Life Insurance, Accident Insurance etc. The agent collects monthly premiums on the policies in the form of cheques of local bank. Database should provide the various details to the user.
   (i) Identify all entities.
   (ii) Identify all relations.
   (iii) E-R Diagrams.

Ans. Refer to Section 'Case Studies'.

Winter 2016

1. Differentiate between Network model and Relational model. (4M)

Ans. Refer to Section 2.9.2.

2. Differentiate between specialization and generalization. (4M)

Ans. Refer to Section 2.11.

3. Write a note on Relational model and Hierarchical model. (4M)

Ans. Refer to Section 2.9.2.

4. 'ABC' is ice-cream manufacturing firm having many distributors located in different area. Marketing manager would like to know the demand of products in market and so what discount strategies should be followed and also how much sale perform by each marketing representative. A database should provide the following details: **(8)**

(i) Identify all entities.
(ii) Identify all relationship.
(iii) Draw E-R diagram.

Ans. Refer to Section 'Case Studies'.

Summer 2017

1. Explain the applications of DBMS. **4 M**

Ans. Refer to Section 2.4.

2. Differentiate between network model and relational model. **4 M**

Ans. Refer to Section 2.9.2.

3. List the various users of DBMS and specify their jobs. **4 M**

Ans. Refer to Section 2.7.

4. What are the functions of Database Administrator ? **4 M**

Ans. Refer to Section 2.7.1.2.

5. Explain the advantages and disadvantages of DBMS. **4 M**

Ans. Refer to Section 2.6.

6. What are strong and weak entities ? **4 M**

Ans. Refer to Section 2.13.

7. Explain the term Entity **(1 M)**

Ans. Refer to Section 2.10.1.

8. "Star" is an agency for flat booking and it has number of builders and agents who are jointly working. A customer can get a flat for residential or commercial purpose. If customer is approached through an agent the agency and builders are giving some commission to the agent. Agent shows various flats and sites within various location.

   Study the case and do the following: **[8 M]**

   (i) Identify all entities.
   (ii) Identify all relationships.
   (iii) Draw E-R diagram.

Ans. Refer to Section 'Case Studies'.

Winter 2017

1. Explain the role of a DBA.

Ans. Refer to Section 2.7.1.2.

2. Write a short note on Data Abstraction.

Ans. Refer to Section 2.8.

3. What are the advantages and disadvantages of a DBMS ?

Ans. Refer to Section 2.6.

4. What are the different types of relationship ?

Ans. Refer to Section 2.10.4.

5. Explain Data Models and explain any one in detail. **4 M**

Ans. Refer to Section 2.9.

6. Differentiate between Specialization and Generalization. **4 M**

Ans. Refer to Section 2.11.

7. Design database for banking enterprise which records information about customers, employees of bank. A customer can be a depositor or a borrower.
   An employee of the bank can be customer of bank.
   There are two types of accounts, saving and current account.
   A database should provide the following details: **8 M**
   (i) Identify all entities.
   (ii) Identify all relationship.
   (iii) Draw E-R Diagram.

Ans. Refer to Section 'Case Studies'.

Summer 2018

1. State and explain the advantages of DBMS. **4 M**

Ans. Refer to Section 2.6.

2. Explain the views of Database Management System. **4 M**

Ans. Refer to Section 2.8.

3. Aggregation and Generalization. **4 M**

Ans. Refer to Section 2.11.

4. Define terms: Entity, attribute, superkey, tuple. **2 M**

Ans. Refer to Section 2.10.3.

❖❖❖

3...

Relational Model

Learning Objectives...

- To understand basic concepts of Relational Model.
- To get familiar with basic terms used in Relational Data Model.
- To know different types of keys of relation.
- To learn Relational Algebra Operations.

3.1 INTRODUCTION

- Most of the database software uses tree or complex structures for storing the databases.
- But these methods are quite complex to manage the relationships between the database.
- There is a simple and elegant method available called **Relational Data Model**.
- The Relational Data Model (RDM) stores the data in the form of a table.

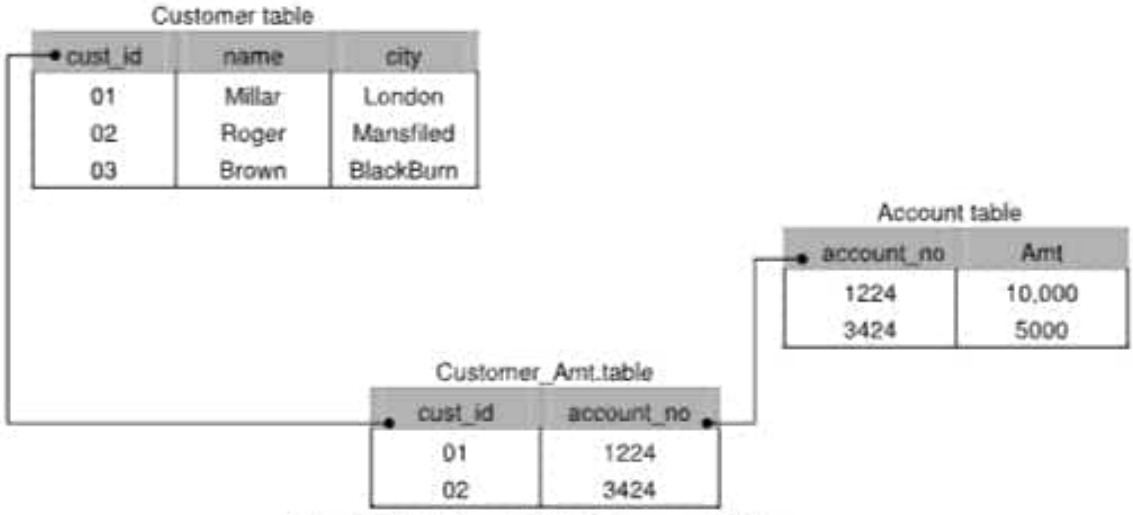


Fig. 3.1: Relational Model Representation

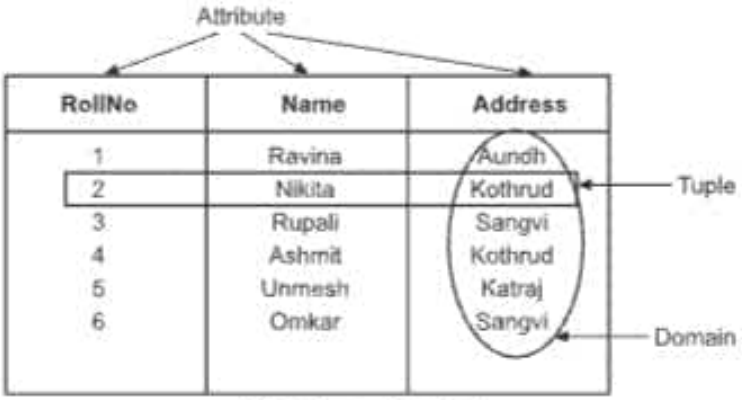
- Another important feature of RDM is that a single database can be spread across several tables.
- In this case collection of tables represents both data and the relationship.
- Each table consists of multiple columns and each column is recognized by a unique name.
- Let us consider two tables: Customer table and Account table.
- The relationship between these two tables have been shown by a third table, where one field or column name from both the table is taken together.
- So a new table called Customer_Amt is formed.

Advantages and Disadvantages of RDM

- The advantages of relational model are as follows:
  1. The RDM database can be used with the computer that has limited memory and processing capability.
  2. RDM is easier to use.
  3. RDM enables a computer system to accommodate a variety of enquiries in an efficient manner.
  4. RDM is useful for small databases.
  5. RDM improves the performance of a system because it is concerned with data and not with the structures. Changes in the database structure do not influence the data access.
- The disadvantages of RDM are as follows:
  1. RDM hides the physical data storage details from the user and it also hides implementation complexities.
  2. In RDM, we need not know how data is actually stored. This ease of design may lead to bad design.
  3. As it is easy to use and implement, large number of departments or people will create their databases, which may lead to data inconsistency, data duplication and data redundancy.

3.2 RELATIONAL DATA MODEL TERMS (W- 16, 17, S-18)

- There are various terms which are used in relational data model. The terms are shown in following diagram.
- The relational data model is the most popular model in DBMS because it is easy to understand by all types of users and conceptually simple.
- This model is developed in 1970, by Dr. E. F. Codd.
- He described this model to simplify the database structure.
- This model is based on relation, a two dimensional table. A table consists of rows and columns.



Relation = Student
Cardinality = 6 i.e. 6 records/tuples
Degree = 3 i.e. 3 attributes
Primary key = RollNo

Fig. 3.2: Relational Data Model

- A row called **tuple** represents one record and a column represents an attribute of an **entity** of a database.
- This model shows collection of table to represent both data and the relationship. (See Fig. 3.3).
- A table is a file. It is unified set of data consisting of collection of records.
- A database is a structured as a set of files. In the Fig. 3.3 database 1, 2, and 3 are three files which are showing relationship between each other through a field or attribute called as *social security number*.

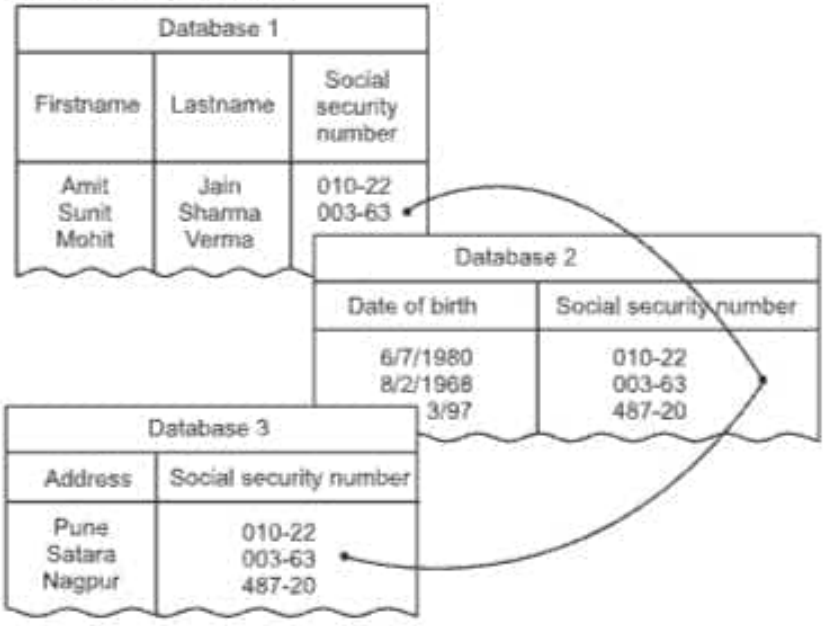


Fig. 3.3: Relational Database

(a) Relation: (W-17)

- A relation r of the relation schema R (A_1, A_2, ..., A_n) is a set of n-tuples r = (t_1, t_2, ..., t_n), which has the ordered list of n values < v_1, v_2, ..., v_n >, where each value v_i is a special value or a null value.
- In other words, we can say that a relation is nothing but a table of values.
- A record is set of attributes (field) and a possible value is defined for each attribute. Records of each type form a table or relation.

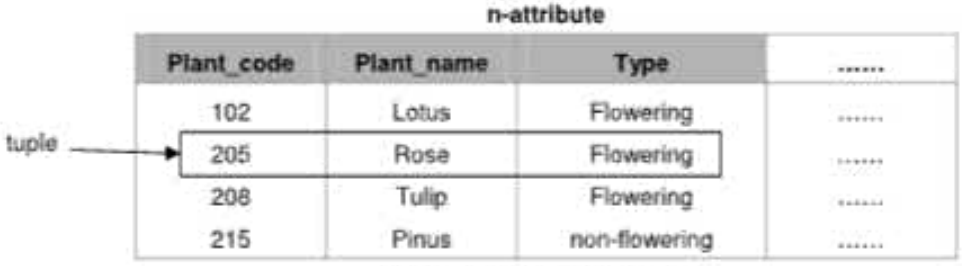


Fig. 3.4: A Relation (table)

(b) Tuple:

- A **tuple** is one row in a table or one record in a table. A **relation** is therefore referred as a set of tuples.

  For example:

Sr. No.	SName	City
1	Adams	London
2	Blake	London
3	Jones	Paris
4	Smith	Sydney
5	Clark	Paris

Tuple

Fig. 3.5: Tuples

(c) Attribute:

- In general, attribute is a property or characteristic of an entity. For example, colour is an attribute of your hair.
- An **attribute** corresponds to a column or header of a table.

  For example:

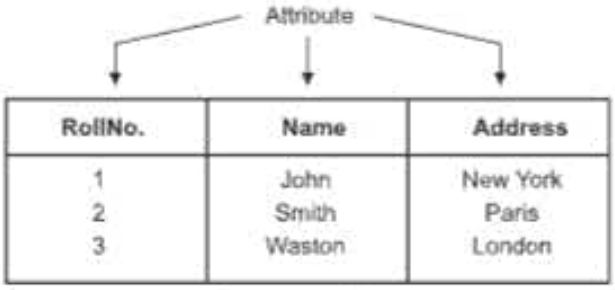


Fig. 3.6: Attribute

Type of Attribute: (W-17)

- **Single valued Attribute:** An attribute, that has a single value for a particular entity is known as single valued attribute. For example, age of an employee entity.
- **Multi-valued Attribute:** An attribute that may have multiple values for the same entity is known as multi-valued attribute. For example, colors of a car entity.
- **Compound Attribute/Composite Attribute:** Attribute can be subdivided into two or more other Attribute. For Example, Name can be divided into First name, Middle name and Last name.
- **Simple Attributes/Atomic Attribute:** The attributes which cannot be divided into smaller subparts are called simple or atomic attributes. For example, age of employee entity
- **Stored Attribute:** An attribute, which cannot be derived from other attribute, is known as stored attribute. For example, BirthDate of employee.
- **Derived Attribute:** Attribute derived from other stored attribute is called derived attribute. For example, age is derived attribute and birth data is stored attribute.

(d) Cardinality:

- The number of tuples (records) in a relation is called **Cardinality**.
- This shows the uniqueness of data values contained in a column.

  For example: If there are 10 tuples or records in a table then cardinality of that table is 10.

(e) Degree of Relationship Set:

- A relationship is an association among several entities.
- The number of entity sets that participate in a relationship set is called degree of Relationship set.
- Generally, most relationship sets are binary. But there are cases where higher degree relationship sets occur.

  For example: Degree = 3 is a ternary relationship.

(f) Domain:

- A domain determines the type of data values that permitted for that attribute. In other words, we can say that a **domain** is a pool of values from which specific attributes of specific relations draw their actual values.

  For example: A domain called address is a set of city names.

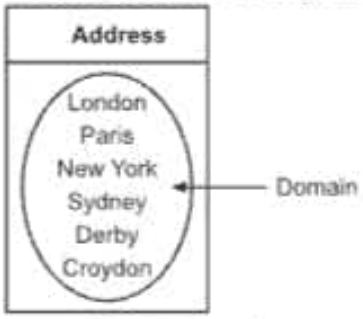


Fig. 3.7: Domain

3.3 DR. E. F. CODD'S 12 RULES

- In 1985, Dr. E. F. Codd first published this list of rules. Dr. E. F. Codd has introduced 12 rules for the relational model for databases popularly known as Codd's rules. The rules are as follows:

Rule 0: Foundation Rule

- This rule states that all next rules are based on the idea which states that, for a database to be considered relational, it must use its relational facilities entirely to manage the database.

Rule 1: Information Rule

- All the information in the database is to be represented in just one way, i.e. by values in tables.

Rule 2: Guaranteed Access Rule

- Each item of data in RDBMS must be accessible with no ambiguity by resorting to a combination of table name, primary key values and column name.

Rule 3: Systematic Treatment of Null Values Rule

- Null values (distinct from an empty character string or a string of blank characters and distinct from zero or any other number) are supported in a fully relational DBMS for representing missing information in a systematic way, independent of data type.

Rule 4: Dynamic On-line Catalog Based on the Relational Model Rule

- The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.

Rule 5: Comprehensive Data Sublanguage Rule

- A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible: data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, transaction boundaries (begin, commit, rollback).

Rule 6: View Updating Rule

- All views of the data which are theoretically updateable must be updateable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

- The system is able to insert, update and delete operations fully. It can also execute the operations on multiple rows simultaneously.

Rule 8: Physical Data Independence Rule

- Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

Rule 9: Logical Data Independence Rule

- Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

Rule 10: Integrity Independence Rule

- Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

Rule 11: Distribution Independence Rule

- The RDBMS has distribution independence. It implies that users should not have to be aware of whether a database is distributed.

Rule 12: Nonsubversion Rule

- If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

3.4 KEYS

- The rows (tuples/records) are unordered in a relation(table) and cannot be identified by its position in a relation(table).
- Every table must have some columns or combination of columns which uniquely identify each row in the table. For that, we require a key.
- In other words, a key is a relation of subset of attributes with following properties:
  1. The value of key is unique for each tuple.
  2. No data redundancy
- In a database every entity and the relationship between them is distinct. This difference is expressed in terms of their attributes.
- The concept of key allows making such distinction between two or more entities.

3.4.1 Super key (S-17,W-16)

- Super key is assigned to each entity set. Super key is a set of one or more attributes which is taken collectively to identify an entity in an entity set.
- Example: Consider a relation **Customer**, where social-security-number of the entity set customer is sufficient to distinguish one customer entity from another.
- Thus social-security-number is a superkey, (See Fig. 3.8).

Relation = CUSTOMER

Social_security_no	Cust_name	address
100 - 205	John	PARIS
103 - 102	Ashwin	NEWYORK
208 - 56	Raj	TORANTO

Superkey

Superkey

Fig. 3.8: Superkey

- Similarly, combination of Cust_name and Social_security_no is a super key for the entity set customer.
- Only Cust_name is not considered as a super key because many people can have same name.
- We are often interested in super key for which no proper subset is a super key.
- Such minimal super keys are called as **candidate keys** .

3.4.2 Candidate key (W-16)

- It is possible that several distinct sets of attributes could serve as candidate keys.
- Candidate key is a subset of super key whose no proper subset is a super key.
- It is set of one or more attributes which is used to identify a record.
- Candidate key should have following things:
  (a) It must be unique.
  (b) A candidate key's value must exist. It cannot be null.
- For example:
  1. Student_id is one of the candidate key.
  2. Consider relation customer where various employees were working and having unique identification number called as **Social Security Number** (SSN), (Refer Fig. 3.9)

Relation = CUSTOMER

Cust_name	SSN	BASIC
John	001 - 256	14000$
Jack	005 - 123	2000$
Smith	008 - 200	1000$
Jones	101 - 401	18000$
Mac	102 0 303	36000$

Candidate key

Fig. 3.9: Candidate Key

- SSN alone will work as a primary key or candidate key.
- Sometimes, we can consider two fields Cust_name and SSN as a key to identify a record then it is called as **Candidate Key**.
- In simple words, candidate keys has a primary key (Cust_name) and unique key which has a unique value only, (Refer Fig. 3.10)

Fig. 3.10: Example of Candidate Key

3.4.3 Primary Key (W-16, 17; S-18)

- The term primary key is used to denote a candidate key i.e. chosen by a database designer as the principal way of identifying entities within an entity set. Out of the multiple candidate keys for a table, only one candidate key can be primary key.
- A primary key uniquely identifies a record (a row) in a table. A primary key is made up of one column (simple primary key) or a combination of columns (Composite primary keys).
- The Primary key should contain only those columns which are needed to uniquely identify a record.
- For example:
  1. Consider relation ADVISOR where simple primary key *Advisor_ID* uniquely identifies records, (Refer Fig. 3.11).
  2. Consider relation STUDENT where each student can take more than one subject. Student_ID, Subject or marks alone does not returns a unique record, But if we combine Student ID and subject, it will identify a record, (Fig. 3.12).

Relation = ADVISOR

Advisor_ID	Name	Phone
01	Shirish	444333
02	Atul	323232
03	Mahesh	501011
04	Ravindra	890012

Fig. 3.11: Simple Primary Keys

Relation = STUDENT

Student_ID	Subject	Marks
101	Botany	95
101	English	90
105	English	82
106	Physics	74
106	Botany	98
107	Maths	94

Composite key

Fig. 3.12: Composite Primary Keys

3.4.4 Foreign Key (W- 16, 17, S-18)

- The attribute which acts as primary keys in one table will act as a foreign key in other table (Fig. 3.13).

- Foreign key can have duplicate values and it is used to search a record from two relations. For a foreign key column, the values in that column must refer to existing values in parent(link) table.

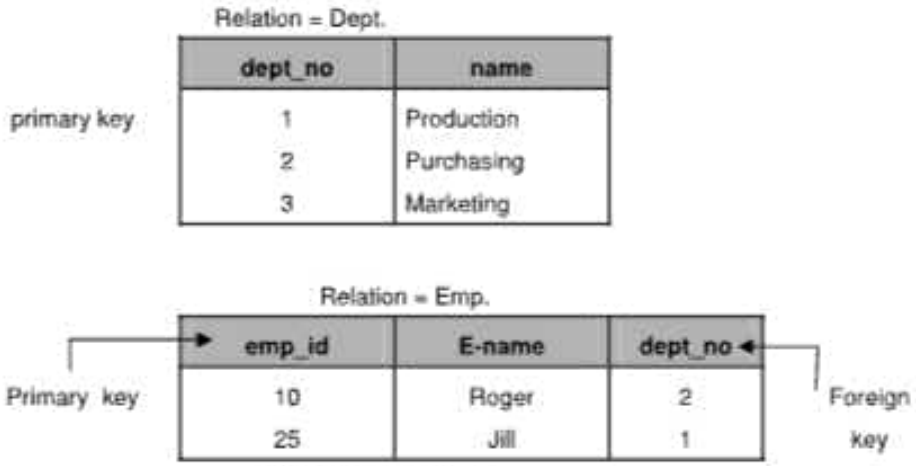


Fig. 3.13: Foreign Keys

3.4.5 Composite Key (W-16)

- There are situations when one attribute cannot form a key, because single attribute cannot uniquely identify every tuple or row in the table.
- So we need two or more attributes to identify tuple in the table.
- The key consists of two or more columns is called as *Composite Key*.
- For example: Consider a table of supplier which tells us the parts which supplier sells. Table is shown below:

Supid	Partid	Quantity	Price
S1	P1	12	300
S1	P2	10	100
S2	P2	5	50
S3	P3	7	200

- Here, neither the *Supid* nor the *Partid* can identify a row in the table uniquely.
- However, two of them together can easily identify a tuple or row in the table uniquely.
- Hence, composite key is (*Supid, Partid*).

3.4.6 Alternate Key or Secondary Key

- Alternate key or Secondary key are keys that may or may not identify a record uniquely, but help in faster scanning.
- In other words, Alternate key is any candidate key, which is not the primary key.

- **Significance of Alternate Key:**
  1. The DBMS creates and uses an index based on the alternate key, like primary key.
  2. Searching is faster because it is index.
  3. Alternate key can have duplicates.

Difference between Primary key and Secondary key:

Primary key	Secondary key
1. It has unique value.	1. Duplication is allowed.
2. There can be only one primary key per table.	2. Any number of alternate keys are possible.
3. A primary key has to be a candidate key also.	3. An alternate key may or may not be a candidate key.

3.5 RELATIONAL ALGEBRA OPERATIONS (S-17)

- Knowledge about relational algebra allows us to understand query execution and optimization in RDM.
- Each relational query describes a step by step procedure to compute desired answer.
- It is based on the order in which the operations are appeared in the query.
- The relational algebraic operations are divided into two groups.

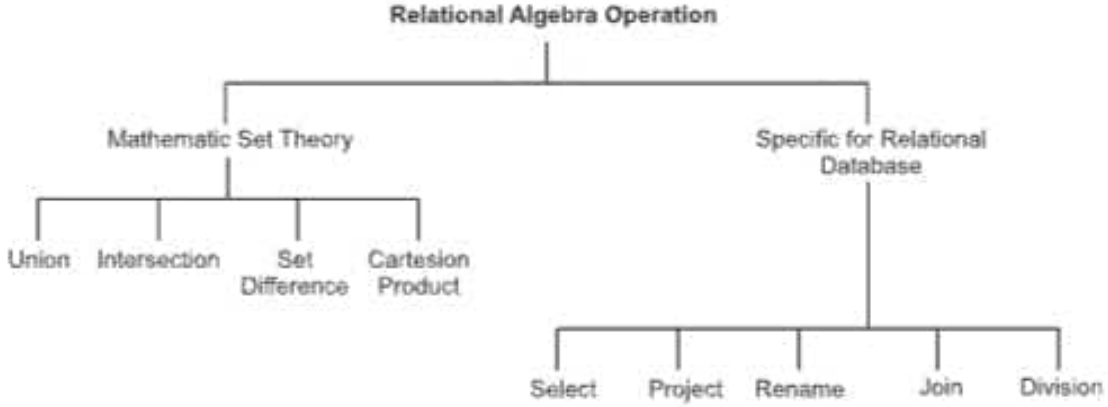


Fig. 3.14: Relational Algebra Operations

- There are some operations operate on a single relation called **unary operation** and other operation operate on two relations called **binary operation**.
  - Select (Unary)
  - Project (Unary)
  - Rename (Unary)
  - Product (Binary)

- Union (Binary)
- Difference (Binary)

- Before we go into the detail of operations, consider some important concepts here:
- Consider the following table:

A	B	C	D
a	b	c	l
e	b	g	h
i	j	k	l

→ Column (attributes)

Row (tuple) →

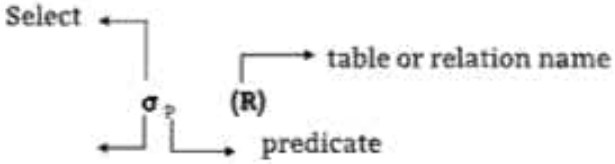
domain

- The column represents the attribute domain. The **domain** is a collection or set of all possible values for that column.
- For example: emp_id → Should be numeric value only; name; salary → Salary > = 0
- The following are the operations in detail:

(a) Select Operation **(W-16, 17)**

- This operation selects the tuples which satisfy a given predicate (condition). To represent this operation, lowercase Greek letter sigma (**σ**) is used.

Syntax:



- To specify the predicate we can use Relational or Comparison operators shown in Fig. 3.15 and certain logical operators shown in Fig. 3.16.

Operator	Meaning	Example
=	equal to	name = "Atul"
< >	not equal to	city < > "Pune"
>	greater than	age > 20
>=	greater than equal to	salary >= 20000
<	less than	profit < 1000
<=	less than equal to	marks <= 35

Fig. 3.15: Comparison Operators

Operator	Meaning	Example
AND	Logical AND	name = "Atul" AND city = "Pune"
OR	Logical OR	age = 18 OR age <= 42

Fig. 3.16: Logical Operators

- To explain the operations, we will consider a database or relation STUDENT, See Fig. 3.17.

Relation = STUDENT

Roll_no.	Name	Physics_marks	Chemistry_ marks	Maths _marks
1	Smith	70	82	86
2	Ashwin	45	56	59
3	Jack	88	60	65
4	Roger	95	87	72
5	Ketan	60	65	59

Fig. 3.17: STUDENT database

For example:

1. **Select all student details from STUDENT table where marks of physics are >= 70.**

Sol.:

Query → $\sigma_{Physics\_marks >= 70}$ (STUDENT)

Result →

Roll_no	Name	Physics_marks	Chemistry_ marks	Maths_marks
1	Smith	70	82	86
3	Jack	88	60	65
4	Roger	95	87	72

Fig. 3.18: Result

2. **Select all student details from STUDENT table where name is Ketan.**

Sol.:

Query → $\sigma_{name = "Ketan"}$ (STUDENT)

Result →

Roll_no	Name	Physics_marks	Chemistry_marks	Maths_marks
5	Ketan	60	65	59

Fig. 3.19: Result

(b) Project Operation (W - 16, 17)

- This is also referred as **projection**. It is a unary operation that returns certain columns from the table.
- This operation is useful to retrieve one column or more columns from a given table. The arity is number of columns in a table. This operation is denoted by a Greek letter Pi (π).

Syntax:

$\Pi_{a1,a2,\ldots,an}(R)$ where a1,a2,....,an are attributes and R stands for relation.

For example,

(1) List roll_no and physics_marks from table STUDENT.

Sol.:

Query → $\Pi_{Roll\_no, Physics\_marks}$ (STUDENT)

Result

Roll_no	Physics_marks
1	70
2	45
3	88
4	95
5	60

Fig. 3.20: Result

We can also merge select operation with project operation.

(2) List all the names having chemistry_marks less than 70.

Query: $\Pi_{Name, Chemistry\_marks}$ ($\sigma_{Chemistry\_marks <= 70}$ (STUDENT))

Result

name	Chemistry_marks
Ashwin	56
Jack	60
Ketan	65

Fig. 3.21: Result

(c) Union Operation (W- 16, S- 18)

- It is represented by ∪ in set theory.
- The union of two relations is formed by combining the tuple from one relation with those of a second relation to produce a third relation. For union, both tables must be of the same arity. Further the domain of i^{th} attribute of first and second relation must be same.

Syntax:

Table_name1 ∪ Table_name2

- In Union, duplicate tuples are eliminated, (Refer Fig. 3.22).

STUDENT1

roll_no	name	mark
1	Rohan	70
3	Jay	85
6	Akash	92
7	Ajit	57

STUDENT2

name	mark
Jay	85
Ajit	57
Anup	75
Biju	82
Rohan	70

Fig. 3.22: STUDENT1 and STUDENT2 Relations

- Here, both tables do not have same attributes. So we have to select common attribute.
- Query →

  Π_{name} (STUDENT1)

  Π_{name} (STUDENT2)
- Now we have to find union of these two sets. Therefore we can say,

  Π_{name} (STUDENT1) ∪ Π_{name} (STUDENT2)
- The result is depicted in Fig. 3.23.

name
Rohan
Jay
Akash
Ajit
Anup
Biju

Fig. 3.23: Result of Union query

- The result consists of 6 tuples even though there are 4 distinct student1 and 5 distinct student2. Here duplicate values are eliminated.

(d) Set Difference Operation

- This is represented by a minus (–) sign. The set difference of two relations is a third relation containing tuples which occurs in the first relation but not present in the second relation.
- If R and S are two relations the R - S is a set of tuples in R but not in S provided that R and S should have same arity as well as same name attributes.

Syntax:

Table_name1 – Table_name2

- Just like Union, Difference is also taken between compatible relations. Consider Student1 and Student2 relations.

  Query → Π_{name} (Student1) – Π_{name} (Student2)

  The result is depicted (Shown) in Fig. 3.24.

name
Akash

Fig. 3.24: Result of Difference query

- We will consider one more example

Emp _ 1

E_id	name
1	Manish
2	Chetan
3	David

(a)

Emp _ 2

E_id	name
1	Manish
4	Ajay

(b)

Fig. 3.25: Emp_1 and Emp_2 database

Query →

$\Pi_{e\_id, name}$ (Emp_1) – $\Pi_{e\_id, name}$ (Emp_2).

- The result is shown in Fig. 3.26.

e_id	name
2	Chetan
3	David

Fig. 3.26: Result of Difference query of Emp_1 and Emp_2

(e) Intersection Operation

- It is denoted by set intersection (∩). Let R and S be the relations. Then R ∩ S are set of tuples which are in R as well as in S.

  ∴ $R \cap S = R - (R - S)$

- It means that the intersection of two relations is a third relation containing common tuples.

- Consider Fig. 3.21 which has two relations.

  Query →

  Π_{name} (STUDENT1) $\cap$ Π_{name} (STUDENT2)

  Or

  Π_{name} ((STUDENT1) – (STUDENT1 – STUDENT2))

- Result is shown in Fig. 3.27.

name
Rohan
Jay
Ajit

Fig. 3.27: Intersection of Student1 and Student2.

(f) Cartesian Product Operation

- This is used to combine information from any two relations.
- It is denoted by '×' sign. If R and S are two relations, R × S is a relation with $K_1 \times K_2$ such that K_1 component of relation R × S are tuples in R while next K_2 components are tuples in S.
- Actually relation is defined as a subset of a Cartesian Product of a set of domains. If we consider two different tables and the attribute name is same then that attribute is distinguished by attaching the name of the relation from which the attribute originally came.

  R1 × R2

- Let's consider the example. If there are 5 tuples in Relation R1 and 2 tuples in Relation R2 then the number of tuples in R1 × R2 is 5×2 =10.
- Suppose we have following tables in a database.

Relation = STUD_NAME

Roll_No	Name
1	Brown
2	Roger
3	Jack
4	Lalonde
5	Mac

Relation = STUD_MARK

Roll_No	Mark
2	97
3	56
5	78

Fig. 3.28: Database

(1) If we want to do Cartesian product operation on these two tables, the result is as follows:

STUD_NAME × STUD_MARK

Roll_No	Name	Roll_No	Mark
1	Brown	2	97
2	Roger	2	97
3	Jack	2	97
4	Lalonde	2	97
5	Mac	2	97
1	Brown	3	56
2	Roger	3	56
3	Jack	3	56
4	Lalonde	3	56
5	Mac	3	56
1	Brown	5	78
2	Roger	5	78
3	Jack	5	78
4	Lalonde	5	78
5	Mac	5	78

Fig. 3.29: Cartesian product

(2) If we want to display list of all records where name of student is "Jack". Then the query is as follows:

Query: $\Pi_{roll\_no,\ name,\ mark}\ (\sigma_{name = "Jack"}\ (STUD\_NAME \times STUD\_MARK))$

Sol.: The result is drawn from the above table.

roll_no	name	roll_no	mark
3	Jack	2	97
3	Jack	3	56
3	Jack	5	78

Fig. 3.30: Result of name = "Jack" (STUD_NAME × STUD_MARK)

But, here we do not get proper result. So there should be comparison of common fields from both table.

(3) We again repeat the query 2.

Query: $\Pi_{roll\_no,\ name,\ mark}\ (\sigma_{STUD\_NAME.roll\_no\ =\ STUD\_MARK.roll\_no}\ (\sigma_{name\ =\ "Jack"}\ (STUD\_NAME \times STUD\_MARK)))$

Result

roll_no	name	mark
3	Jack	56

Fig. 3.31: Result of name = "Jack"

(g) Natural Join Operation (S-17, W-16)

- It is denoted by a symbol ⋈. The natural join operation is possible only when both R and S relations have at least one same name of attribute.
- For natural join find Cartesian product first. Then for each common attribute find the common elements, write them into new table.
- This new table is called as **Equijoin.** Remove the duplicate attribute and the result will be the **Natural Join.**
- Suppose we have two tables i.e. STUD_NAME AND STUD_MARK, (See Fig. 3.32).

Relation = STUD_MARK

roll_no	mark
1	60
2	80
3	90

Relation = STUD_NAME

roll_no	name
1	Rohan
2	Ajit
3	Ketan

Fig. 3.32: Database

- Let's find Cartesian Product:

$$\Pi_{\text{roll\_no, name, mark}} (\sigma_{\text{STUD\_NAME.roll\_no} = \text{STUD\_MARK.roll\_no}} (\text{STUD\_NAME} \times \text{STUD\_MARK}))$$

- The result is as shown in Fig. 3.33.

Roll_No	name	Roll_No	Mark
1	Rohan	1	60
2	Ajit	1	60
3	Ketan	1	60
1	Rohan	2	80
2	Ajit	2	80
3	Ketan	2	80
1	Rohan	3	90
2	Ajit	3	90
3	Ketan	3	90

Fig. 3.33: Result of Cartesian product

- To find the equijoin, find common elements of common attributes. Here, common attribute is roll_no. The equijoin result is shown in Fig. 3.34.

Roll_No	name	Roll_No	Mark
1	Rohan	1	60
2	Ajit	2	80
3	Ketan	3	90

Fig. 3.34: Result of Equi_Join

- Now remove the duplicate attribute so that we will get the result of natural join as depicted in Fig. 3.35.

Roll_No	name	Mark
1	Rohan	60
2	Ajit	80
3	Ketan	90

Fig. 3.35: Result of natural join

- The expression of natural join is as shown below:

  $\Pi_{roll\_no,\ name,\ mark}$(STUD_NAME ⋈ STUD_MARK)

- The Natural Join is a binary operation which allows to combine certain selections and a Cartesian Product into one operation. It is denoted by the symbol ⋈.

SOLVED CASE STUDIES

Example 3.1: Let's consider a database where Branch, Account, Customer and Loan are the relations available. There are many accounts in a branch. Customers and loans are related with Many-to-Many relationship. Similarly, Customer and Account are related with Many-to-Many. A branch can have many loans. The E-R diagram is as shown below in Fig. 3.36.

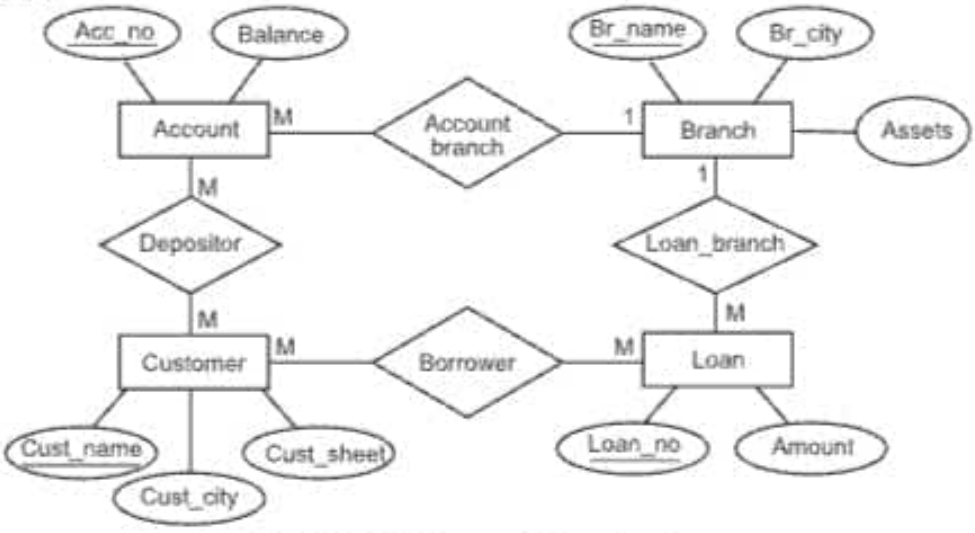


Fig. 3.36: E-R diagram for Bank system

- Now, let's write the relational algebra expressions for the following questions.

  1. Select a record from relation Loan where branch is Jhons Street.

  Sol.: $\sigma_{br\_name = \text{"Jhons Street"}}$ (Loan)

  2. Find name of all branches in the loan relation.

  Sol.: Π_{br_name} (Loan)

3. Find the branch in the loan relation where branch is Jhons Street and Loan amount is greater than 2,00,000

Sol.: $\Pi_{br\_name}(\sigma_{br\_name = \text{"Jhons Street"} \cap amt > 200000}(Loan))$

4. Find names of all customers who have either a loan or an account.

Sol.: $\Pi_{cust\_name}(Borrower) \cup \Pi_{cust\_name}(Depositor)$

5. Find all customers of the bank who have an account but not a loan.

Sol.: $\Pi_{cust\_name}(Depositor) - \Pi_{cust\_name}(Borrower)$

6. Find the name of all customers who have loan at Adams Street branch.

Sol.: Here, we need the information of both the relations i.e. Loan and Borrower. Because loan_no is the common attribute available so the common loan_no of Adams Street branch should get selected.

$\Pi_{cust\_name}(\sigma_{borrower.loan\_no = loan.loan\_no}(\sigma_{br\_name = \text{"Adams Street"}}(Borrower \times Loan)))$

7. Find the names of all customers who live on the same street and in the same city as Mac.

Sol.: $\Pi_{cust\_street,\ cust\_city}(\sigma_{cust\_name = \text{"Mac"}}(Customer))$

But to find other customer with this street and city, we must reference the customer relation second time.

$\Pi_{customer.cust\_name}$

$(\sigma_{customer.cust\_street = dummy.cust\_street \cap customer.cust\_city = dummy.cust\_city}$

$(customer \times \rho_{dummy(street,\ city)}$

$(\Pi_{cust\_street,\ cust\_city}(\sigma_{cust\_name = \text{"Mac"}}(Customer)))))$

8. Find customers who have both a loan and an account.

Sol.: $\Pi_{cust\_name(borrower)} \cap \Pi_{cust\_name}(Depositor)$.

9. Find the asset and name of all branches, which have depositors living in London.

Sol.: $\Pi_{br\_name,\ assets}(\sigma_{cust\_city = \text{"London"}}(Customer \bowtie Depositor \bowtie Branch \bowtie Account))$

10. Find all customers who have both an account and a loan at Bezant Street branch.

Sol.: $\Pi_{cust\_name}(\sigma_{br\_name = \text{"Bezant Street"}}(Branch \bowtie Account \bowtie Depositor \bowtie Customer)) \cap$

$\Pi_{cust\_name}(\sigma_{br\_name = \text{"Bezant Street"}}(Branch \bowtie Loan \bowtie Borrower \bowtie Customer))$

Example 3.2: Consider the following relational database:

DOCTOR (Doctor_no, Doctor_name, address, City)

Hospital (Hosp_no, hosp_name, street , H_city)

Doct_Hosp (Doctor_no, hosp_no, Date)

Construct following queries into relational algebra.

1.: Find out all the doctors who have visited to hospitals in the same city in which they live.

Sol.: $\Pi_{doctor\_name}(\sigma_{doctor.city = hospital.H\_city}(DOCTOR \bowtie Hospital \bowtie Doct\_hosp))$

2. : Find out to which hospital Dr. Will Smith has visited.

Sol.: $\Pi_{hosp\_name}(\sigma_{doctor\_name = \text{"Dr. Will Smith"}}(Doctor \bowtie Hospital \bowtie Doct\_hosp))$

Example 3.4: An orchestra database consists of the following database.

Conducts (conductor, composition)

Requires (composition, instrument)

Plays (player, instrument)

1. : List the composition conducted by Bran Adam who do not use violin.

Sol.: $\Pi_{composition}(\sigma_{conductor = \text{"Bran Adam"} \wedge instrument <> \text{"Violin"}}(Conducts \bowtie Requires))$

2.: List the players which are likely to play for the composition 'Last Symphony'.

Sol.: $\Pi_{player}(\sigma_{composition = \text{"Last Symphony"}}(Requires \bowtie Plays))$.

Example 3.5: Consider the following relational databases,

Student (S_no, S_name, B_data, Class)

Course (C_no, C_name)

Result (S_no, C_no, grade)

Construct the following queries in Relational Algebra.

(a) Find out student names who have secured grade 'A' in course 'DBMS'.

Sol. $\Pi_{student.s\_name}(\sigma_{result.grade = \text{"A"}} \sigma_{course.c\_name = \text{"DBMS"}}(Student \bowtie Course \bowtie Result))$

(b) Find out the grades of student 'Roger' along with course name.

Sol. $\Pi_{result.grade, course.c\_name}(\sigma_{student.s\_name = \text{"Roger"}}(Student \bowtie Course \bowtie Result))$

Example 3.6: The student database consists of the following database.

Student (Stud_no, S_name)

Teach (Prof, c_code, section)

Guides (Prof, Stud_no)

Enroll (Stud_no, c_code, section)

Construct the following queries into Relational Algebra.

1. **List all students taking course with prof. "Adam".**

Sol. $\pi_{s\_name,\ stud\_no}(\sigma_{prof = \text{"Adam"}}(\text{Enroll} \bowtie \text{Teach} \bowtie \text{Student}))$

2. **List all the project guides of the student named Alex.**

Sol. $\pi_{prof}(\sigma_{s\_name = \text{"Alex"}}(\text{Guides} \bowtie \text{Student}))$

Example 3.7: Consider the following database. Shown in Fig 3.38.

Employee

Fname	init	Lname	**SSN**	Bdate	Addr	Sex	Salary	Supur SSN	DNO

Department

Dname	**Dno**	mgrSSN	mgrstartdate

Dept_location

Dno	Dlocation

Project

Pname	**Pno**	plocation	Dnum

works_on

ESSN	**Pno**	Hour

Dependent

ESSN	Dependent_name	sex	Bdate	Relationship

Fig. 3.37: Database

Solved Queries:

1. : **Select all employee records whose department is 10.**

Sol. : $\sigma_{DNO = 10}(\text{Employee})$

2. : **Select employee records whose salary is greater than ₹ 30,000.**

Sol. : $\sigma_{salary > 30,000}(\text{Employee})$

3. : **Select all employees who either work in department 10 and make over ₹ 50,000 per year or work in department 5 and make over ₹ 60,000.**

Sol. : $\pi_{SSN}(\sigma_{(DNO = 10 \text{ and salary} > 50,000) \text{ OR } (DNO = 5 \text{ and salary} > (60,0000)}(\text{Employee}))$

4. : **List each employees first and last name and salary.**

Sol. : $\pi_{Lname,\ Fname,\ salary}(\text{Employee})$

5. : **Retrieve the first name, last name and salary of all employees who work in department number = 10**

Sol. : $\pi_{Fname,\ Lname,\ salary}(\sigma_{DNO=10}(\text{Employee}))$

6. : Retrieve the SSN of all employees who either work in department 10 or directly supervise an employee who work in department 10.

Sol. : $D_{10} \leftarrow \sigma_{DNO = 10}$ (Employee)

Result 1 $\leftarrow \pi_{SSN}$ (D_{10})

Result 2 $\leftarrow \pi_{superSSN}$ (D_{10})

Result 3 $\leftarrow$ Result 1 $\cup$ Result 2

7. : Retrieve the names of Manager of each department.

Sol. : Dept. MGR $\leftarrow$ Department $\bowtie$ mgrssn = SSN (Employee)

Result $\leftarrow \pi_{Dname, Lname, Fname}$ (Dept_MGR)

8: Retrieve department, the names & address of all employees who works for the 'Research' department.

Sol. : Research_Dept $\leftarrow \sigma_{Dname = 'Research'}$ (Department)

Reasearch_Emps $\leftarrow$ (Research_Dept $\bowtie_{Dno = DNO}$ (Employee))

Result $\leftarrow \pi_{Fname, Lname, Adress}$ (Research_Emps)

9 : For every project located in 'London', list the project number, the controlling department number and the department manager's last name, address and birth date.

Sol. : London_projs $\leftarrow \sigma_{Plocation = 'London'}$ (Project)

CONTR_Dept $\leftarrow$ (London_projs $\bowtie_{Dnum = Dno}$ Department)

Proj_Dept_MGR $\leftarrow$ (CONT_Dept $\bowtie_{MGRSSN = SSN}$ Employee)

Result $\leftarrow \pi_{Pno, Dnum, Lname, Addr, Bdate}$ (Proj_Dept_MGR)

Summary

- In a relational model, real world objects are represented in tables. Each table is made out of rows and columns. Each row known as tuple or record is made out of fields, which known as attributes. An attribute is defined by a name and its value.
- Each Attribute stands for a certain feature of the real world object. in database management system an attribute may describe a component of the database such as a table or a field.
- The number of tuples (records) in a relation is called **cardinality**. a **domain** is a pool of values from which specific attributes of specific relations draw their actual values.
- Key is used for identifying unique rows from table. A primary key is a special relational database table column (or combination of columns) designated to uniquely identify all table records.

- A super key is a set of one of more columns (attributes) to uniquely identify rows in a table. A super key with no redundant attribute is known as candidate key.
- Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.
- Relational algebra is a procedural query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data.
- Basic/Fundamental Operations are: Select, Project, Union, Difference, Cartesian Product.

Check Your Understanding

1. What is a foreign key?
   (a) A foreign key is a primary key of a relation which is an attribute in another relation.
   (b) A foreign key is a superkey of a relation which is an attribute in more than one other relations.
   (c) A foreign key is an attribute of a relation that is a primary key of another relation.
   (d) A foreign key is the primary key of a relation that does not occur anywhere else in the schema.
2. An attribute is a ..... in a relation.
   (a) Row (b) Column
   (c) Value (d) Tuple
3. In relational model terminology, table is considered as
   (a) range (b) domain
   (c) relation (d) tuple
4. Cardinality in relational data model is considered as .....
   (a) total number of values.
   (b) limited number of values.
   (c) two numbers from set.
   (d) three numbers from set.
5. Basic operations that can be performed on relations are .....
   (a) deletion.
   (b) insertion.
   (c) modification.
   (d) all of above.

   Ans. (1) c (2) b (3) c (4) a (5) d

Practice Questions

Q.1 Write the following questions in brief.

1. What is Relational Data Model ?
2. What is a relation ? Explain the degree of a relation with tuple ?
3. What are the advantages of RDM ?What is Primary Index.
4. Define foreign key.
5. What is Union operation in Relational agebra?

Q.2 Write the following questions.

1. Differentiate between primary key and foreign key.
2. Explain the following operations from relational algebra.
   (i) Select (ii) project
   (iii) Union (iv) Cartesion product
   (v) Intersection (vi) Natural join
3. What is a select operation ? How it is represented ? Explain with an example.
4. Enlist the steps followed for conversion of E-R to a relational model.
5. What is a attribute ? Explain with its types.

Q.3 Define the Terms

Relation, tuple, Attribute, cardinality, domain, super key, candidate key, primary key, entity, Degree of relationship set.

Previous Exams Questions

Summer 2015

1. Explain primary key and foreign key with suitable example. **4 M**

Ans. Refer to Section 3.4.

2. Explain select and project operation in Relational Algebra. **4 M**

Ans. Refer to Section 3.5.

3. Define term **1 M**
   (i) Domain.

Ans. Refer to Section 3.2 (f).

4. Attempt the following. **4 M**
   Consider relation database:
   customer (cust-no, cust-name, address, city)
   Loan (loan-no, loan-amt, loan-date, cust_no)
   Customer and loan are related with one to many relationship.
   Write relational algebraic expression for the following:
   (i) List loan details of customer name as "Mr. Shinde".
   (ii) List names of customer who have taken loan of amount more than 50,000.
   (iii) List names of customer who stay in city "Mumbai".
   (iv) Display customer with loan amount greater than 1,00,000.

Ans. Refer to Section 'Solved Case Studies'.

Summer 2016

1. Define the term: **4 M**
   (i) Attribute.

Ans. Refer to Section 3.2 (c).

(ii) Cardinality.

Ans. Refer to Section 3.2 (d).

(iii) Primary key.

Ans. Refer to Section 3.4.

(iv) Foreign key.

Ans. Refer to Section 3.4.

2. What is Relational Algebra ? Explain UNION and CARTESIAN PRODUCT operation example. **4 M**

Ans. Refer to Section 3.5.

3. What is an attribute ? Explain various types of attribute with example. **4 M**

Ans. Refer to Section 3.2 (c).

4. Attempt the following.
   Consider the following entities and their relationship. **8 M**
   ALBUM (id, name, category, release-year)
   SINGER (sid, name, address)
   Write relational algebraic expression for the following:
   (i) Display customer names having quotation for 'LCD'.
   (ii) List all the customer bearing quotation dated '20-May-2010'.
   (iii) List all the customers who like in 'M.P.' or 'U.P'.
   (iv) Display customers of amt_quoted as ₹ 15,000.

Ans. Refer to Section 'Solved Case Studies'.

5. Consider the following relational database: **8 M**
   Customer (cno, cname, city)
   Quotation (qno, qdate, description, amt_quoted, cno) customer and quotation are related with one-many relationship.
   Write relational algebraic expression for the following:
   (i) Display customer names having quotation for 'LCD'.
   (ii) List all the customer bearing quotation dated "20 May 2010'.
   (iii) List all the customers who live in 'M.P.' or 'U.P.'.
   (iv) Display customers of amt_quoted as ₹ 15,000.

Ans. Refer to Section 'Solved Case Studies'.

Winter 2016

1. Explain the concept of Primary key and Foreign key with suitable example. **4 M**

Ans. Refer to Section 3.4.

2. Explain select and project in the relational algebra with example. **4 M**

Ans. Refer to Section 3.5.

3. What is Domain and Cardinality ? Explain with suitable example. **4 M**

Ans. Refer to Sections 3.2 (c) and 3.2 (f).

4. Define terms: **1 M**

(i) Attributes

Ans. Refer to Section 3.2.

(iii) Degree

Ans. Refer to Section 3.2 (e).

5. Explain Union and Natural Join in relational algebra with example. **4 M**

Ans. Refer to Section 3.5.

6. Attempt the following.

Consider the following database and write the queries into relational algebra. **8 M**

Suppliers (Sid, Sname, address)

Parts (Pid, Pname, colour)

Catalog (Sid, Pid, cost)

(i) Find the name of suppliers who supply some yellow parts.

(ii) Find names of all parts whose cost is more than ₹ 40.

(iii) Find name of all parts whose colour is red.

(iv) Find name of supplier and parts with its colour and cost.

Ans. Refer to Section 'Solved Case Studies'.

Summer 2017

1. Explain Unary and Binary relations. **4 M**

Ans. Refer to Section 3.5.

2. Explain the Natural join and Cartesian product with example. **4 M**

Ans. Refer to Section 3.5.

3. Consider relational database: **8 M**

Supplier (Supno, sname, supaddress)

Item (Itemno, Iname, stock)

Supp Item (Supno, Itemno, rate)

Write relational algebraic expression for the following:

(a) List all suppliers from 'Varanasi' city who supplies PISTON.

(b) Display all suppliers supply PISTON RINGS.

(c) Change supplier names to upper case.

(d) List all suppliers supplying DOOR Lock from 'Jaipur' city.

Ans. Refer to Section 'Solved Case Studies'.

Winter 2017

1. Explain select and project operations in Relational Algebra. **4 M**

Ans. Refer to Section 3.5.

2. Explain the terms: **1 M Each Q**

(1) Tuple.

Ans. Refer to Section 3.2.

(2) Cardinality.

Ans. Refer to Section 3.2 (d).

(3) Relation.

Ans. Refer to Section 3.2 (a).

3. Explain entity and attributes and explain its types. **4 M**

Ans. Refer to Section 3.2.

4. Explain primary key and foreign key with suitable example. **4 M**

Ans. Refer to Section 3.4.

5. Attempt the following. **8 M**

Consider relational database:

Movie (mvno, mvname, release year)

Actor (actno, actname)

Movie and actor are related with many-to-many relationship.

Write relational algebraic expression for the following (any four):

(i) Display actorwise movie details.

(ii) Display all actor details of movie 'Airlift'.

(iii) Count all the movie names released in the year 2015.

(iv) Add 'age' column to Actor table.

(v) Display all movies of actor 'Amitabh'.

(vi) Change the actor name from 'Ranbir' to 'John'.

Ans. Refer to Section 'Solved Case Studies'.

Summer 2018

1. Explain Primary key and Foreign key with example. **4 M**

Ans. Refer to Section 3.4.

2. Explain union and difference in Relational algebra with suitable example. **4 M**

Ans. **Refer** to Section 3.5.

3. Attempt the following. **8 M**

Consider the database and write relational algebraic expression

Patient Master (PatientNo. PatientName, Sex, Address City, Allergy, Chief Complaints).

(i) Display all patients whose Allergy is "Nimesulide".

(ii) Display all male patients from city Calcutta.

(iii) Update all patients whose sex is "M" with "Male".

(iv) List all patients whose chief complaint is "fever".

Ans. Refer to Section 'Solved Case Studies'.

❖❖❖

4...

(SQL) Structured Query Language

Learning Objectives...

- To get family with SQL.
- To know about DDL and DML commands.
- To learn how to create simple and nested queries.
- To use aggregate functions within query.

4.1 INTRODUCTION

- A query language for relational model is based on relational algebra. It provides a notion for representing queries.
- The commercial database system requires a more *user-friendly* query language. *SQL i.e. Structured Query Language*, it uses a combination of relational algebra and relational calculus constructs.
- SQL contains other capabilities besides querying a database.
- These capabilities includes features for defining the structure of the data, features for modifying data in the databases and features for specifying security constraints.
- SQL is used as a standard relational database language because it allows user to access data in RDBMS (Relational Database Management System) such as Oracle, Sybase, Informix, Postgres etc.
- The user can write a SQL statement and submit it to the DBMS.
- DBMS will retrieve the appropriate data from disk and return it to the user.

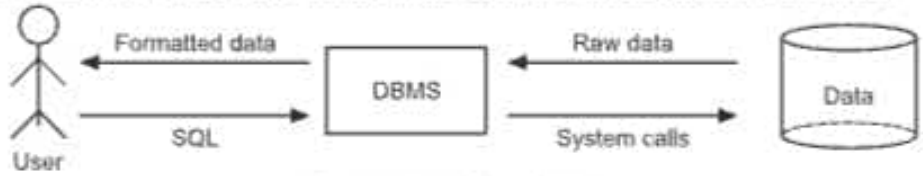


Fig. 4.1: Working of DBMS

- From 1986, SQL has become universally adopted language.
- The non-relational database systems also supports SQL interface. SQL is a special purpose, non-procedural language that supports the definition, manipulation and control of data in RDBMS.
- It is called as special-purpose because only the database can be handled.
- The SQL has several parts:
  - (a) **Data Definition Language (DDL):** The DDL provides commands for defining structure of relations, deleting relations, creating indexes and modifying the relations.
  - (b) **Data Manipulation Language (DML):** The DML includes a query language based on relational algebra and tuple relational calculus. It includes the command like insert, delete and modify the tuples in a database.
  - (c) **View definition:** The SQL DDL includes commands for defining views. A view is a relation (virtual rather than base) and can be used in query expressions. It means that the queries can be written using the views as a relation.
  - (d) **Embedded SQL:** The embedded form of SQL is designed which is used within general purpose programming language such as COBOL, C, C++, Java, Fortran and Pascal.
  - (e) **Integrity:** The SQL DDL includes commands for specifying integrity constraints which a data source in a database has to satisfy.
  - (f) **Authorization:** The SQL DDL includes commands for satisfying access rights to the relations and views.
  - (g) **Transaction control:** SQL includes the commands for specifying the beginning and ending of transaction. It also provides locking of data of concurrency control.
- Even though SQL has various parts, we will consider only DDL which allows us to create and maintain data manipulation objects such as tables, views, sequence.
- Views are used for security purpose. The sequence is for creating numeric values.

4.2 HISTORY OF SQL

- There are numerous versions of SQL. The original version was developed at IBM's San Jose Research Laboratory by Dr. E. F. Codd in early 1970.
- This was based on the relational database model.
- After that IBM's research division and Donald Chamberlin developed a prototype language called SEQUEL i.e. Structured English Query Language.
- This language was expanded and revised to SEQUEL/2 which was referred as application programming interface.
- For some legal reason, IBM eventually changed the name SEQUEL/2 to SQL (Structured Query Language). In 1986, ANSI and ISO had declared SQL as a standard language.

- Therefore sometimes, it is referred as **Standard Query Language**.

Advantages of SQL:

1. **Simplicity:** Many problems can be expressed in SQL more easily and concisely than in lower level languages. Simplicity in turn means increase in productivity.
2. **Completeness:** The language is relationally complete. User can write very large class of queries.
3. **Non-procedurality:** SQL DML are non-procedural languages.
4. **Data independence:** SQL DML statements include no reference to explicit access paths such as indexes or physical sequence.
5. **Ease of extension:** It can be easily extended by using built-in functions.
6. **Support for higher level languages:** SQL or some other language of comparable power, can conveniently be used as a common target for those higher languages. It means it will work as intermediate step in the translation of those languages to the language of the underlying machine.

4.3 BASIC STRUCTURE

- A relational database consists of a collection of relation, each of which is assigned a unique name.
- For retrieving record from table, we use basic structure for data retrieval.
- This SQL expression consists of 3 clauses or components.
- These components are as follows:

  SELECT < list of attributes >

  FROM < table name >

  WHERE < condition >;
- A SQL query has the following form:

  SELECT $A_1, A_2, \ldots\ldots, A_n$

  FROM $r_1, r_2, r_3, \ldots\ldots, r_m$

  WHERE p;
- The A_n represents attributes, r_m represents the relation name and p is predicate [condition].
- This query is equivalent to the relational algebra expression.

$$\pi_{A_1, A_2, \ldots\ldots, A_n} (\sigma_p (r_1 \times r_2 \times \ldots\ldots \times r_m)).$$

- Sometime, the list $A_1, A_2, \ldots\ldots, A_n$ may be replaced with an asterisk (•) to select all attributes of all relations appearing in the FROM clause.
- SQL forms the Cartesian product of the relation named in the FROM clause, performs a relational algebra selection using the WHERE clause predicate and projects the result onto the attribute of the SELECT clause. The semicolon (;) is the terminator of SQL expression.

4.4 DATA DEFINITION LANGUAGE (DDL) COMMANDS (S-17, W-17)

- DDL is a part of SQL which describes the structure of information in the database.
- DDL basically allows us to create, delete and alter(modify) tables.

4.4.1 Data Types (S-18)

- Let's learn some basic data types used in SQL for table creation.
- The attributes which we define in the table must have a data type. The data types supported by SQL are:
  - Char
  - Varchar
  - Number
  - Int or integer
  - Float or real
  - Date and Time

1. Char:

- We can use *char* data type to denote a fixed-length string. The number we define with char is to show that fixed memory of those many characters is allowed for that data type.
- For example, *name char (10)*:
- Here, memory of 10 bytes will be stored for the name as shown below in Fig. 4.2.

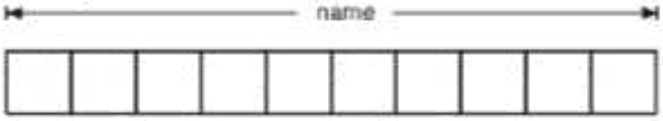


Fig. 4.2: Char (10) for name

- Even if we accept or write a name of 6 character. The utilization of memory is for 6 characters. But rest 4 bytes are not used for other purpose. It is memory wastage, (See Fig. 4.3).

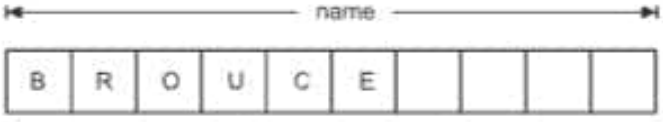


Fig. 4.3: Shows Wastage of Memory

2. Varchar:

- To avoid wastage of memory, we must use varying data type called varchar. Varchar stands for varying length character.
- Varchar (n) denotes a string upto n characters. Sometimes, varchar 2(n) is used.
- For example,

```
name varchar(10);
```

- If we accept or write name of 6 character; Examples: BROUCE, only 6 bytes will be used. Rest 4 bytes will be utilized for other purpose.

3. **Number:**

- This data type represents a decimal number.
- For example,

```
rollno number(3);
```

- Here, 3 is representing the number of digits present. If we want we can write the above example as,

```
rollno number(3, 2);
```

- Here, 2 represents that there are 2 digits present after a decimal point.

4. **Int or Integer:**

- Both these names are synonymous which denote the integer value.
- For example,

```
rollno integer;
```

5. **Float or Real:**

- We can use float or real as they are synonyms words. Using this we define floating numbers.
- We can write a fixed decimal point using decimal(n, d) i.e. n decimal digits and d position from the right.
- For example,

```
123.45 is a possible value of type decimal(6, 2).
```

6. **Date and Time:**

- The dates and times can be represented by the data type DATE and TIMESTAMP. They are treated as character values so written into a single character.
- The date is a data type available which has components year, month and day in the form of DD-MON-YY.
- TIMESTAMP allows you to store date and time data which has year, month, day, hour, minute and second. Additionally it stores fractional seconds.
- For example,

```
Date_of_Birth date;
```

4.4.2 Create Table Command (W-17)

- The simple form using which we can declare a relation scheme in data definition in SQL is through **create** statement.
- This statement is used to create a new table.
- The SQL relation is defined by using create table command by giving a table name and specifying attributes and constraints.

Integrity Constraints:

- Integrity Constraints are used to apply business rules for the database tables. The constraints available in SQL are Foreign Key, Not Null, Unique, Check. Constraints can be defined in two ways:
  (1) The constraints can be specified immediately after the column definition. This is called column-level definition.
  (2) The constraints can be specified after all the columns are defined. This is called table-level definition.

(1) Primary key:

- This constraint defines a column or combination of columns which uniquely identifies each row in the table.

  Define a Primary key at column level:

  Syntax:

  column name datatype PRIMARY KEY

  Define a Primary key at table level:

  Syntax:

  [CONSTRAINT constraint_name] PRIMARY KEY
  (column_name1,column_name2,..)

- **column_name1, column_name2** are the names of the columns which define the primary key.
- The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

  For Example: To create an employee table with Primary Key constraint, the query would be like.

  Primary Key at Column Level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

or

```
CREATE TABLE employee
( id number(5) CONSTRAINT emp_id_pk PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
```

```
location char(10)
);
```

Primary Key at Table Level:

```
CREATE TABLE employee
( id number(5),
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT emp_id_pk PRIMARY KEY(id)
);
```

Primary Key at table level:

```
CREATE TABLE employee
( id number(5) NOT NULL,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10));
ALTER TABLE employee ADD CONSTRAINT PK_EMPLOYEE_ID PRIMARY KEY (id);
```

(2) Foreign key or Referential Integrity:

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between two different tables. For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.

Syntax to define a Foreign key at Column Level:

```
Column_name REFERENCES
Referenced_Table_name(column_name);
```

Syntax to define a Foreign key at Table Level:

```
[CONSTRAINT constraint_name] FOREIGN KEY(column_name) REFERENCES
referenced_table_name(column_name);
```

For Example:

Foreign Key at Column Level:

```
CREATE TABLE Product
( product_id number(5) PRIMARY KEY,
product_name char(20),
supplier_name char(20),
```

```
unit_price number(10)
);
CREATE TABLE Order_items
( order_id number(5) PRIMARY KEY,
product_id number(5) REFERENCES product(product_id),
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```

Foreign Key at Table Level:

```
CREATE TABLE order_items
( order_id number(5) ,
product_id number(5),
product_name char(20),
supplier_name char(20),
unit_price number(10)
CONSTRAINT od_id_pk PRIMARY KEY(order_id),
CONSTRAINT pd_id_fk FOREIGN KEY(product_id) REFERENCES product(product_id)
);
```

- If the employee table has a 'mgr_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the SQL query would be like,

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
mgr_id number(5) REFERENCES employee(id),
salary number(10),
location char(10)
);
```

(3) NOT NULL Constraint:

- This constraint ensures all rows in the table contain a definite value for the column which is specified as not null.

Syntax to define a NOT NULL constraint:

```
Column_name NOT NULL
```

For Example: To create a employee table with Null value, the query would be like

```
CREATE TABLE Employee
( id number(5),
name char(20) NOT NULL,
dept char(10),
```

```
age number(2),
salary number(10),
location char(10)
);
```

(4) UNIQUE Key:

- This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.

Syntax to define a Unique key at Column Level:

```
Coloumn_name datatype [CONSTRAINT constraint_name] UNIQUE
```

Syntax to define a Unique key at Table Level:

```
[CONSTRAINT constraint_name] UNIQUE(column_name)
```

For Example: To create an employee table with UNIQUE key, the query would be like,

Unique Key at Column Level:

```
CREATE TABLE Employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) UNIQUE
);
```

or

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) CONSTRAINT loc_un UNIQUE
);
```

Unique Key at Table Level:

```
CREATE TABLE Employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT loc_un UNIQUE(location)
);
```

(5) CHECK Constraint:

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax to define a CHECK constraint:

```
[CONSTRAINT constraint_name] CHECK (condition)
```

For Example: In the Employee table to select the gender of a person, the query would be like,

Check Constraint at Column Level:

```
CREATE TABLE Employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1) CHECK (gender in ('M','F')),
salary number(10),
location char(10)
);
```

Check Constraint at Table Level:

```
CREATE TABLE Employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1),
salary number(10),
location char(10),
CONSTRAINT gender_ck CHECK (gender in ('M','F'))
);
```

DEFAULT Values:

A column in a table can be assigned a default value. In case if no value is specified for this column, then the column is filled with its respective default value. If no default value is declared for a column, then the default value for that column is the NULL value.

Example:

```
CREATE TABLE item (
it_code number PRIMARY KEY,
it_name char NOT NULL,
it_desc char,
rate number DEFAULT 10
);
```

Example 4.1: Create a table Customer with name, city name and street name.

Query:

```
Create table Customer
(cust_name varchar2(20) NOT NULL,
cust_city varchar2(20) NOT NULL,
cust_street varchar2(40));
```

This will create a table with name Customer. By using INSERT command, we can insert the records into the table.

Consider that the table Customer is related with a table Depositor. Account is one more table which has *account number* as primary key and it is also related with depositor. The Fig. 4.4 shows the relationship diagram.

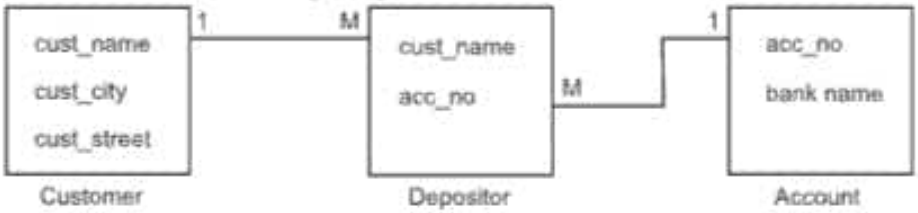


Fig. 4.4: Many-to-Many Relationship

In customer table, cust_name is a primary key. To create a table customer with primary key, we need to write the following create command.

Example 4.2: Create a table Customer with name as primary key, city name and street name.

Query: Create table Customer

```
(cust_name varchar2(20) NOT NULL,
 cust_city varchar2(20) NOT NULL,
cust_street varchar2(40) NOT NULL,
constraint PK_cust_name primary key
(cust_name));
```

Example 4.3: Create a table depositor where cust_name is a foreign key and acc_no is also foreign key. Acc_no is from table account which has been already created.

Query: Create table Depositor

```
(cust_name varchar2(20) NOT NULL,
acc_no int NOT NULL,
constraint FK_ cust_name Foreign Key(cust_name)
references customer(cust_name),
constraint FK _ acc_no Foreign Key(acc_no)
references account(acc_no));
```

UNIQUE constraint or candidate key is also added through create.

Example 4.4: Create table project with primary key Pnumeric and Foreign key Dnum.

Query: Create Table Project

```
(Pname varchar(15) NOT NULL,
Pnumber int NOT NULL,
Dnum int NOT NULL,
constraint PK_ Pnumber primary Key(Pnumber),
Unique (Pname),
constraint FK_ Dnum Foreign Key(Dnum) references Dept(Dnumber));
```

Difference between Primary Key and Unique Constraints :

1. A table can have only one primary key, but it can have many unique constraints.
2. When a primary key is defined, the column that compose the primary key are automatically mandatory. But with the unique constraint you must have to specify the column as NOT NULL.

4.4.3 Drop Table Command

- When the condition changes, it becomes unnecessary to maintain the information in the table.
- So we need to delete a table.
- If a table is not needed any more, the table should get deleted.
- For this we use drop command.
- It removes the structure of the table as well as the data contained in it.

Syntax: DROP table <*tablename*>

For example: Drop table Account;

- It will delete table account from the database. Delete is used to delete tuples from the table where as drop is used to delete the whole table (structure of the table as well as the data within it)

4.4.4 Alter Table Command (S-18)

- Without deleting a relation we may forcefully modify the existing relation.
- These modifications can be performed by the statement called **Alter Table**.
- The most important options are ADD and MODIFY.
- The Alter Table command is used to add attributes to an existing relation.
- The newly added attribute has all **NULL** values at the start. After that we can insert values for it.

Syntax:

```
Alter table table_name
Add (new_column datatype(size), ....);
```

[**Note:** Variation in ALTER TABLE Command:

Clauses (in Oracle):

- ADD : To add a new column to an existing table.
- MODIFY : To modify an existing column of an existing table.

Clauses (in Postgres):

- ADD : To add a new column to an existing table.
- DROP : To delete a column from a table.

- ALTER COLUMN column SET DATA TYPE datatype : To change the data type of a column.
- ALTER COLUMN column SET DEFAULT default_value : To set default value for a column of a table.]

Example 4.5: Add a new column Adate in table account.

Sol.:
```
Alter table account
Add (Adate date);
```

If we want to modify data type of an existing column then in place of ADD use MODIFY keyword.

Example 4.6: Modify acc_no in Account table.

Sol.:
```
ALTER TABLE Account
    MODIFY (acc_no number (2));
```

4.4.5 Views

- View is a virtual (logical) table built on base tables(actual tables).

Syntax:
```
CREATE  [OR REPLACE] VIEW view_name AS
SELECT column_1, column_2,.., column_n
FROM tables
[WHERE condition];
```

- The clauses written in square brackets are optional. If OR REPLACE is specified, it would update the definition of the view already present with the same name without dropping it.
- If we omit the OR REPLACE clause, the CREATE VIEW statement will display an error message.
- Example: Suppose we have a table emp with fields empno, empname, addr, phno, sal, deptno. We can create a view empview with empno, addr fields for the employees living at address='main road' as follows:

```
CREATE OR REPLACE VIEW empview
      AS SELECT empno, addr  FROM emp WHERE addr='main road';
```

- View can be dropped using DROP VIEW statement.

Syntax:
```
DROP VIEW view_name;
```

4.4.6 Indexes

- Indexes help to speed up data retrieval. An index in a database is exactly same as an index in the back of a book.

Syntax:
```
CREATE INDEX index_name ON table_name(column_name);
```
Example: Suppose we want to define an index on empname field of emp table.
```
CREATE INDEX ename ON emp(empname);
```
An index can be dropped by using DROP INDEX command.

Syntax:
```
DROP INDEX ename;
```

4.5 DATA MANIPULATION LANGUAGE (DML) COMMAND (S-18)

- The SQL DML includes the commands to insert tuples into database, to delete tuples from the table and to modify tuples in the table.

4.5.1 Insert Command

- To insert a data into a relation, we either specify a tuple to be inserted or write a query whose result is set of tuples to be inserted.

Syntax:

```
INSERT into table-name (column-name [, column-name] ...)
VALUES (column-value [, column value] .... )
```

Example 4.7: Insert values into Depositor table.

Query:
```
INSERT INTO Depositor (cust_name, acc_no)
VALUES ('Brouce', 558);
```

The result is shown in depositor table as shown in Fig. 4.5.

cust_name	acc_no
Smith	443
Jill	998
Zara	443
Smith	880
Alex	112
Brouce	558

Fig. 4.5: Result of insert query

- The data type of columns and respective values must match. According to the syntax of insert, column list is an optional element. It is optional only when the sequence of the data types of the values passed through insert statement and the sequence of the data types of columns of the relation match. Therefore, the above example could be written as:

Query:
```
INSERT INTO depositor
VALUES ('Brouce', 558);
```

- Sometimes, we want to insert tuples on the basis of result of a query. Suppose we want to provide all loan customers in the London branch. Let the loan number serves as the Account Number for the new savings account, we write following query.

Query:
```
insert into depositor
   select  branch_name, loan_no, cust_name
from borrower
where branch_name = 'London';
```

- It will first find the result of select and then that result will be inserted.

4.5.2 Delete Command

- This is used to delete the tuple. We cannot delete values on only particular attributes, we have to delete complete record. In SQL, a delete is expressed by the following syntax.

Syntax:

```
DELETE  FROM tablename
   [WHERE condition];
```

- The optional part is represented in [ ]. Delete without the "WHERE" part will empty the table completely.

Example 4.8: Delete all records for depositor.

Query: `DELETE FROM depositor;`

- This will delete all records from depositor table. Note that a delete command operates on only one relation or table. If we want to delete tuples from several relations, we must write different delete commands with the table name.
- Sometimes, we do not want to delete all the tuples. Then we can apply where clause with certain condition. Depending upon the condition related record will get deleted.

Example 4.9: Delete all the records where name of the customer is 'Smith'.

Query:
```
DELETE FROM depositor
   WHERE cust_name = 'Smith';
```

The results is shown in Fig. 4.6.

cust_name	acc_no
Jill	998
Zara	443
Alex	112

Fig. 4.6: Result of query

Example 4.10: Delete all the loans with loan number between 4 to 7.

Query:
```
DELETE from borrower
   WHERE loan_no BETWEEN 4 AND 7;
```

The result is shown in Fig. 4.7.

cust_name	loan_no
Jill	1
Mac	3
Anna	2

Fig. 4.7: Result of query 4.10

Example 4.11: Delete all the accounts at branches located in London.

Query:
```
DELETE FROM depositor
    WHERE branch_name IN
      (SELECT branch_name
      FROM branch
      WHERE branch_city = 'London');
```

- This delete requests first, finds all the branches in London and then deletes all deposit tuples for those branches. Note that, we may delete tuples from only one relation at a time but we may reference any number of relations in a SELECT-FROM-WHERE nested in the where clause of delete.
- The execution of delete requests just mark the tuples to be deleted. Once, it finishes processing the request i.e. once all the tuples were marked for deletion, then only delete command actually deletes all the marked tuples.

4.5.3 Update Command

- In some situations, we may wish to change a value in a tuple without changing all the values in the tuple.
- For this purpose, the update statement can be used.
- The syntax of update command is as follows:

Syntax:
```
UPDATE table-name
SET column-name = Expression [, column-name = Expression, ……]
[WHERE condition];
```

- The update command will update or change values of specified column-name with the entered expression for all tuples.
- This will be stopped by providing optional WHERE clauses.

Example 4.12: Update all the branches with increase of 5% annual interest.

Query:
```
UPDATE account
SET balance = balance * 1.05;
```

- This statement is applied once to each tuple in depositor. The result is as shown in Fig. 4.8.

acc_no	branch_name	balance
112	Willam Street	10500
443	Adams Street	5250
553	Willam Street	2100
880	Adams Street	15750
998	Bezant Street	21000

Fig. 4.8: Result of update query

Example 4.13: Update account with balances over $ 10000 by 6% interest and other by 5%.

Query:
```
UPDATE account
SET balance = balance * 1.06
WHERE balance > 10000;
UPDATE account
SET balance = balance * 1.05
WHERE balance ≤ 10000;
```

Here, we can write two update statement.

The result of this is depicted in Fig. 4.9 depending on Fig. 4.8.

acc_no	branch_name	balance
112	Willam Street	11130.00
443	Adams Street	5512.50
553	Willam Street	2205.00
880	Adams Street	16695.00
998	Bezant Street	22260.00

Fig. 4.9: Result of update with where clause

- For tuple 1, 4 and 5 i.e. acc_no 112, 880 and 998 the interest is calculated as 6% and for other it is calculated as 5%.

4.6 DATA CONTROL LANGUAGE (DCL)

- Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges defines the access rights provided to a user on a database object. There are two types of privileges.
  - **System:** Creating session, table etc are all types of system privilege.
  - **Object:** Any command or query to work on tables comes under object privilege.

  DCL has two commands,

1. **GRANT:** Gives user access privileges to database.

   The syntax for this command is defined as follows:

   ```
   GRANT [privilege]
     ON [object]
       TO [user]
       [WITH GRANT OPTION]
   ```

   Where,

   - **privilege** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.

- **object** is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- **user** is the name of the user to whom an access right is being granted.
- **WITH GRANT OPTION** - allows a user to grant access rights to other users.

For example:

GRANT SELECT ON employee TO user1;

This command grants a SELECT permission on employee table to user1.

GRANT SELECT, UPDATE, DELETE ON employee TO user1;

- This command grants SELECT, UPDATE and DELETE permission on employee table to user 1.

2. **REVOKE:** Take back permissions from user.

The syntax for this command is defined as follows:

```
REVOKE [GRANT OPTION FOR] [permission]
  ON [object]
  FROM [user]
```

For example:

REVOKE SELECT ON employee FROM user1;

- This command will revoke a SELECT privilege on employee table from user1.

REVOKE SELECT, UPDATE, DELETE ON employee FROM user1;

- This command will revoke a SELECT, UPDATE, DELETE privilege on employee table from user 1.

Transaction Control Language Commands (TCL)

- This section give you an insight into the commands which are used to manage transactions in the database. The commands are as follows:

```
COMMIT
ROLLBACK
SAVEPOINT
COMMIT
```

- This command is used to save the transaction into the database.

Syntax:

```
COMMIT;
ROLLBACK;
```

- This command is used to restore the database to the last committed state.

Syntax:

```
ROLLBACK;
```

NOTE: When you use ROLLBACK with SAVEPOINT, then you can directly jump to a savepoint in an ongoing transaction.

Syntax: `ROLLBACK TO SavepointName;`

```
SAVEPOINT;
```

- This command is used to temporarily save a transaction. So if you wish to rollback to any point, then you can save that point as a 'SAVEPOINT'.

  Syntax:

  ```
  SAVEPOINT SAVEPOINTNAME;
  ```

- Consider the below example to understand the working of transactions in the Employee database.

ID	Name
01	Ruhaan
02	Suhana
03	Aayush
04	Rashi

- Now, use the below SQL queries to understand the transactions in the database.

```
1  INSERT INTO Employee_Table VALUES(05, 'Avinash');
2  COMMIT;
3  UPDATE Employee_Table SET name = 'Akash' WHERE id = '05';
4  SAVEPOINT S1;
5  INSERT INTO Employee_Table VALUES(06, 'Sanjana');
6  SAVEPOINT S2;
7  INSERT INTO Employee_Table VALUES(07, 'Sanjay');
8  SAVEPOINT S3;
9  INSERT INTO Employee_Table VALUES(08, 'Veena');
10 SAVEPOINT S4;
11 SELECT * FROM Employee_Table;
```

Output to the above set of queries would be as follows:

ID	Name
01	Ruhaan
02	Suhana
03	Aayush
04	Rashi
05	Akash
06	Sanjana
07	Sanjay
08	Veena

4.7 SIMPLE SQL QUERIES

- Lets consider the relational database shown in Fig. 4.10. Underlined attributes are the primary keys.

Branch (branch_name, branch_city, assets)
Customer (cust_name, cust_street, cust_city)
Loan (loan_no, branch_name, amount)
Account (acc_no, branch_name, balance)
Borrower (cust_name, loan_no)
Depositor (cust_name, acc_no)

Fig. 4.10 : Relational database (Banking System)

- In above Fig. 4.10, the relations (tables) branch and loan are related with One to Many relationship.
- Branch and Account are also related with One-to-Many relationship.
- Customer and Loan are related with Many-to-Many relationship.
- Similarly, Customer and Account are related with Many-to-Many relationship.
- Therefore, Borrower and Depositor are the two tables created to show Many-to-Many relationship.
- The E-R diagram is created from the given schema which is shown in Fig. 4.11.

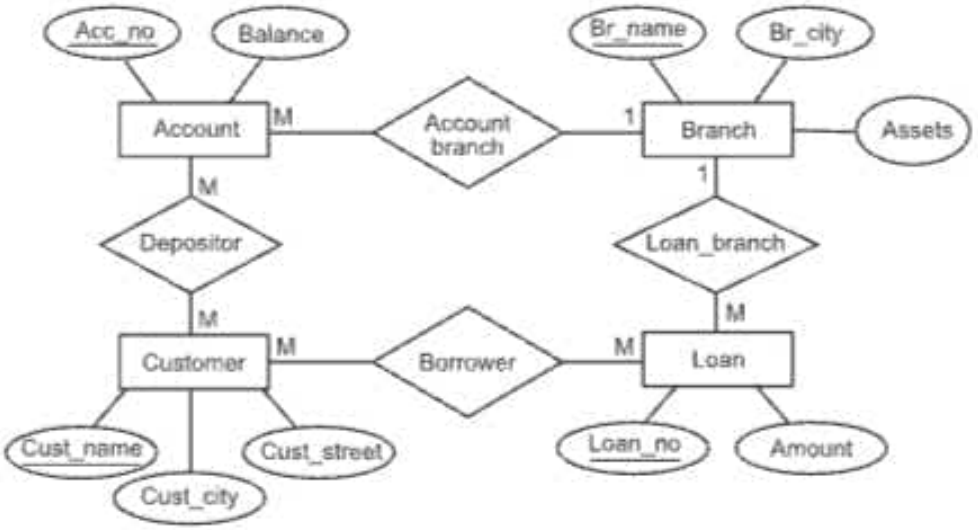


Fig. 4.11: E-R diagram for Banking System

- When we want to write simple queries the relationships should be created. The relationship between different entities is shown in Fig. 4.12 which helps us to understand the above E-R diagram better.

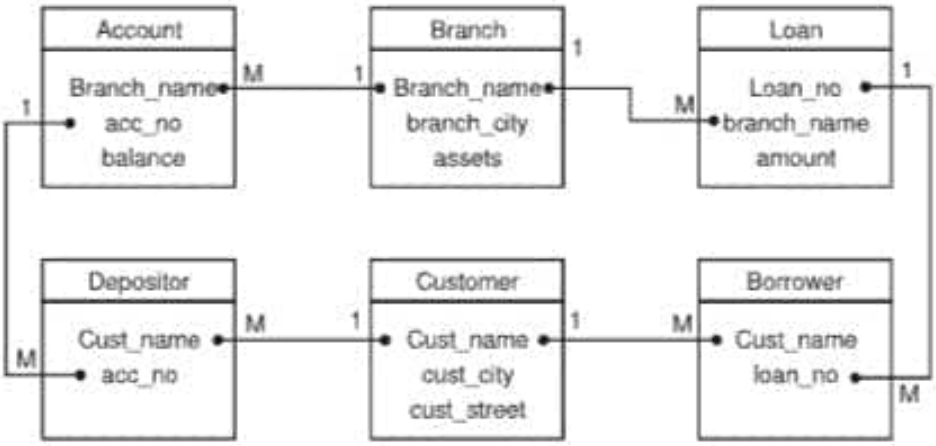


Fig. 4.12: Relationship between entities

- To understand the queries, let's maintain all the above tables (entities) with some data in the form of records, Refer Fig. 4.13.

Relation = Loan

Loan_no	Branch_name	Amount
1	Jhons Street	200000
2	Jhons Street	500000
3	Adams Street	680000
4	Willam Street	520000
5	Bezant Street	250000
6	Adams Street	100000
7	Adams Street	100000

(a)

Relation = Borrower

Cust_name	Loan_no
Jill	1
Mac	3
Alex	5
Anna	2
Jill	6
Anna	7

(b)

Relation = Branch

branch_name	branch_city	assets
Jhons Street	London	100000
Willam Street	London	500000
Adams Street	London	200000
Bezant Street	New York	120000

(c)

Relation = Account

acc_no	branch_name	balance
112	Bezant Street	10000
443	Adams Street	5000
553	Bezant Street	2000
880	Adams Street	15000
998	Willam Street	20000

(d)

Relation = Customer

cust_name	cust_city	cust_street
Smith	New York	Willam Street
Jill	London	Jhons Street
Mac	London	Adams Street
Van	London	Adams Street
Zara	New York	Willam Street
Alex	London	Bezant Street
Anna	London	Bezant Street

(e)

Relation = Depositor

Cust_name	acc_no
Smith	443
Jill	998
Zara	443
Smith	880
Alex	112

(f)

Fig. 4.13: Tuples in all tables

4.7.1 Select Clause

Example 1: Find name of all branches in the loan relation:

Query: Π_{branch_name} (Loan)

- The same query can be written in following syntax:

(a) SELECT branch_name
FROM Loan;

- Here; is a terminator which shows the end of a query. The result of this query is shown in Fig. 4.14 (a).

branch_name
Jhons Street
Jhons Street
Adams Street
Bezant Street
Willam Street
Adams Street
Adams Street

Fig. 4.14 (a) : Result of Select Query

- It displays the duplicate values also. If we want to force the elimination of duplicate values, insert a keyword **distinct** before a field name.
- By default SQL takes a keyword **all** and that's why it displays all the values.
- The syntax with distinct is as follows:

(b) SELECT distinct branch_name
FROM Loan;

- The result of this query is as shown in Fig. 4.14 (b).

branch_name
Jhons Street
Adams Street
Bezant Street
Willam Street

Fig. 4.14 (b): Result of Select with Distinct

- SQL allows us to use the keyword **all** to specify explicitly that duplicates are not removed. The query is as follows:

(c)
```
SELECT all branch_name
    FROM loan;
```

- The result of this query is similar to the result depicted in Fig. 4.14 (a).

Example 2: Find all the attributes in loan relation

Query: $\Pi_{branch\_name,\ loan\_no,\ amount}$ (loan)

OR

```
SELECT loan_no, branch_name, amount
FROM Loan;
```

- It displays the result depicted in Fig. 4.13 (a).
- When all the attributes from a table to be selected then in place of writing all the attribute names, a symbol '*' (asterisk) can be placed.
- So the above query can be written in the following format.

i.e.

```
SELECT * FROM Loan;
```

- It will display the same result like Fig. 4.13 (a).

- The **Select Clause** can also contain an arithmetic expression which includes the following operators in it, such as,

Addition	+
Subtraction	-
Mod (to find remainder)	%
Division	/
Multiplication	*
Raise to and Constants.	^

- Let's write some more queries using these operators.

Example 3: Find all the attributes in Loan relation where interest on the amount is calculated as 3% of the amount.

Query:

```
SELECT loan_no, branch_name, amount, amount * 0.03
FROM Loan;
```

- As we know, the table has only three attributes, but we want to have result of amount * 0.03.
- As this is the arithmetic expression, no attribute name is available for it. So in Oracle, it provides a name amount*0.03.
- The result of this query is shown in Fig. 4.15.

loan_no	branch_name	amount	amount*0.03
1	Jhons Street	200000	6000
2	Jhons Street	500000	15000
3	Adams Street	680000	20400
4	Bezant Street	520000	15600
5	Willam Street	250000	7500
6	Adams Street	100000	3000
7	Adams Street	100000	3000

Fig. 4.15: Result of Select with 3% of Amount

4.7.2 Where Clause

- It corresponds to select operation. This clause is used to define a predicate.
- While specifying predicate, SQL uses **logical connectives** AND, OR and NOT. SQL allows us to use the comparison operator to compare strings and arithmetic expressions as well as data types.
- The available comparison operators are <, < =, >, > =, =, < >.
- Let's write queries for where clause.

Example 4: Find all the loan numbers for loan made at 'Adams Street' branch with loan amount greater than 300000.

Query:

```
SELECT loan_no FROM Loan
WHERE   branch_name = 'Adams Street' AND amount > 300000;
```

- It will select all loan numbers first from loan table and then apply predicate for branch_name and amount.
- The result of the query is as shown in Fig. 4.16.

loan_no
3

Fig. 4.16: Result of Where Clause

Example 5: Find the loan number of those loans with loan amount between ₹ 200000 to 550000.

Query:

```
SELECT Loan_no FROM Loan
WHERE amount ≥ 200000 and amount ≤ 550000;
```

- The result is as shown in Fig. 4.17.

loan_no
1
2
4
5

Fig. 4.17: Result of Range Checking

(a) Between Operator: The range can be specified by relational operator. If we do not want to use these operators, the range can be specified by a keyword **Between**. This operator allows to select range. So the query (5) can be rewritten as follows:

Query: `SELECT loan_no FROM loan`
WHERE amount Between 200000 and 550000;

4.7.3 From Clause

- From clause itself defines a Cartesian product of a relation.
- This clause is also used to find natural join of more than one table.

Example 6: Find all customers who have a loan from the bank, find their names and loan_no.

Query:

```
Π cust_name, loan_no (Borrower × Loan)
SELECT cust_name, loan·loan_no, borrower·loan_no
FROM Loan, Borrower;
```

- The result will have Cartesian product of borrower and loan as shown in Fig. 4.18.

Cust_name	Loan·loan_no	Borrower·loan_no
Jill	1	1
Mac	3	1
Alex	5	1
Anna	2	1
Jill	6	1
Anna	7	1
Jill	1	2
Mac	3	2

Cust_name	Loan-loan_no	Borrower-loan_no
Alex	5	2
Anna	2	2
Jill	6	2
Anna	7	2
Jill	1	3
Mac	3	3
Alex	5	3
Anna	2	3
Jill	6	3
Anna	7	3
Jill	1	4
Mac	3	4
Alex	5	4
Anna	2	4
Jill	6	4
Anna	7	4
Jill	1	5
Mac	3	5
Alex	5	5
Anna	2	5
Jill	6	5
Anna	7	5
Jill	1	6
Mac	3	6
Alex	5	6
Anna	2	6
Jill	6	6
Anna	7	6
Jill	1	7
Mac	3	7
Alex	5	7
Anna	2	7
Jill	6	7
Anna	7	7

Fig. 4.18: Result of Borrower x Loan

- For finding natural join, we have to provide the comparison using WHERE clause. Now query 6 will be as follows:

```
SELECT cust_name, loan_no
FROM Loan, Borrower
WHERE Loan·loan_no = Borrower·loan_no;
```

- This will find cartesian product first and then natural join. The result of natural join is as shown in Fig. 4.19.

cust_name	loan_no
Jill	1
Anna	2
Mac	3
Alex	5
Jill	6
Anna	7

Fig. 4.19: Result of Natural Join

Example 7: Find names and loan numbers of all customers who have loan at 'Adams Street' branch.

Query:

```
SELECT cust_name, Loan·loan_no
FROM Loan, Borrower
WHERE Loan·loan_no = Borrower·loan_no
AND branch_name = 'Adams Street';
```

- The result is depicted in the Fig. 4.20.

cust_name	loan_no
Mac	3
Jill	6
Anna	7

Fig. 4.20: Result of Natural Join where branch-name = 'Adams Street'

4.7.4 Rename Operation

- SQL provides a mechanism for renaming both the relations and attributes. For this keyword **as** is used. It can be appear in both SELECT and FROM clause.
- Let's consider query of above section.

Example 8: Suppose we want to change name of loan·loan_no to loan_id, the query is rewritten as follows:

Query:

```
SELECT cust_name, Loan·loan_no as loan_id
  FROM Loan, Borrower
```

```
WHERE Loan·loan_no = Borrower·loan_no
        and branch_name = "Adams Street";
```

- The result of this query is similar to Fig. 4.20 except the change in the name of attribute. This is depicted in Fig. 4.21.

cust_name	loan_id
Mac	3
Jill	6
Anna	7

Fig. 4.21: Result of Natural Join with AS Keyword

- Similarly, we can change the relation name.

Example 9: Find names of all branches that have assets greater than atleast one branch located in London.

Query:

```
SELECT distinct T·branch_name
FROM branch T, branch S
WHERE T·assets > S·assets and S·branch_city = 'London';
```

- Here, S and T are called as **Tuple variables** which are used to compare two records (tuples) in the same relation.
- The tuple variables are defined in the From clause.
- In 9th query we have used two tuple variables to compare the assets and attribute.
- The result of this query is as depicted in Fig. 4.22.

branch_name
Adams Street
Willam Street
Bezant Street

Fig. 4.22: Result of T·assets > S·assets and S·branch_city = 'London'

- If we do a single change in the query (9), the result is as shown in Fig. 4.23.

```
SELECT distinct T·branch_name
FROM branch T, branch S
WHERE T·assets > S·assets
    and T·branch_city = 'London';
```

branch_name
Adams Street
Bezant Street

Fig. 4.23: Result of T·branch_city = 'London'

4.7.5 String Operation

- Database consists of some string values. The most commonly used operation on string is pattern matching. The operator used for pattern matching is **Like.**
- The patterns are case sensitive, i.e. uppercase characters do not match lowercase characters or vice versa.
- With the keyword **like**, we can use two characters:

(1) Percent (%):

It matches with any substring. It means that % stands for more number of characters.

(2) Underscore (_):

- It matches with any single character. It means _ stands for one character only.
- Consider following examples.
  - (a) "mad%": It matches with any string which begins with "**mad**". It may have characters after "mad".

    For example: madagascar, madake, madam, madhura etc.
  - (b) "%mi%": It matches any string containing **"mi"** in it. The result may have characters before and after it.

    For example: millar, milly, smith, etc.
  - (c) "---": matches with any string which has exactly 3 characters in it.

    For example: jam, sam, mat etc.
  - (d) "_ a _": It matches with any string which has exactly 3 characters and the middle character is "a".

    For example: cat, bat, mat etc.
- Let's write queries for string operation.

Example 10: Select the row from branch table whose branch name starts with letter "B".

Query:
```
SELECT * FROM Branch
WHERE branch_name like 'B%';
```

- The result of this query is shown in Fig. 4.24.

branch_name	branch_city	assets
Bezant Street	London	500000

Fig. 4.24: Result of like query

Example 11: Select tuples from customer table whose name of customer has a character 'l' at third position.

Query:
```
SELECT *
FROM Customer
WHERE cust_name like "_ _ l%"
```

- The result is depicted in Fig. 4.25.

cust_name	cust_city	cust_street
Jill	London	Jhons Street

Fig. 4.25: Result of like query

Example 12: Find names of all customers whose street address includes the substring 'Adams'.

Query:
```
SELECT cust_name
FROM Customer
WHERE cust_street like '%Adams%';
```

- The result is shown in Fig. 4.26.

cust_name
Mac
Van

Fig. 4.26: Result of like query

- The SQL allows the specification of an **escape character.**
- The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated as a normal character.
- We can define the escape character for a like comparison using the escape keyword.
- Let's consider backslash "\" as escape character in the following pattern.
- For example:

(1) Like "ab\%cd%" Escape "\"

It matches all strings beginning with "ab%cd" i.e. ab%cdy, ab%cdd etc.

(2) Like "ab\\cd%" Escape "\"

It matches all strings beginning with "ab\cd".

4.7.6 Order by Clause – Ordering of Tuples (W-16, S-18)

- The SQL provides a facility of controlling the order of tuple in the relation. The order by clause causes the tuple in the result of a query to appear in sorted order i.e. it may be in ascending or descending order.
- For ascending order the keyword **asc** is used and for descending order the keyword **desc** is used.

Example 13: List all the customer in alphabetical order who have loan at Adams Street branch.

Query:
```
SELECT cust_name
FROM loan, borrower
WHERE loan·loan_no = borrower·loan_no
AND branch_name = 'Adams Street'
ORDER BY cust_name asc;
```

- This query finds the cartesian product of Loan and Borrower.
- Then finds the equijoin by selecting similar loan_no from the result and then search for branch name as "Adams Street".
- After that it place the result in ascending order. The result of this query is as shown below in Fig. 4.27.

cust_name
Anna
Jill
Mac

Fig. 4.27: Result of order by asc

- By default SQL lists items in ascending order. The ordering can be performed on multiple attributes.
- There is no restriction given on the sorted order. If we want, we can use ascending for one attribute and descending for other.
- This is shown into the following query.

Example 14: List branch_name, city and assets in alphabetical order.

Query:
```
SELECT * FROM branch
    ORDER BY branch_city asc, branch_name desc;
```

The result is shown in Fig. 4.28.

branch_name	branch_city	assets
Jhons Street	London	100000
Bezant Street	London	500000
Adams Street	London	200000
Willam Street	New York	120000

Fig. 4.28: Result of Query 14

- Here, branch_name is in descending order whereas branch_city is in ascending order. However, branch_city comes first in the query and so although Adam Street should not be the first record, since London, New York are in ascending order.

4.7.7 Set Operations

- The SQL provides some set operations like Union, Intersection and Except (Minus) which are similar to the relational-algebra operations $\cup$, $\cap$ and $-$.
- The relations participating in the relational algebra and SQL operations must be compatible i.e. they must have same set of attributes.

- Consider the following example:

Example 15: Find set of all customers who have account at the bank.

```
SELECT cust_name
FROM depositor;
```

It displays the result as shown in Fig. 4.29.

cust_name
Smith
Jill
Zara
Smith
Alex

Fig. 4.29: cust_name from Depositor table

Example 16: Find set of customers who have a loan at the bank.

Query:
```
SELECT cust_name
FROM borrower;
```

The result is as shown in Fig. 4.30.

cust_name
Jill
Mac
Alex
Anna
Jill
Anna

Fig. 4.30: cust_name from Borrower table

- Now, let's perform the set operations on these two resultant tables.

(a) UNION Operation: UNION is one of the relational algebra operation which automatically eliminate duplicate values. It works just like OR operation.

Example 17: Find all customers having loan, an account or both at the bank.

Query:
```
(SELECT cust_name
FROM borrower)
UNION
(SELECT cust_name
FROM depositor);
```

- The Union operation automatically eliminates duplicate values and the order is also maintained alphabetically. The result of above query is shown in Fig. 4.31.

cust_name
Alex
Anna
Jill
Mac
Smith
Zara

Fig. 4.31: Result of Union

- If we want to retain all duplicate values then we should use a keyword **UNION ALL.** Let's write query 17 again with this keyword and see the result.

Query:
```
(SELECT cust_name
FROM borrower)
UNION ALL
(Select cust_name
FROM depositor);
```

- The result is as shown in Fig. 4.32.

cust_name
Jill
Mac
Alex
Anna
Jill
Anna
Smith
Jill
Zara
Smith
Alex

Fig. 4.32: Result of UNION ALL

- Here, all the names are selected from Borrower and then from Depositor.
- Hence, the union clause merges the output of two or more queries into a single set of rows and columns. Fig. 4.33 shows the output of union clause.

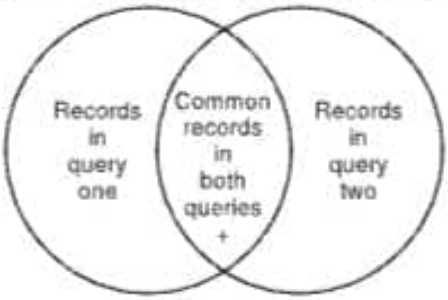


Fig. 4.33: Output of UNION clause

- Therefore, the final output of the union clause will be:

  Output = Record only in query one + records only in query two + a single set of record which is common in both queries.

- For the use of union clause there are some restrictions:
  - Number of columns in all the queries should be same.
  - Data types of the column in each query must be the same.
  - Union cannot be used in nested queries.

(b) Intersection operation: This will find the common elements from both the relations. This operation automatically eliminates duplicate values. But if we want all duplicate values, then a keyword **intersect all** is used. The intersect operation is shown in Fig. 4.34.

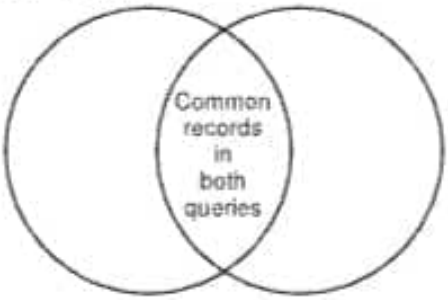


Fig. 4.34: Output of Intersection clause

Example 18: Find all customers who have both Loan and Account at the bank.

Query:

```
(SELECT cust_name
FROM borrower)
INTERSECT ALL
(SELECT cust_name
FROM depositor);
```

(c) The Except Operation (Minus Operation): It is actually a set difference operator used to find difference between two relations. The output of minus operation will be records only in query one. This is shown in Fig. 4.35.

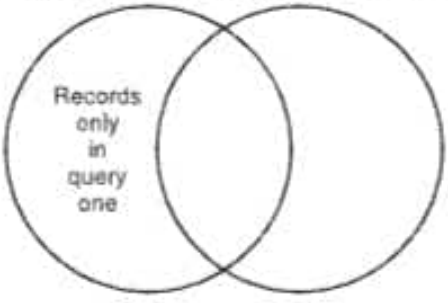


Fig. 4.35: Output of Minus (except) operation

Example 19: Find all customers who have an account but no loan at the bank.

Query:

```
(SELECT distinct cust_name
FROM Depositor)
EXCEPT
(SELECT cust_name
FROM Borrower);
```

- The Except operation automatically eliminates the duplicate. If we want to retain duplicate write **Except all.**
- And that's why, for intersection and difference two keywords introduced called EXISTS and NOT EXISTS.
- The intersection of two sets contains all elements that belong to both of the original sets.
- The difference contains elements that belong only to the first of the two sets.

Example 20: For intersection, we again write the same query i.e. Find all customers who have both loan and amount at the bank.

Query:

```
SELECT DISTINCT cust_name
FROM borrower
WHERE EXISTS
(SELECT cust_name
  FROM depositor);
```

- The result is depicted in Fig. 4.36.

cust_name
Mac
Alex
Anna
Jill

Fig. 4.36: Result of Intersection using EXISTS

Example 21: For difference let's write the same query i.e. Find all customer who have an account but no loan at the bank.

Query:

```
SELECT DISTINCT cust_name
FROM borrower
WHERE NOT EXISTS
(SELECT cust_name
FROM depositor);
```

- Here, it would not display any customer name because in difference the record which is present in first table but not in second will be displayed.
- According to our database this situation is not present.
- If we change the query again.

Example 22: Find all customers who have loan but no account at the bank.

Query:

```
SELECT DISTINCT cust_name
FROM depositor
WHERE NOT EXISTS
(SELECT cust_name
FROM borrower);
```

- The result is shown in Fig. 4.37.

cust_name
Zara
Smith

Fig. 4.37: Result of difference using NOT EXISTS

4.8 NESTED QUERIES

- A **query** within a query is called **nested query**. Sometimes it is referred to as nested subquery.
- A **subquery** is a select-from-where expression that is nested within another query.
- The nested queries can be written for the following:
  1. Set Membership.
  2. Set Comparison.
  3. Set Cardinality.

4.8.1 Set Membership

- Set is a collection of values produced by a select clause.
- Set membership is the operation which allows testing the tuples for membership into the relation.
- There are two keywords used for set membership viz. **IN** and **NOT IN.**
- The **IN** is the connective test for set membership whereas **NOT IN** is the connective test for absence of set membership. **IN** is similar to intersection and **NOT IN** is similar to difference operation.

Example 23: Find all customers who have both a loan and an account at bank.

Query:

```
SELECT distinct cust_name
FROM borrower
```

```
WHERE cust_name in
    (SELECT cust_name
    FROM depositor);
```

- The result is as shown in Fig. 4.36.

Example 24: Find all customers who have loan at bank but don't have account at the bank.

Query:

```
SELECT distinct cust_name
FROM borrower
WHERE cust_name NOT IN
    (SELECT cust_name
    FROM depositor);
```

- The result is as shown in Fig. 4.37.

4.8.2 SET Comparison

- For comparison of sets we have already written Query (9) and we got result which has been already depicted in Fig. 4.22.
- The alternate way of writing same query is by using keyword **SOME**.
- SQL allows < some, <= some, >= some, = some and < > some. Here, again = some is identical to **IN** and < > some is identical to **NOT IN**. Lets write query for this.

Example 25: Find name of all branches that have assets greater than atleast one branch located in London.

Query:

```
SELECT branch_name
FROM branch
WHERE assets > some
    (SELECT assets
    FROM branch
    where branch_city = "London");
```

- The result is as shown in Fig. 4.22.
- The > **some** comparison in the where clause of the outer select is **true** if the assets value of the tuple is greater than atleast one member of the set of all asset values for branch_city London.
- Sometimes, in place of **some** the keyword **any** is synonymously used.
- In the above query whenever, a single value appears which is greater DBMS displays the result.
- It never checks for other values. If we want that DBMS should check for all values greater than assets then a keyword **ALL** is used.

- SQL allows < all, <= all, >= all = all, and < > all comparisons. < > all is identical to **NOT IN.**

  Example 26: Find names of all branches that have an assets value greater than that of each branch in London.

  Query:

```
SELECT branch_name
FROM branch
WHERE assets > = all
    (SELECT assets
    FROM branch
    WHERE branch_city = "London");
```

- The result is as shown in Fig. 4.38.

branch_name
Bezant Street

Fig. 4.38: Result of set comparison using ALL

4.8.3 Set Cardinality

- **Cardinality** means the number of tuples available in a database. For example, the cardinality of relation depositor from Fig. 4.13 (f) is five.
- In Set Cardinality, we can test for **empty relations** or test for **absence** of **duplicate values.**
- The test for empty relations can be checked by keyword **exists** or **not exists** which we have already seen in this chapter.
- For the test of absence of **duplicate tuples**, the keyword **unique** is used.
- This keyword returns value true if the subquery consists of no duplicate values or tuples.
- If the subquery has any duplicate value, let's see one example.

  Example 27: Find all customers who have at most one account at the Bezant Street branch.

  Query:

```
SELECT T.cust_name
FROM depositor as T
WHERE unique
    (SELECT R.cust_name
    FROM account, depositor as R
WHERE R.acc_no = account.acc_no
AND account.branch_name = 'Bezant Street');
```

- The result is as shown in Fig. 4.39.

cust_name
Jill

Fig. 4.39: Result of unique command

- This query will find the cust_name from the table depositor and then search for a unique record in the account table by checking specified condition.
- Similarly, we can check for existence of duplicate tuples just by using keyword **not unique**.
- Formally, the unique test on a relation is defined to fail if the relation contains two tuples t1 and t2 such that t1 = t2. If t1 or t2 is null, then the result of unique is true even if there are multiple copies of a tuple as long as one of the attributes of a tuple is null.

4.9 AGGREGATE FUNCTIONS

- SQL offers the ability to compute functions on groups of tuples using the **group by** clause.
- The attribute or attributes given in the **group by** clause are used to form groups.
- Tuples with the same value on all attributes in the group by clause are placed in one group.
- These functions take a collection of values as inputs and returns a single value. SQL offers 5 built-in aggregate functions:
  1. Average: avg
  2. Minimum: min
  3. Maximum: max
  4. Total: sum
  5. Count: count
- The input to **sum** and **avg** must be a collection of numbers but the other operation can operate on collection of non-numeric datatypes such as string.

  Example 28: Find average amount balance of Adams Street branch.

  Query:
```
SELECT Avg(balance)
    FROM Account
    WHERE branch_name = 'Adams Street';
```
- All the balance value of Adams Street are added and divided by the total number of values for Adams Street. The result is shown in Fig. 4.40.

Avg (balance)
10000

Fig. 4.40: Result of average

- Sometimes, we have to apply aggregate function not only to a single tuple but for set of tuples. This is specified using **Group by** clause.
- The group by clause is used to form a group for same attributes and same values.
- Let's write query for it.

Example 29: Find average account balance of each branch.

Query:
```
SELECT branch_name, Avg(balance)
FROM Account
Group by branch_name;
```

- The result is shown in Fig. 4.41.

branch_name	Avg (balance)
Adams Street	10000
Bezant Street	6000
Willam Street	20000

Fig. 4.41: Result of group by clause

- The retention of duplicates is important in computing an average.
- If we eliminate duplicates, then average may be wrong.
- If we want to eliminate duplicates then use **distinct** as keyword, but it is used only prior to computing an aggregate function.

Example 30: Find the number of depositors for each branch.

Query:
```
SELECT branch_name, count(distinct cust_name)
FROM Depositor
GROUP BY branch_name;
```

- At times, it is useful to state a condition that applies to groups rather than to tuples.
- For example: we might be interested only in branches where the average account balance is more than Rs. 1000. This condition does not apply to a single tuple. Rather, it applies to each group constructed by **group by** clause.
- To express such a query, we use the **having clause** of SQL. Predicates in the **having** clauses are applied after the formation of groups, so aggregate functions may be used.

Having Clause: (S-17)

- Sometimes, it is essential to apply or to state the conditions with group by when a condition is applied to a group rather than tuples **having** clause is used.

Example 31: List the branches, where the average account balance is more than 6200/-

Sol.:
```
SELECT branch_name, Avg(balance)
FROM Account
GROUP BY branch_name
HAVING Avg(balance) > 6200;
```

- The result is shown in Fig. 4.42.

branch_name	Avg (balance)
Adams Street	10000
Bezant Street	20000

Fig. 4.42: Result of average with having clause

Example 32: Find those branches with the highest average balance.

- Since, aggregate functions cannot be composed in SQL, we cannot use max [avg (_ _ _)].
- Instead we find those branches for which the average balance is greater than or equal to all average balances.

Query:
```
SELECT branch_name from Depositor
GROUP BY branch_name
HAVING Avg(balance) ≥ all
  (SELECT Avg(balance) from Depositor
     GROUP BY branch_name);
```

Example 33: Find the average balance in all accounts.

- Here, we wish to treat the entire relation as a single group, therefore, we do not use a group by clause.

Query:
```
SELECT Avg(balance)
FROM Account;
```

- Let's write queries on different functions.

Example 34: Find minimum and maximum account balance of each branch.

Query:
```
SELECT branch_name, min(balance)
FROM Account
GROUP BY branch_name;
```

- The result of this query is depicted in Fig. 4.43.

branch_name	min (balance)
Adams Street	5000
Willam Street	2000
Bezant Street	20000

Fig. 4.43: Result

Query:
```
SELECT branch_name, max(balance)
FROM account
GROUP BY branch_name;
```

- The result is as shown in Fig. 4.44.

branch_name	max (balance)
Adams Street	15000
Willam Street	10000
Bezant Street	20000

Fig. 4.44: Result

Note: If a **where** clause and a **having** clause appears in the same query, the predicate in the where clause is applied first. Tuples satisfying the **where** predicate are then placed into groups by **Group by** clause.

- The having clause is then applied to each group.
- The group that satisfy the having clause predicate are used by the select clause to generate tuples of the result of the query.
- If there is no having clause, the entire set of tuples satisfying the where clause is treated as a single group.

Example 35: Find the average balance of all depositors who live in London and have atleast 3 accounts.

Query:

```
SELECT avg(balance)
FROM Depositor, Customer, Account
WHERE Depositor.cust_name = Customer.cust_name
AND Depositor.acc_no = Account.acc_no
AND cust_city = 'London'
GROUP BY Depositor.cust_name
HAVING count(distinct acc_no) ≥ 3;
```

SOLVED CASE-STUDIES

(**Note:** Following examples are solved using Oracle)

1. **Consider the following entities and relationships.**

   Owner (Licence_no, name, address, phone)

   Car (car_no, model, colour)

   Owner and Car are related with one-to-many relationships.

   Create a RDB for the above and solve the following queries in SQL.

   Sol.: Owner and Car are the two relations given. Let's draw an E-R diagram with the given relationships. The E-R diagram is shown in Fig. 4.45.

   After drawing E-R diagram, we have to convert it into the form of table with the given relationship.

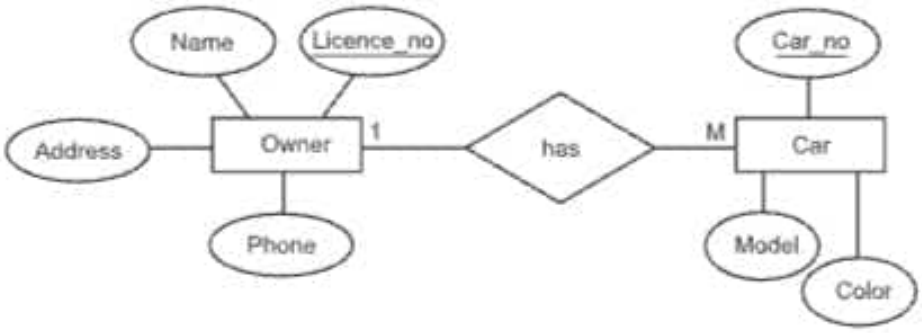


Fig. 4.45: E-R diagram

The relationship is One-to-Many, the primary key of relation one (called parent) is taken and added as a foreign key into the Many (called child) relation. Let's draw on RDB which is shown in Fig. 4.46.

Fig. 4.46: One-to-Many Relationship

After this we should add some tuples into the table which will answer the written queries.

Query 1: Find the name of owner of 'Zen' and 'Indica' cars.

```
SELECT name, address, model
FROM Car c, Owner o
WHERE (model='Zen' Or model='Indica') And c.car_no=o.car_no;
```

Query 2: Insert a record in a car relation.

```
INSERT INTO car
VALUES (154, 'Maruti', 'BLACK');
```

Query 3: List all the models of owner 'Mr. Shon' having colour 'blue'.

```
SELECT model, colour
FROM Car c, Owner o
WHERE name='Mr. Shon' and colour='blue' and c.car_no=o.car_no;
```

Query 4: To list the information of all cars in 'London'.

```
SELECT name, address, model, colour
FROM Car c, Owner o
WHERE c.carno=o.carno and address='London'
GROUP BY name, address, model, colour;
```

2. Consider the following Entities and Relationships.

Machine (m_no, m_name, m_type, m_cost)

Part (p_no, p_name, p_desc)

Machine and Part are related with One-to-Many relationships.

Create a RDB for the above and solve the following queries in MS ACCESS.

Solution: The One-to-Many relationship is as shown in Fig. 4.47.

Fig. 4.47: One-to-Many

Query 1: Increase the cost of machine by 5%.

```
UPDATE Machine SET m_cost = m_cost + (m_cost * 0.05);
```

Query 2: Delete all machines having particulars "wheel".

```
DELETE
FROM Machine
WHERE m_type="wheel";
```

Query 3: List all the machines whose cost > 1,00,000.

```
SELECT m_name, m_cost
FROM Machine
where m_cost>100000;
```

---

3. Consider the following Entities and Relationships.

Customer (cust_no, name, address)

Quotation (quot_no, desc, amt_quoted)

Customers and quotations are related with One-to-Many relationships.

Create a RDB for the above and solve the following queries in SQL.

Sol.: The RDB is as shown in Fig. 4.48.

Fig. 4.48: One-to-Many

Query 1: List all the customers who are demanding for the quotation of "washing machine".

```
SELECT name, address
FROM Customer, Quotation
WHERE desc="washing machine"
      And customer.cust_no = quotation.cust_no;
```

Query 2: List all the customer who are demanding for the quotation of "fridge".

```
SELECT name, address
FROM Customer, Quotation
WHERE desc="Fridge" And customer.cust_no = quotation.cust_no;
```

Query 3: Delete all the customer with details with address 'London'.

```
DELETE
FROM Customer
WHERE address="London";
```

Query 4: To print customerwise list of quotation but only those quotations whose amt > 5000.

```
SELECT name, customer.address, desc, amt_quoted
FROM Customer, Quotation
WHERE amt_quoted>5000 And customer.cust_no = quotation.cust_no;
```

4. Consider the following Entities and Relationships.

Company (c_product, c_name, region, state)

Branches (b_product, city)

Company and Branches are related with one-to-many relationships.

Create a RDB for the above and solve the following queries in SQL.

Sol.: Here we cannot consider c_product and b_product as primary keys because product name can be duplicate. So lets add one more field called c_id into relation company which will save as primary key. The one-to-many relationship is as shown in Fig. 4.49.

Fig. 4.49: One-to-Many

Let's write queries:

Query 1: List all the cities having branch product 'H_2SO_4' and 'SO_3'

```
SELECT city, b_product
FROM Branches b, Company c
WHERE b.c_id=c.c_id and (b_product='H2SO4' or b_product='SO3');
```

Query 2: List all the states whose branch product is 'oleum'.

```
SELECT state
FROM Branches b, Company c
WHERE b.c_id=c.c_id and (b_product='oleum');
```

Query 3: Delete all the company details of city 'New York'.

```
DELETE
FROM Branches
WHERE city='New York';
```

Query 4: Print city wise branches in sorted order.

```
SELECT c_name as company, city
FROM Branches b, Company c
WHERE b.c_id=c.c_id
ORDER BY c_name;
```

5. Consider the following Entities and Relationships. **(W-16)**

Teacher (t_no, t_name, college_name, dept)

E_Test(e_no, test_name)

Teacher and E_test are related with Many-to-Many relationships.

Create a RDB for the above and solve the following queries in SQL.

Sol.: The relationship is Many-to-Many. So, we have to create a new table with the primary keys. The RDB is as shown in Fig. 4.50.

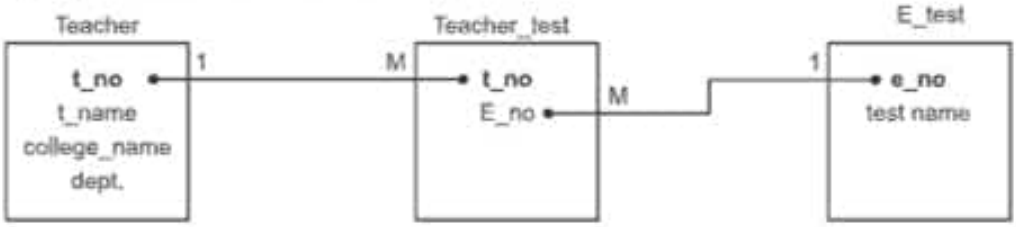


Fig. 4.50: Many-to-Many Relationship

Let's write the queries.

Query 1: Give the name of teachers who have passed either "SET" or "NET".

```
SELECT t_name, college_name, test_name, dept
FROM Teacher t, E_test e, Teacher_test r
WHERE r.t_no=t.t_no and r.e_no=e.e_no and
                              (test_name='SET' or test_name='NET');
```

Query 2: Delete all the teachers details of 'Physics' dept.

```
DELETE
FROM Teacher
WHERE dept='Physics';
```

Query 3: To list exam wise list of teachers who have passed the respective exams.

```
SELECT test_name, t_name, college_name, dept
FROM Teacher t, E_test e, Teacher_test r
WHERE r.t_no=t.t_no and r.e_no=e.e_no
GROUP BY test_name, t_name, college_name, dept;
```

Query 4: To print the total number of teachers passing the respective exam.

```
SELECT test_name, count(*) AS No_of_teacher
FROM Teacher t, E_test e, Teacher_test r
WHERE r.t_no=t.t_no and r.e_no=e.e_no
GROUP BY test_name;
```

6. Consider the following Entities and Relationships.

Country (con_code, name, capital)

Population (Pop_code, population)

Country and Population are related with One-to-One relationships.

Create a RDB for the above and solve the following queries in SQL.

Solution: The relationship given is one-to-one. There is no need to create a new table or take primary key and add it to other table. Here, primary keys of both tables are directly related with each other. This is shown in Fig. 4.51.

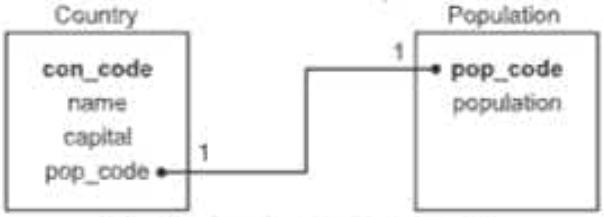


Fig. 4.51: One-to-One Relationship

Query 1: Give the name and population of country whose capital is 'Tokyo'.

```
SELECT name, population
FROM Population p, Country c
WHERE P.pop_code=c.pop_code and capital='Tokyo';
```

Query 2: Give the name of all the countries whose population is greater than 50,00,000.

```
SELECT name, capital, population
FROM Population p, Country c
WHERE P.pop_code=c.pop_code and population>5000000;
```

Query 3: To print countrywise population.

```
SELECT name, population
FROM Population p, Country c
WHERE p.pop_code=c.pop_code
GROUP BY name, population;
```

7. Consider the following Entities and Relationships.

Wholesaler (w_no, w_name, addr, city)

Product (p_no, p_name)

Wholesaler and Product are related with Many-to-Many relationships.

Create a RDB for the above and solve the following queries in SQL.

Solution: Many-to-Many relationship allows us to create new table. The RDB is as shown in Fig. 4.52.

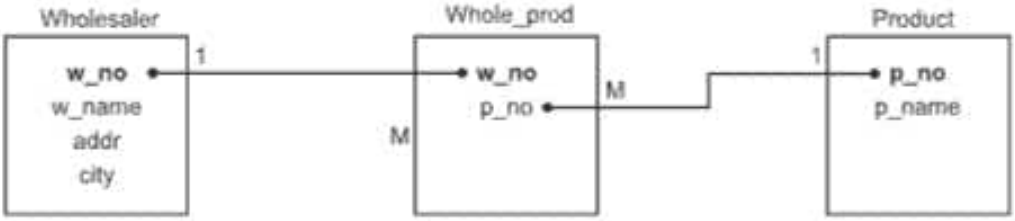


Fig. 4.52: Many-to-Many

Query 1: List all the wholesaler of product "books".

```
SELECT w_name, city
FROM Wholesaler w, Product p, Whole_prod r
WHERE w.w_no=r.w_no and p.p_no=r.p_no and p_name= "books";
```

Query 2: Count the no of wholesaler in the city "London".

```
SELECT COUNT(*) as wholesaler_from_London
FROM Wholesaler
WHERE city="London";
```

Query 3: Insert a record in wholesaler relation.

```
INSERT INTO Wholesaler
VALUES (111, "Alex", "123, abc Road", "London");
```

Query 4: To print wholesaler wise product.

```
SELECT DISTINCT w_name, city, p_name
FROM Wholesaler w, Product p, Whole_prod r
WHERE w.w_no=r.w_no and p.p_no=r.p_no;
```

8. Consider the following Entities and Relationships.

Politician (p_no, p_name, p_desc, constituency)

Party (partycode, partyname)

Politician and party are related with Many-to-One relationships. Create a RDB for the above and solve the following queries in SQL.

Sol.: The relationship is Many-to-One. Therefore, the primary key of one relation is taken as foreign key in many tables. The relationship is shown in Fig. 4.53.

Fig. 4.53: Many-to-One

Query 1: List all the politicians of the party 'BJP'.

```
SELECT Politician.p_name
FROM Party, Politician
WHERE party_name='BJP'And politician.party_code=party.party_code;
```

Query 2: Count the no of politicians having political description as a 'member of parliament'.

```
SELECT count(p_name) AS Total_MPs
FROM Politician
WHERE p_desc='MEMBER OF PARLIAMENT';
```

Query 3: Give the party name of politician 'mulayam singh yadav'.

```
SELECT Party_name
FROM Party, Politician
WHERE Politician.name='Mulayam Singh Yadav'
                        And Politician.party_code=Party.Party_code;
```

Query 4: List party wise list of politicians of 'London' constituency.

```
SELECT politician.name, party.party_name, politician.constituency
FROM Party, Politician
WHERE politician.constituency='London'
                        And party.party_code=politician.party_code;
```

Query 5: Print the total no. of politician in each party for 'London' constituency.

```
SELECT count(name) no_of_politician, party_name
FROM Party, Politician
WHERE party.party_code = politician.party_code
                                And politician.Constituency='London'
GROUP BY party.party_name;
```

9. Consider the following Entities and Relationships.

Game (g_no, gname, no_of_players, coach_name, captain)

Player (p_no, p_name)

Game and Players are related with Many-to-Many relationships.

Create a RDB for the above and solve the following queries in SQL.

Solution: The relationship is many-to-many so create a new table which is shown in Fig. 4.54.

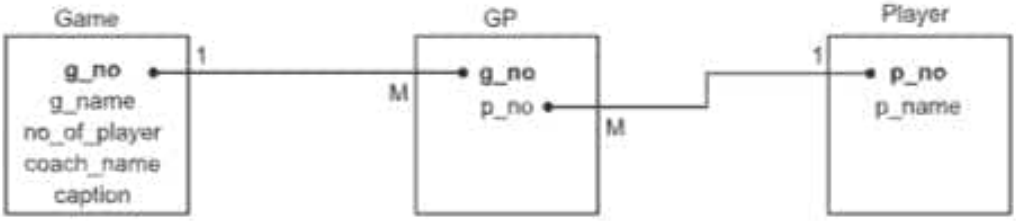


Fig. 4.54: Many-to-Many

Query 1: List the name of players playing 'basketball' and 'handball'.

```
SELECT p_name, g_name
FROM Game g, Player p, GP r
WHERE p.p_no=r.p_no and g.g_no=r.g_no and
                        (g_name='basket' or g_name='handball');
```

Query 2: List the name of players playing game 'cricket'.

```
SELECT p_name
FROM Game g, Player p, GP r
WHERE p.p_no=r.p_no and g.g_no=r.g_no and (g_name= 'cricket');
```

Query 3: Count the total no. of players whose coach name is 'mr. Anderson'

```
SELECT COUNT(*) as no_of_player
FROM Game g, Player p, GP r
WHERE p.p_no=r,p_no and g.g_no=r.g_no and coach_name='mr Anderson;
```

Query 4: List the game wise player list.

```
SELECT g_name as game, p_name as player
FROM Game g, Player p, GP r
WHERE p.p_no=r.p_no and g.g_no=r.g_no
GROUP BY g_name, p_name;
```

10. Consider the following Entities and Relationships. **(S-17)**

Item (i_no, i_name, i_qty)

PO (p_no, p_date)

Supplier (s_no, s_name, s_addr)

Item and PO are related with One-to-Many relationships along with descriptive cost and quantity. Supplier and PO are related with One-to-Many relationship.

Create a RDB for the above and solve the following queries in SQL.

Solution: The RDB (relational database) is shown in Fig. 4.55,

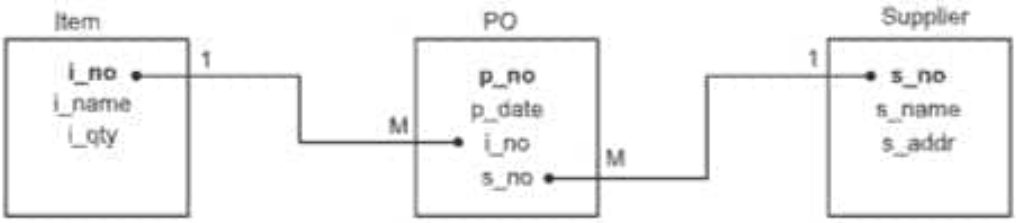


Fig. 4.55: One-to-Many

Query 1: List the name of supplier to whom po is given for "mouse".

```
SELECT s_name, i_qty
FROM PO, Item i, Supplier s
WHERE s.s_no=PO.s_no And PO.i_no-i.i_no And i_name="mouse";
```

Query 2: List the name of supplier and item_name in PO generated on "30-sept-2001".

```
SELECT s_name, i_name
FROM PO, Item i, Supplier s
WHERE s.s_no-PO.s_no And PO.i_no=i.i_no And to_date(p_date)=
                                                    '30-SEP-01'
```

Query 3: Find out po number, po date and supplier name of the po which is of maximum amount.

```
SELECT p_no, p_date, s_name
FROM PO, Supplier
WHERE PO.s_no=Supplier.s_no and cost in(select max(cost)from po);
```

Query 4: Display all the po which contains the number, date, supplier name of the po details of all items included in that i.e. name of item, qty and rate).

```
SELECT p_no, p_date, s_name, i_name, qty, cost
FROM PO, Supplier, Item
WHERE PO.s_no=Supplier.s_no And Item.i_no=PO.i_no;
```

---

11. Consider the following relational database

lives (person_name, street, city)

works (person_name, company_name, salary)

located_in (company_name, city)

manager (person_name, manager_name)

Write the SQL statements for each of the following.

Query 1: Find all employees who have more than the average salary of employee in their company.

Solution:

```
SELECT distinct W.person_name
FROM works W
WHERE W.salary >= (SELECT avg(t.salary)
   FROM works t
```

```
    WHERE t.company_name=w.company_name);
```

Query 2: Find the company with least number of employees.

Solution:

```
SELECT t.company_name
FROM works t
GROUP BY t.company_name
HAVING count(t.person_name) <= all
(SELECT count(S.person_name)
FROM works S
GROUP BY S.company_name);
```

12. Consider the following relational database.

Sailors (sid, sname, rate, age)

Boats (bid, bname, colour)

Reserves (sid, bid, day)

Write SQL statement for each of the following queries.

Sol.: From the given database, we can directly recognize that the relationship between sailors and boats is Many-to-Many i.e. a new table called Reserves is created by the primary keys of both tables. Therefore, the Relational database diagram is as shown in Fig. 4.56.

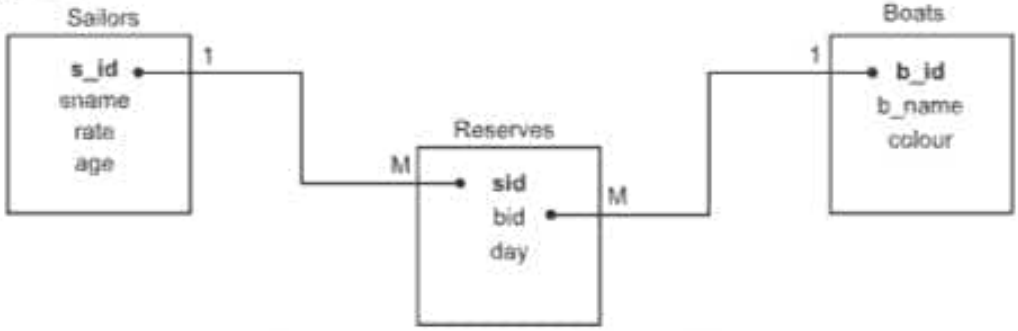


Fig. 4.56: Many-to-Many Sailors Database

Lets write queries on the above database.

Query 1: Find names and ages of all sailors.

```
SELECT DISTINCT sname, age FROM Sailors;
```

Query 2: Find all the sailors with a rating above 8.

```
SELECT *
FROM Sailors
where rate > 7;
```

Query 3: Find the names of sailors who have reserved boat no. 103.

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103;
```

Query 4: Find the Sids of sailors who have reserved a green boat.

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE B.bid=R.bid AND B.colour="green";
```

Query 5: Find colours of boats reserved by Alex.

```
SELECT B.colour
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.Bid=B.bid AND S.Sname="Alex";
```

Query 6: Find the names of sailors who have reserved atleast one boat.

```
SELECT S.Sname
FROM Sailors S, Reserver R
WHERE S.sid=R.sid;
```

Query 7: Find the ages of sailors whose name begins and ends with P.

```
SELECT S.age
FROM Sailors S
where S.sname like "P%P";
```

The name may be pop, P... P where ...means lot of characters considered in place of %.

Query 8: Find name of sailors who have reserved red and green boat.

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid
AND (B.colour="red" OR B.colour="green");
```

13. Consider the following relational database:

Doctor (d_no, d_name, address, city)

Hospital (h_no, name, street, city)

Many doctors are working into many hospitals. Doctors visits to hospital on certain date. Create a RDB and answer following queries in SQL.

Solution: Many to many relationships allows us to create a new table in which we will consider date as one of the attributes. The RDB is as shown in Fig. 4.57.

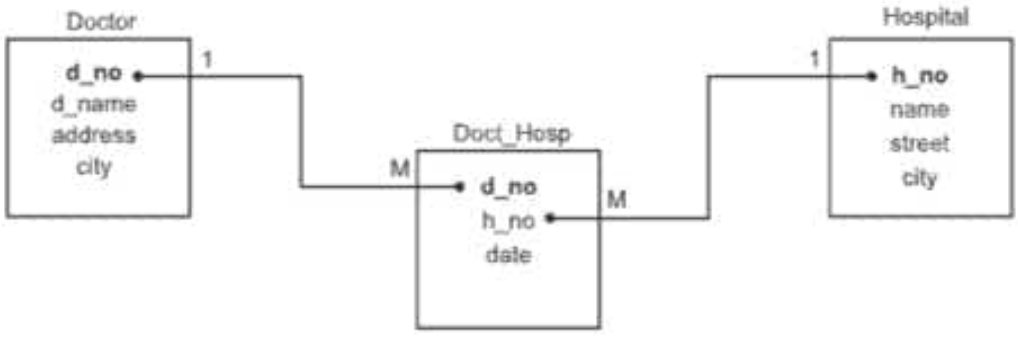


Fig. 4.57: Many-to-Many with Attribute Date

Write following queries into SQL.

(1) Count number of doctors visited to "Washigton Hospital" on 1st March 2019.

```
SELECT COUNT(distinct d_no)
FROM Hospital, Doct_hosp
WHERE Doct_hosp.H_no=Hospital.H_no
AND name = " Washigton Hospital"
AND TO_DATE(date) = "01-MAR-19";
```

(2) Find out how many times 'Dr. Smith' visited to Washigton Hospital.

```
SELECT COUNT(d_no)
FROM Doctor d, Hospital h,
   Doct_hosp m
WHERE D.D_dname = "Dr. Smith"
  AND H.name = " Washigton Hospital"
AND D.d_no = m.d_no
AND H.H_no = m.h_no;
```

(3) List the doctors in city London.

```
SELECT * FROM Doctor WHERE city = "London";
```

14. Consider the following database:

Employee

F_name	MINIT	L_name	SSN	B_date	Addr	Sex	Salary	D_No	Super SSN

Department

Dname	Dnumber	Mgr SSN	Mgr STartdate

Dept-Locations

D_number	D_location

Project

P_name	P_number	Plocation	Dnum

Works on

ESSN	P_NO	Hours

Dependent

ESSN	Dep_name	Sex	B_date	Relation

Fig. 4.58: Databases

Write the SQL for the following queries.

(i) Retrieve the birthdate and address of the employee whose name is "John B. Smith".

Sol.:
```
SELECT B_date, Addr
FROM Employee
WHERE F_name="John" AND MINIT="B" and L_name="Smith";
```

(ii) Retrieve the name and address of all employees who work for the research department.

Sol.:
```
SELECT F_name, L_name, Addr
FROM Employee, Department
WHERE Dname="Research" AND Dnumber=D_no;
```

(iii) For every project located in "London", list the project number and controlling dept-number and department manager's name.

Sol.:
```
SELECT P_number, Dnum, L_name
FROM Project, Department, Employee
WHERE Dnum=Dnumber and
Mgr_SSN=SSN and Plocation="London";
```

(iv) For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

Sol.:
```
SELECT E.F_name, E.L_name, S.F_name, S.L_name
FROM Employee E, Employee S
WHERE E.superSSN=S.SSN;
```

(v) Select all employee SSN's.

Sol.:
```
SELECT SSN FROM Employee;
```

(vi) Retrieve all the attribute values of employer tuples who work in dept-number 5.

Sol.:
```
SELECT * FROM Employee
WHERE D_NO=5;
```

(vii) Retrieve the salary of every employee.

Sol.:
```
SELECT salary FROM Employee;
```

If we are interested only in distinct salary values then we write,

```
SELECT distinct salary
  FROM Employee;
```

(viii) Find the list of all project numbers for projects that involve an employee whose last name is "Smith", either as a worker or as a manager of the department that controls the project.

Sol.:
```
SELECT P_number FROM Project, Department, Employee
WHERE Dnum=D_number and Mgr_SSN=SSN and L_name="Smith"
UNION
SELECT P_number FROM Project  works_on Employee
WHERE P_number=PNO and ESSN=SSN AND L_name="Smith";
```

(ix) Retrieve the names of employees whose salary is greater than the salary of all the employees in department 5.

Sol.:
```
SELECT L_name, Fname FROM Employee
 WHERE salary > all
    (SELECT salary from Employee
      WHERE D_NO=5);
```

(x) Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

Sol.:
```
SELECT E.F_name, E.name
FROM Employee E
WHERE E.SSN in
(SELECT ESSN
FROM Dependent
WHERE ESSN=E.SSN AND
E.Fname=Dep_name AND sex=E.sex);
```

OR

```
SELECT E-Fname, E.Lname
FROM employee E, Dependent D
WHERE E.SSN=D.ESSN AND
           E.sex=D.sex AND
           E.Fname=D.Dep_name;
```

(xi) Find the sum of salaries of all employees, the maximum, minimum and average salary.

Sol.:
```
SELECT sum(salary), max(salary), min(salary), avg(salary)
 FROM Employee
```

(xii) Retrieve the dependent number, the number of employees in the department and their average salary.

Sol.:
```
SELECT DNO, count(*), avg(salary)
 FROM Employee
 GROUP BY D_NO;
```

(xiii) Retrieve the project number, the project name and the number of employees who work on that project.

Sol.:
```
SELECT P_number, P_name, count(*)
 FROM Project, works_on
 WHERE P_number = P_NO
 GROUP BY P_number, P_name;
```

(xiv) For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

Sol.:
```
SELECT P_number, p_name, count(*)
    FROM Project, works_on, Employee
    WHERE P_number = P_NO
    GROUP BY P_number, P_name
    HAVING count(*) > 2;
```

(xv) Retrieve the project number, name and the number of employees from department 5 who work on the project.

Sol.:
```
SELECT P_number, P_name, count(*)
    FROM Project, works_on, Employee
    WHERE P_number = P_NO AND SSN = ESSN AND DNO = 5
    GROUP BY P_number, P_name;
```

(xvi) For each department having more than five employees, retrieve the department number and the number of employees making more than ₹50,000.

Sol.:
```
SELECT D_name, count(*)
    FROM Department, Employee
      WHERE Dnumber=D_No and salary > 50000 and
          D_NO in (SELECT D_NO FROM Employee GROUP BY D_NO
                HAVING count(*) > 5)
      GROUP BY Dname;
```

(xvii) Retrieve all employees whose address is in London.

Sol.:
```
SELECT F_name, L_name
    FROM Employee
    WHERE Addr like "%London%";
```

(xviii) Show the resulting salaries if every employee working on the "Product X" project is given a 10% raise.

Sol.:
```
SELECT F_name, L_name, 1.1 * salary
FROM Employee, Works_on, Project
WHERE SSN=ESSN and P_no=P_number AND P_name="Product X";
```

(xix) Retrieve a list of employees and the projects they are working on, ordered by department in descending order and within each department, ascending of name.

Sol.:
```
SELECT Dname, L_name, F_name, P_name
FROM Department, Employee, Works_on, Project
WHERE Dnumber=D_NO AND SSN=ESSN AND P_NO=P_NUMBER
ORDER BY Dname desc, L_name Asc;
```

Summary

- SQL is a standard Database language which is used to create, maintain and retrieve the relational database.
- Data Definition Language is used to define the database structure.
- DML statements are used for managing data within database.
- Data Control Language (DCL): Consists of commands which deal with the user permissions and controls of the database system.
- Transaction Control Language(TCL): Consist of commands which deal with the transaction of the database
- The SQL SELECT statement is used to fetch the data from a database table which returns this data in the form of a result table.
- Sub queries are nested queries that provide data to the enclosing query.
- An aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Check Your Understanding

1. What is the full form of SQL
   (a) Structured Query Language (b) Simple Query Language
   (c) Structured Query List (d) Simple Question Language
2. Which command in SQL is used to change table storage characteristics?
   (a) Alter Table (b) Drop Table
   (c) Change Table (d) All of the above
3. Which is the right statement to insert values to the employee table.
   (a) Insert Values (value1, value2,....)
   (b) Insert into Values (value1, value2,....)
   (c) Insert into Employee values(value1, value2,....)
   (d) Insert values into Employee (value1, value2,....)
4. Which is not a Set operation in SQL ?
   (a) Union (b) Intersection
   (c) Create (d) Minus
5. In SQL, the count function returns the number of .....
   (a) Groups (b) Columns
   (c) Tables (d) Values

Ans.: (1) a (2) a (3) c (4) c (5) d

Practice Questions

Q.1 Answer the following question in brief

1. What is a SQL?
2. What is generalized structure of SQL query?
3. Explain term query.
4. What aggregate operators does the SQL support?
5. Explain the following operations
   (i) String operations
   (ii) Predicates and join
   (iii) Set membership
   (iv) Set comparison
   (v) Aggregate functions

Q.2 Answer the following questions.

1. How tables are created and maintained by using SQL?
2. How database manipulation is carried out using SQL?
3. What are nested queries ? How would you use the operators, IN, EXISTS, UNIQUE, ANY, ALL in writing nested queries ?
4. What is grouping ? Discuss the interaction of the **having** and where clauses.
5. Consider following database.
   customer (cust_no, cust_name, addr)
   plant (plant_code, pl_name, pl_type, pl_cost)
   Nutrients (N_name, N_quantity, time)
   Customer and plants are related with one-to-many and plant and nutrients are also related with one-to-many relationship.
   Create RDB and answer following queries.
   (1) List the customer who purchase plant 'rose'.
   (2) List the plants whose cost is more than Rs. 500/-.
   (3) List all those plants to which nutrient 'manure' is given.

Q.3 Define the Terms:

DDl, DML, SQL, Predicate, varchar, constraint, primary key, foreign key, unique key, group by clause

Previous Exam Questions

Summer 2015

1. List various DDL commands. Explain any one with example. **4 M**

Ans. Refer to Section 4.4.

2. What are different aggregate functions in sql. **4 M**

Ans. Refer to Section 4.8.

3. Attempt the following

Consider the following entities and their relationship **16 M**

Game (gno, gname, no-of-player, coach_name)

player (no, pname)

Game and player are related with many-to-many relationship.

Create RDB in 3NF and solve the following queries by using SQL (any five):

- Insert a row in Game table.
- List total number of players playing "Basket Ball".
- Display all players having coach as "Mr. Dixit".
- Add birth date column to player table. Use alter table command.
- List all games played by Amit.
- Count total No. of players whose coach name is "Mr. Sharma".

Ans. Refer to 'Solved Case Studies' Section.

4. Explain the terms of commands. **2 M**

(i) Group By.

Ans. Refer to Section 4.8.

(ii) Order By.

Ans. Refer to Section 4.6.6.

Summer 2016

1. What is DDL and DML ? Explain any one DDL command with example. **4 M**

Ans. Refer to Sections 4.4 and 4.5.

Winter 2016

1. Explain different aggregate functions with example.

Ans. Refer to Section 4.8.

2. Explain the various DCL commands with example.

Ans. Refer to Section 4.6

3. Consider the following entities and relationships.

Teacher (t_no, t_name, college_name, dept)

E-test (e_no, test_name)

Teacher and E-test are related with Many-to-Many relationships.

Create a RDB for the above and solve the following queries:

(a) Give the name of teachers who have passed either "SET" or "NET".

(b) Count the no. of teachers who passed 'NET' exam of 'Computer Management'.

(c) Delete all the teachers' details of 'Biology' dept.

(d) Display exam wise list of teachers who have passed the respective exams.

(e) Print the total no. of teachers passing the respective exam.

(f) Insert the teacher record in to the table teacher.

Ans. Refer to 'Solved Case Studies' Section.

4. Explain the terms of commands:

(i) Order by

Ans. Refer to Section 4.6.6.

Summer 2017

1. Explain the various DDL commands with examples. **4 M**

Ans. Refer to Section 4.4.

2 Attempt the following **16 M**

Consider the following entities and relationships.

Item (I_no, I_name, I_qty)

Po (P_no, P_date)

Supplier (S_no, S_name, S_addr)

Item and Po are related with one to many relationships along with descriptive cost and quantity. Supplier and Po are related with one-to-many relationships.

Create a RDB for the above and solve the following queries:

(i) Insert a row in Item table.

(ii) List the name of supplier to whom Po is given for "mouse".

(iii) List the name of supplier and item_name in Po's generated on "30-sep-2009".

(iv) List the names of suppliers who is going to supply "monitor" with minimum cost.

(v) Find out Po number, Po date and supplier name of the Po which is of maximum amount.

(vi) Display all Po which contains the number, date, supplier name of the Po details of all items included in that i.e. name of item, qty and rate.

3. Explain the following terms **1 M**

(i) Group by.

Ans. Refer to Section 4.8.

4. Explain different aggregate functions with an example. **4 M**

Ans. Refer to Section 4.8.

Winter 2017

1. Explain Referential Integrity with suitable example.

Ans. Refer to Section 4.4.2.

2. Explain aggregate functions in SQL with examples.

Ans. Refer to Section 4.8.

3. List various DDL commands. Explain any one with example.

Ans. Refer to Section 4.4.

4. Consider the following Entities and Relationship.

Employee (empno, empname, salary, commission, desgination)

Department (deptno, deptname, location)

Relationship between Employee and Department is many-to-one.

Constraints: Primary key, Salary should be > 0.

Create a RDB in 3NF and write queries in oracle:

(a) Display all details of employees who are working at 'Pune' location.

(b) Display department namewise list of employees.

(c) Count the number of employees who are working in 'computer' department.

(d) Display maximum salary for every department.

(e) Display average salary for every designation.

(f) Update commission for every employee by 5% for all department.

Ans. Refer to 'Solved Case Studies' Section.

Summer 2018

1. Explain the various DML commands with examples. **4 M**

Ans. Refer to Section 4.5.

2. Explain the syntax of ALTER Table. **4 M**

Ans. Refer to Section 4.4.4.

3. Attempt the following **16 M**

Consider the following entities and their relationship:

Game (gno, gname, no-of-player, coachname)

Player (pno, pname)

Game and player are related with many-to-many relationship.

Create RDB in 3NF and solve the following queries using SQL (any five):

(a) Delete a row from Game table for game "Cricket".

(b) Display all players who play game "Table Tennis".

(c) List all games played by Rajesh.

(d) Add column Age in the player table.

(e) Count total no. of players whose coach is "Kiran".

(f) Count max no. of players in a game.

Ans. Refer to 'Solved Case Studies' Section.

4. Define Order by and Group by. **4 M**

Ans. Refer to Sections 4.6.6 and 4.8.

❖❖❖

5... Relational Database Design

Learning Objectives...

- To know about basic concept of relational database design.
- To learn about normalization and normal forms of database.

5.1 INTRODUCTION

- A relational database is made up of a number of relations and the relational database scheme which consists of number of attributes.
- Data is organized in the form of table, so data retrieval, updates, processing and structural changes are more efficient.
- In this chapter, we focus on the issues involved in the design of a database scheme using a relational model.
- The goal of a relational database design is to generate set of relation schemes that allows us to store information without unnecessary redundancy and also to retrieve the information easily.
- One approach is to design schemes with normal forms. The normal forms used to ensure that various types of anomalies and inconsistencies are not introduced into the database.
- To determine whether a relation scheme is in one of the normal forms, we actually need the additional information about real world. This additional information is given by a collection of constraints called *data dependencies*.
- Hence, good database design is extremely important for ensuring that the data which we want to store is consistent.
- Also data can be updated and retrieved in efficient manner. Some of the factors which are considered for designing databases are:
  - How many tables should we have ?
  - How the tables are inter related ?
  - How many columns should each table have ?
  - Which are the key columns ?

- What type of constraints we have ?
- How can we save the memory space ?

5.2 ANOMALIES OF UNNORMALIZED DATABASE (S-18, 17; W-17)

- To understand the pitfalls in any relational database, we will see a simple example of student database.
- A student's information is maintained in the database, such as rollno, name, address, class, subject, teacher who taught this subject with teacher_id.
- This information is maintained in a single table. See Table 5.1.

Table 5.1: Student database

rollno	name	address	class	subject	teacher_id	teacher_name
1	Jack	London	TY	java	1	Kelvin
1	Jack	London	TY	php	2	Rozar
2	Zara	Paris	FY	C	3	Adams
3	Smith	New York	SY	C++	4	Blacke
3	Smith	New York	SY	Mysql	1	Kelvin
2	Brouce	London	TY	php	1	Rozar

- By observation, we can see that data is repeated again and again that is called redundancy.
- *Redundancy* consumes more space and resources than necessary.
- Redundancy may lead to *inconsistency*.
- This can be removed by separating these tables which are having keys.
- For example:
  (1) Create a table Student which have rollno, name and address.
  (2) Create a table Class with class_id and class.
  (3) Create a table Teacher with T_id and T_name.
  (4) Then create the relationship between them using foreign key concept.
- Users can ask queries and database provides proper results.
- For designing the tables, consider two important criteria.
- When a database is stored, avoid redundancy i.e. repetition of data should not be available in a database.
- When a database is stored, it must be in a normalized form. This will provide better efficiency and reliability. This also allows us to use storage space efficiently, eliminate data redundancy and also reduce inconsistency in database.

Example 1:

- Let's consider Bank database from scratch.
- Suppose one customer can go to the bank to take loan.

- So the necessary information which we should maintain is shown in Table 5.2.

  relation = Lending

Table 5.2: Relation = Lending loan

branch_name	branch_city	assets	cust_name	loan_no	Amount
Adam Street	London	4,10,000	Smith	1	2,00,000
Bezant Street	London	37,00,000	Brouce	4	5,00,000
Willam Street	New York	4,50,000	Jill	1	2,00,000

- Suppose, if we want to add a new loan in our database i.e. Lending.
- Assume that the loan is made by Adam Street to Smith with the amount of Rs. 50,000 and loan_no is 3.
- So we need to add the tuple in the lending relation.

  (Adam Street, London, 400000, Smith, 3,50000).
- In general, we were repeating assets and city data for a branch, for each loan made by that branch.
- So data is repeated (*Redundancy* occurred).
- Let's consider that by mistake the branch_city name is typed as 'Londn' in new tuple.
- If a query is written to find the loans done into London city, it won't consider the newly added record.
- So *loss of information* (inconsistency) is also available.
- Similarly, if user wants to delete a loan tuple, while deleting a loan tuple branch_name will also get deleted.
- If that is the only loan made by that branch, the database will loose that information about branch_name and its assets also.
- To eliminate redundancy and inconsistency, the data should be in normalized format.

5.3 NORMALIZATION (S-18, 17; W-16, 17)

- Normalization is simply the process of distilling the structure of database to the point where user removes repeated group of data into separate tables.
- In this normalization process, grouping of data can be done.
- It is essentially the process in which a wide table with lots of columns but few rows is taken and redesigned it as several narrow tables with fewer columns but more rows.
- The components of normalization are Normal Forms.
- Normal form is a progressive series of rules that can be applied to simplify and refine each relation. In 1970 (1NF) and 1971 (2 NF and 3 NF). Codd classified and defined the normal forms. Initially, Codd proposed three normal forms i.e. First, Second and Third normal form.

Advantages:

1. It improves query retrieval performance.
2. It eliminates insertion, deletion and modification problems to great extent.
3. It reduces redundancy so problems are minimized.

Disadvantages:

1. It degrades query retrieval performance.
2. Numeralized tables will loose real world meaning.

- A stronger definition of 3NF was proposed by Boyce and Codd, it is known as BCNF i.e. **Boyce-Codd Normal Form (BCNF).**
- The Fig. 5.1 indicates all normalize relations. The database designer should generally aim for a design involving relation in 3NF and not in 2NF and 1NF.

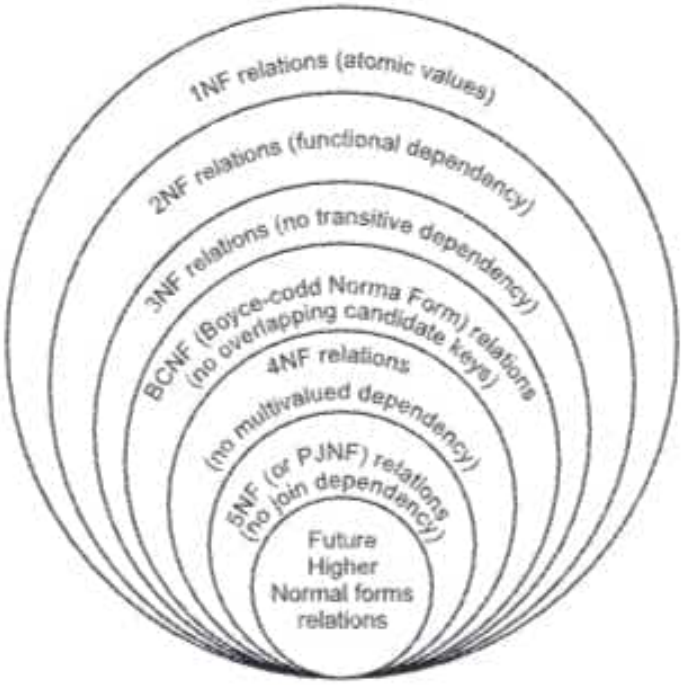


Fig. 5.1: Normal Forms

- The goal of normalization is to define database tables that can be updated and modified with predictable results. The ultimate benefit of normalization is *data integrity*.
- If a relational database is not designed properly, it contains too many anomalies or problems.
- The problems are created while inserting new data, modifying existing data, deleting existing data.
- Different problems are data redundancy or duplication of data and unavailability of data etc.

Example 2:

- Consider the following relation stud_info (name, course, ph_no, grade, prof).
- Let's consider atleast 4 records available into table called Stud_info. This is shown in Fig. 5.2.

Stud_info relation

name	course	ph_no	grade	prof
Roger	TCS	111122	Ist	Blacke
Mac	FDB	423456	Ist	Van
Smith	DC	282852	IInd	Adam
Jhons	C++	493358	Pass	Anna

Fig. 5.2: Stud_info relation

- This relation scheme can lead to several undesirable problems.

5.3.1 Problems Caused by Redundancy

(a) Wastage of Storage Space :

- The aim of database is to reduce the redundancy.
- It means that information is to be stored only once.
- Storing information for several times leads to the waste of storage space and an increase in the total space of the data stored.
- In Stud_info relation, if single student offers multiple subjects, his/her ph_no and course field are repeated unnecessarily.
- For example, if we say that Roger offers for subject DC also. The information entered into the database will have one more record i.e. (**Roger, DC, 111122**, Ist, Adam). So DC and 111122 are repeated. Similarly, Smith has opted for DC. So name of prof. i.e. Adam is repeated.

(b) Update Anomalies: (W-16)

- Multiple copies of the same field may lead to update anomalies i.e. problems.
- In above relation, if a change is required in ph_no of a particular student say Roger, then that change must get implemented in all occurrences of same phone number.
- If one of the tuples is not modified to reflect new phone_no, there will be an inconsistency.
- Similarly, if there is a change in specific course professor, then same change must be implemented in all occurrences, failure of which will lead database into an inconsistent state.
- Same thing is also applicable for all other repeating fields.

(c) Insertion anomalies:

- In this case, if any existing student is going to offer any more subjects, then it increases the duplication while inserting new record.

- Similarly, in above relation, it is not possible to insert either only student information with subject and corresponding professor or only course and professor without student's personal information i.e. all fields values must be specified for each record.
- Example: we can not add a tuple with the information (Roger, DC).

(d) Deletion anomalies:

- If the only student registered in a given course discontinues or cancels his/her registration from the course, the information has to which professor is offering the course will be lost, since this is the only relation in the database showing the association between a faculty member and the course she or he teaches.
- In other words, if any course is offered only by one student, then corresponding professor information will be available only in that record which is to be deleted. Hence, after deleting this record the professor information will be deleted unnecessarily.
- To remove all the above problems normalization is needed which ultimately provides benefit of **data integrity**. This benefit is provided because of following reasons:
  - Simplify entities by creating relations with one theme.
  - Build tables that can be easily joined with other tables to produce information.
  - Reduce redundant data across relations (because data storage is only in one place).
  - Avoid loss of data by requiring that each field contains atomic values and by designating appropriate primary and foreign keys.
  - Reduce modification anomalies such as deletion, insertion and updation.
  - Define relation constraints (conditions) that are logical consequence of domain and keys.
- To avoid above problems normalization is important.
- There are two approaches available to represent the database design in a normal form:
  1. Decomposition
  2. Synthesis
- The *decomposition approach* starts with one relation and associated set of constraints in the form of:
  - Functional dependency
  - Multivalued dependency
  - Join dependency
- A relation that has any undesirable properties in the form of above anomalies, repetition of information or loss of information, is replaced by its projections.
- This results in normal forms.

- This chapter deals with decomposition approach and following normal forms:
  1. First Normal Form (1NF)
  2. Second Normal Form (2NF)
  3. Third Normal Form (3NF)
  4. Boyce Codd Normal Form (BCNF)
- The *synthesis approach* starts with a set of functional dependencies on a set of attributes. Then it synthesizes the relation to Third Normal Form (3NF).
- The database design with any above approach is always a good design with all its properties.

5.3.2 Use of Decomposition

- Decomposition means breaking up a relation schema into smaller relation schemas. In order to have a good database design it is necessary to decompose big relations into smaller and meaningful relations.

Definition:

- The decomposition of a relation scheme $R = \{A_1, A_2, A_3, ..., A_n\}$ is it's replacement by a set of relation schemas $\{R_1, R_2, R_3, ..., R_m\}$ such that,

$$R = R_1 \bowtie R_2 \bowtie R_3 \bowtie ..... \bowtie R_m.$$

Example 3:

- Consider the Stud-info relation, it is decomposed into three sub-relations as follows:

  Stud_info (name, ph_no)

  R_1 (name, course, grade)

  R_2 (course, prof).

Example 4:

- Consider the relation schema for relation lending and it is shown in Table 5.2.
- The lending schema is decomposed into following two schemas.

  Branch_cust (branch_name, branch_city, assets, cust_name)

  Cust_loan (cust_name, loan_no, amount)
- These two relations are actually constructed by following relational algebra expression from table 5.2.

  Branch_cust = $\Pi_{branch\_name, branch\_city, assets, cust\_name}$ (lending)

  Cust_loan = $\Pi_{cust\_name, loan\_no, amount}$ (lending).
- Let's add certain records to these two newly created relations. This is shown in Fig. 5.3 and Fig. 5.4.

  Relation = Branch_cust

branch_name	branch_city	assets	cust_name
Adam Street	London	400000	Smith
Bezant Street	London	3700000	Brouce
Willam Street	New York	450000	Jill

Fig. 5.3: Branch_cust Relation

Relation = Cust_loan

cust_name	loan_no	amount
Smith	1	200000
Brouce	4	500000
Jill	1	200000
Smith	2	100000

Fig. 5.4: Cust_loan Relation

- We can reconstruct the original lending relation as follows:
  Branch_cust ⋈ Cust_loan
- The result of this natural join is shown in Fig. 5.6 and the intermediate result is shown in Fig. 5.5.

branch_ name	branch_ city	assets	cust_ name	cust_ name	loan_no	amount
Adam Street	London	400000	Smith	Smith	1	200000
Adam Street	London	400000	Smith	Brouce	4	500000
Adam Street	London	400000	Smith	Jill	1	200000
Adam Street	London	400000	Smith	Smith	2	100000
Bezant Street	London	3700000	Brouce	Smith	1	200000
Bezant Street	London	3700000	Brouce	Brouce	4	500000
Bezant Street	London	3700000	Brouce	Jill	1	200000
Bezant Street	London	3700000	Brouce	Smith	2	100000
Willam Street	New York	450000	Jill	Smith	1	200000
Willam Street	New York	450000	Jill	Brouce	4	500000
Willam Street	New York	450000	Jill	Jill	1	200000
Willam Street	New York	450000	Jill	Smith	2	100000

Fig. 5.5: Intermediate Result

cust_name is the common attribute. So select common cust_name which will give the natural join.

branch_name	branch_city	assets	cust_name	loan_no	amount
Adam Street	London	400000	Smith	1	200000
Adam Street	London	400000	Smith	2	100000
Bezant Street	London	3700000	Brouce	4	500000
Willam Street	New York	450000	Jill	1	200000

Fig. 5.6: Result of Natural Join

- Here, the result has one additional tuple i.e. (Adam Street, London, 400000, Smith, 2, 100000).

Query: Find all the customers who had taken loan from Adam Street branch.

- The result will show two different loan_no's different loan amount.
- It means that the information is not correct, i.e. it results in loss of information.
- For the above case, it is clear that the given decomposition results in bad database design.
- This is actually referred as lossy_join decomposition. This occurred because there is one attribute in common between both the relations.
- But this representation is not adequate because a customer may have several loans.
- Let's consider alternate design, i.e. we decompose the table lending into the following two relations:

  Branch = (**branch_name**, branch_city, assets)

  Cust_loan = (branch_name, **cust_name**, loan_no, amount).

  The common attribute is branch_name.
- Lets add some tuples in both the relations. The relations are shown in Fig. 5.7 and Fig. 5.8.

relation = Branch

branch_name	branch_city	assets
Adam Street	London	400000
Willam Street	New York	800000
Bezant Street	New York	800000

Fig. 5.7: Branch Relation

Relation = Cust_loan

branch_name	cust_name	loan_no	amount
Adam Street	Smith	1	100000
Adam Street	Sujit	2	230000
Willam Street	Simran	4	380000
Bezant Street	Kelvin	3	300000
Bezant Street	Evan	1	400000

Fig. 5.8: Cust_loan Relation

- Let's reconstruct the lending relation as follows:

  Branch ⋈ Cust_loan

- The result of this is shown in Fig. 5.9.

branch_name	branch_city	assets	cust_name	loan_no	amount
Adam Street	London	400000	Smith	1	100000
Adam Street	London	400000	Sujit	2	230000
Willam Street	New York	800000	Simran	4	380000
Bezant Street	New York	800000	Kelvin	3	300000
Bezant Street	New York	800000	Evan	1	400000

Fig. 5.9: Lending: Result of Natural join

- Write the same query as follows:

Query: Find all the customers who had taken loan from Adam Street.

- Now, here from lending relation we will get proper result. No loss of information is here with this newly created relations. This actually shows the good database design.

Example 5:

Consider a database:

Patient (pno, pname, address, dno, dname, symptoms, causing_organisms)

Decompose this relation into sub-relations.

Solution:

- When we decompose the relation, we need to consider the relationship in terms of One to Many, Many to One or Many to Many.
- According to the relationship we need to place the primary keys into the other table/relation.
- Here we decompose, Patient, Doctor and Organism tables or relations separately.

  Patient (pno, pname, address)

  Doctor (dno, dname)

  Organism (symptoms, causing_organisms)

Problems occur in decomposition:

1. Loss of information called lossy decomposition.
2. Loss in some of functional dependency relationship in lossy decomposition, so we use lossless decomposition.

5.4 NORMAL FORMS

- After the overview of functional dependencies, redundancy and decomposition, we take a look at normal forms. Normalization is different from decomposition.
- Decomposition does not abide by any formal rules, whereas normalization does.

- When we apply a normalization rule, the database design takes the next logical form called the *Normal Form.*
- Normalization of data can be considered as a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas those possess desirable properties.
- Let's talk about the basic terms related with normalization.

Atomic Field and Non-atomic Fields:

- A field which is not further splittable into meaningful subparts or which does not contain the sub fields is called as **atomic field**. Otherwise it is a **non-atomic field**.
- For example: student (roll_no, course, exam).
- Here roll_no and course are atomic fields. But, exam contains exam_date, subject_code, max_marks, etc. subfields into it. Hence it is referred to as non-atomic field.
- When we start with any database it has **unnormalized relations** into it.
- The unnormalized relation is the relation which may contain,
  - some non-atomic fields.
  - many keys or it may not contain any common key for identifying all other non-key attributes.
  - some anomalies which are related with data insertion, modification, deletion, etc.
  - it contains almost all fields.
  - it may contain some duplicate values for the multiple records.

Example 6: Consider the following relation.

Stud (roll, course, prof, sport_name, sport_code, exam)

In this relation, the functional dependencies are roll → course, course → prof,

Sport_code → sport_name, roll → exam.

Here, the exam field contains exam_code, sub_code, max_marks, scored marks, etc.

- This relation is unnormalized as it contains some non-atomic fields. It does not contain a key which uniquely identifies all other attributes or each tuple of given relation as shown in Fig. 5.10.

roll	course	prof.	sport_code	sport_name	Exam			
					date	sub_code	max_marks	scored marks
1	FDB	Adam	S1	cricket	1-3-19	C1	100	55
					1-3-19	C2	100	60
					1-3-19	C3	100	70

Fig. 5.10: Unnormalized Stud Relation

- It contains some anomalies for data processing, since for single record of student there are multiple entries for exam field.
- To remove the problems of this relation, it can be normalized through different normal forms.

5.4.1 First Normal Form (1NF) (S-17)

- The definition of First Normal Form is,
  - There are no repeating groups.
  - All the key attributes are defined.
  - All attributes are dependent on the primary key.
- A relation is said to be in first normal form (1NF) if all fields are atomic. In other words, only one value is associated with each attribute.
- A database scheme is in 1NF if every relation schema included in the database schema is in 1NF.
- For example, above relation stud can be replaced by a new relation. N_stud as shown in Fig. 5.11.

roll	course	prof	sport_ code	sport_ name	exam_ date	sub_ code	max_ marks	scored_ marks
1	FDB	Adam	S1	cricket	1-3-19	C1	100	55
1	FDB	Adam	S1	cricket	1-3-19	C2	100	60
1	FDB	Adam	S1	cricket	1-3-19	C3	100	70

Fig. 5.11: N_stud Relation (1 NF)

- Here each field is now in atomic form. Though some field values are getting repeated for same record multiple times, it is simpler than non-atomic fields.

5.4.2 Second Normal Form (2NF)

- A relation schema R is in Second Normal Form (2NF) if,
  - it is in 1 NF and
  - all its non-prime attributes are fully functionally dependent on the primary key of R.
- A database schema is in second normal form, if every relation schema included in the database schema is in 2NF.
- So, we go through all the fields, we can see that sport_name is only dependent on sport_code. Exam_date, max_marks, scored_marks are dependent upon sub_code.
- So by the method of decomposition, we will have the following 3 tables.

1. **Stud table**

   roll (primary key)
   course
   prof

2. **Sport table**
   sport_code (primary key)
   roll
   sport_name
3. **Exam table**
   roll
   Exam_date
   sub_code (primary key)
   max_marks
   scored_marks.

Example 7:

- R (name, course, ph_no, addr, grade)
  The functional dependencies in this relation are,
  name → course, name → ph_no, name → addr and name, course → grade.
- If a combination of the fields name, course is used as a key then,
  → grade field is fully functionally dependent on it.
  → ph_no and addr are dependent only on name i.e. these fields are partially dependent on relation key.
- This relation is already in 1NF, as it contains only atomic fields, but it is not in 2NF because it contains some partial functional dependencies.
- It also contains some anomalies while inserting, deleting or modifying data. Also it contains some data duplications.
  - For each course, the ph_no and addr fields are repeated.
  - Only student information cannot be inserted or only course with grade information cannot be inserted.
  - If a change is required either in address part or ph_no, then it requires change in each occurrence of same record.
- In order to remove these anomalies decompose or split this relation into multiple subrelations so that each relation will contain a key on which all other attributes are fully functionally dependent.
- Above relation can be decomposed as follows:
  R_1 (name, course, grade)
  R_2 (name, ph_no, addr)
  Now, both relations are having full functional dependency.

5.4.3 Third Normal Form (3NF)

- The definition of Third Normal Form is,
  - It's in 2nd normal form.

- It contains no transitive dependencies (where a non-key attribute is dependent on another non-key attribute).

- A relation schema R (S, F) is in 3NF if for all non-trivial functional dependencies in F^+ of the form $X \rightarrow A$, either X contains a key or A is a prime attribute.
- A database schema is in third normal form if every relation schema included in the database schema is in third normal form.
- A functional dependency $X \rightarrow Y$ is said to be non-trivial (Y is not a subset of X).

Definition: (3NF): A relation schema R is said to be in 3NF if it is in 2NF and if it does not contain any transitive functional dependency i.e. every non-key attribute is non-transitively dependent on the primary key.

Example 8:

Consider the relation S (roll, course, prof) where FD's are roll → course, roll → prof, course → prof. Here roll is primary key.

- As roll → course, course → prof, it implies that roll transitively identifies prof or in case of course → prof, neither course nor prof is form table key, hence this relation is not in 3 NF. To make it in 3 NF, split this relation as follows –

  S1 (roll, course) and S2 (course, prof)

A decomposition in 3 NF is always:

- loss-less join decomposition and
- dependency preserving.

5.4.4 Boyce Codd Normal Form (BCNF) (W-16)

- Consider the relation in 3NF has two (or more) candidate keys such that the two candidate keys were composite and they overlapped (i.e. had atleast one attribute in common).
- The original definition of 3NF was therefore replaced by a stronger definition which catered for above case also. Hence, the term Boyce Codd Normal Form (BCNF) is introduced. It was developed by R.P. Boyce and E.F. Codd.

Example 9:

Consider a relation in 3 NF that has a number of overlapping composite candidate keys.

stud (name, roll, course, grade)

with FD's are

name, course → grade,

roll, course → grade

name → roll

roll → name

- Thus, each student has a unique name and a unique student roll no. The relation has two candidate keys (*name, course*) and (*roll, course*).

- Each of the keys is a composite key and contains a common attribute *course*.
- The relation satisfies the criteria of 3NF, i.e. FD's $X \rightarrow A$ in stud, when $A \notin X$ either X is a superkey or A is prime. However, this relation has a disadvantage in the form of repetition of data.
- The association between a name and the corresponding roll is repeated, any change in one of these has to be reflected in all tuples, otherwise there will be inconsistency in the database.
- Furthermore, the student number cannot be associated with a student name unless the student has registered in a course and this association is lost if the student drops all the courses he or she is registered in.
- In the BCNF, which is stronger than the 3NF, these anomalies are avoided.
- This is done by ensuring that for all non-trivial FDs implied by the relation the determinants of the FDs involve a candidate key.
- For a relation scheme $R < S, F >$ to be in BCNF, for every FD in F^+ of the form $X \rightarrow A$ where $X \subseteq S$ and $A \in S$, atleast one of the following conditions hold:
  - $X \rightarrow A$ is a trivial FD and hence $A \in X$, or
  - $X \rightarrow R$ i.e. X is a superkey of R.
  - A database scheme is in BCNF, if every relation scheme in the database is in BCNF.
- The above definition of the BCNF relation indicates that a relation in BCNF is also in 3NF. The only non-trivial FDs allowed in the BCNF are those FDs whose determinants are candidate superkeys of the relation.

Example 10:

Consider the relation Student

Student (SID, name, ph_no, major)

where SID is an unique student identification number and name and ph_no are assumed to be unique.

- The FDs are,

  (SID → major, name → major, ph_no → major, SID → name, SID → ph_no,

  name → SID, name → ph_no, ph_no → SID, ph_no → name).
- The relation student is in BCNF since each FD involves a candidate key as the determinant.

Example 11:

Find a BCNF decomposition of the relation shipping with the following FDs shipping (ship, capacity, date, cargo, value)

$$\begin{aligned} \text{ship} &\rightarrow \text{capacity} \\ \text{ship, date} &\rightarrow \text{cargo} \\ \text{cargo, capacity} &\rightarrow \text{value} \end{aligned}$$

Solution: Step 1: First find non-redundant cover of the given set of FDs, F'.

There are no redundant FDs in the set, hence given set of FDs is a non-redundant cover.

Step 2: For all FDs in F', if there exists a non-trivial FD ($x \rightarrow y$) in F'^+ such that $XY \subseteq R_j$ and $X \rightarrow R_j$ (R_j is a relation in S, is not in BCNF i.e. $X \rightarrow R_j$ is not in F'^+), then add R_j (x, y) to S with Fd, $X \rightarrow Y$.

- Here, FD, ship → capacity and since ship → shipping we replace shipping with relation R_1 (ship, capacity) formed with the FD, ship → capacity and R_2 (ship, date, cargo, value).
- Consider the relation R_2; the FD, ship, date → cargo is a non-trivial FD in the non-redundant cover.
- However, since ship, date → ship, date, cargo, value, the relation R_2 is in BCNF and we have the loss-less BCNF decomposition as follows:

  R_1 (ship, capacity) with the FD ship → capacity

  R_2 (ship, date, cargo, value) with the FD ship date → cargo.
- The decomposition of shipping into R_1 and R_2 is loss less but not dependency preserving because the FD cargo capacity → value is not implied by the set of FDs (ship → capacity, ship date → cargo}.
- When we consider the FD cargo capacity → value the following decomposition is obtained.

  R_1 (cargo, capacity, value) with FD cargo capacity → value

  R_2 (ship, capacity) with FD ship → capacity

  R_3 (ship, date, cargo) with the FD ship date → cargo.
- This decomposition is also dependency preserving.

Difference between 3 NF and BCNF:

3 NF	BCNF
1. Its focus is on one key.	1. Its focus is on all candidate keys.
2. In 3 NF, possibility for high degree of insertion, deletion and modification problem due to candidate keys. i.e. Redundancy is high.	2. They are very much reduced as BCNF is taking care of candidate keys. i.e. Redundancy is low.
3. If there is a dependency x → y, it is allowed in 3 NF, if either x is a super key or y is part of some key.	3. If there is a dependency of the form x → y, x should be super key or candidate key.
4. Redundancy problem is more.	4. Redundancy problem is less.

SOLVED CASE-STUDIES

(S-18)

Case 1: Normalize the following table upto 3 NF step by step. The database is related to order-report shown in Fig. 5.12.

Order_no	Order_date	Invoice_no	Invoice_date	Product_no	Product_name	Quantity_ordered
25	17/3/13	1231	30/4/13	1	Computer	5
130	17/3/13	1232	30/4/13	2	Printer	5
520	28/3/13	1233	30/4/13	2	Printer	1
				4	A4 pages	10
				5	Main sheets	70
				6	Mouse	10

Fig. 5.12: Consolidated Order Report

Solution:

- The given data is in unnormalized form.
- First convert it into 1 NF.

1 NF:

- No repeating groups.
- For this, partition each data structure containing repeating groups which accomplishes the same purpose.
- Fig. 5.13 is the result of 1 NF.

Order_no	Order_date	Invoice_no	Invoice_date	Product_no	Product_name	Quantity_ordered
25	17/3/13	1231	30/4/13	1	Computer	5
130	17/3/13	1232	30/4/13	2	Printer	5
520	28/3/13	1233	30/4/13	2	Printer	1
520	28/3/13	1233	30/4/13	4	A4 pages	10
520	28/3/13	1233	30/4/13	5	Main sheets	70
520	38/3/13	1233	30/4/13	6	Mouse	10

Fig. 5.13 Result of 1NF

- We remove the four repeating elements and group them into a new table called order_record. [See Fig. 5.14 (a)]
- Rest of the elements are added into a new table called order_item. [See Fig. 5.14 (b)]

Order_no	Order_date	Invoice_no	Invoice_date
25	13/3/13	1231	30/4/13
130	17/3/13	1232	30/4/13
520	28/3/13	1233	30/4/13

Fig. 5.14: (a) Order_record

Order_no	Product_no	Product_name	quantity
25	1	computer	5
130	2	printer	5
520	2	printer	1
	4	A4 pages	10
	5	mainsheets	70
	6	mouse	10

Fig. 5.14: (b) Order_item

- Both the relations order_record and order_item are in 1 NF.
- But order_item is not in its ideal form.
- Some of the attributes are not functionally dependent on primary key (order_no, product_no).
- Some non-key attributes such as product name, quantity are dependent only on product_no and not on the key which we have considered.

2 NF:

- All the non-key fields are dependent on the primary key (composite key).
- For this, verify that each non-key field is in first normal form and fully functionally dependent on primary key.
- Product no → product description is the functional dependency.
- To convert order_item into 2 NF, we separate the table (decompose) such that the above functional dependency satisfies correctly. [See Fig. 5.15 (a), 5.15 (b), 5.15 (c)]

Order_no	Order_date	Invoice_no	Invoice_date
25	13/3/13	1231	30/4/13
130	17/3/13	1232	30/4/13
520	28/3/13	1233	30/4/13

Fig. 5.15 (a): Order_record

Order_no	Product_no	Quantity_ordered
24	1	5
130	2	5
520	2	1
	4	10
	5	70
	6	10

Fig. 5.15 (b): Order_item

Product_no	Product_ description
1	computer
2	printer
4	A4 pages
5	mainsheets
6	mouse

Fig. 5.15: (c) Product

3 NF:

- When all non-key fields are independent of one another, then relation is in 3 NF.
- There is check for redundancy within the relation.

- Duplicate data elements which can be derived from other elements are removed at 3 NF.
- For example, in order-record relation we have invoice date which is dependent on both i.e. order_no and invoice_no.
- To convert the Order_Record into 3 NF we have to remove invoice_date and group it with invoice_no as the key in separate table.
- So we have following 3 NF relations: [See Fig. 5.16 (a), (b), (c), (d)]

Order_no	Order_date	Invoice_no
25	13/3/13	1231
130	17/3/13	1232
520	28/3/13	1233

Fig. 5.16 (a): Order-record

Order_no	Product_no	Quantity_ ordered
25	1	5
130	2	5
520	2	1
520	4	10
520	5	70
520	6	10

Fig. 5.16: (b) Order-item

Product_no	Product_description
1	computer
2	printer
4	A4 pages
5	mainsheets
6	mouse

Fig. 5.16 (c): Product

Invoice_no	Invoice_date
1231	30/4/13
1232	30/4/13
1233	30/4/13

Fig. 5.16 (d): Invoice

- Therefore the relationship is transformed into following relations (3 NF)
- Order_record – order_no – primary key
  Order_date
  Invoice_no – foreign key
- Product – product_no – primary key
  Product_ description
- Order_item – order_no – primary key
  Product_no – foreign key
  Quantity_ordered
- Invoice – invoice_ no – primary key
  Invoice_date

Case 2: The given Sales_report in un-normalized relation since it has repeating groups. Normalize this relation upto 3 NF. The Sales_report is shown in Fig. 5.17.

Sales_ person_ no	Sales_ person_ name	Sales_ area	Cust_no	Cust_name	Warehouse_ no	Warehouse_ location	Sales_ amount
501	Vinit	Aundh	101 102 301	CMS IBM Lavasa	4 3 4	Mumbai Thane Mumbai	2 lakh 2.5 lakh 4 lakh
345	Pritesh	Warje	45 127	Infosys Microland	2 1	Chennai Bangalore	55 Thousand 1.2 lakh

Fig. 5.17: Sales_report

Solution: The normalized sales report is as shown in Fig. 5.18.

Sales_ person_ no	Sales_ person_ name	Sales_ area	Cust_no	Cust_name	Warehouse_ no	Warehouse_ location	Sales_ amount
501	Vinit	Aundh	101	CMS	4	Mumbai	2 lakh
501	Vinit	Aundh	102	IBM	3	Thane	2.5 lakh
501	Vinit	Aundh	301	Lavasa	4	Mumbai	4 lakh
345	Pritesh	Warje	45	Infosys	2	Chennai	55
345	Pritesh	Warje	127	Microland	1	Banglore	thousand 1.2 lakh

Fig. 5.18: Normalized Sales Report

1 NF:

- Create two separate structures Sales person (Fig. 5.19) and Sales_customer (Fig. 5.20).

Sales_person_no	Sales_person_name	Sales_area
501	Vinit	Aundh
345	Pritesh	Warje

Fig. 5.19: Sales person

Sales_person_no	cust_no	cust_name	warehouse_no	w_location	Sales_amount
501	101	CMS	4	Mumbai	2 lakh
501	102	IBM	3	Thane	2.5 lakh
501	301	Lavasa	4	Mumbai	4 lakh
345	45	Infosys	2	Chennai	55 thousand
345	127	Microland	1	Banglore	1.2 lakh

Fig. 5.20: Sales-customer

- In sales-customer, some of the attributes are not functionally dependent on primary key – sales person no, cust_no.

2 NF:

- Sales_person table is as it is. [See Fig. 5.21 (a)]
- Sales_customer will be decomposed into two more relations i.e. sales [See Fig. 5.21 (b) and warehouse [See Fig. 5.21 (c)].

Sales_person_no	Sales_person_name	Sales_area
501	Vinit	Aundh
345	Pritesh	Warje

Fig. 5.21 (a): Sales_person

Sales_person_no	Cust_no	Sales_Amount
501	101	2 lakh
501	102	2.5 lakh
501	301	4 lakh
345	45	55 thousand
345	127	1.2 lakh

Fig. 5.21 (b): Sales

cust_no	cust_name	warehouse_no	warehouse_location
101	CMS	4	Mumbai
102	IBM	3	Thane
301	Lavasa	4	Mumbai
45	Infosys	2	Chennai
127	Microland	1	Bangalore

Fig. 5.21 (c): Warehouse

- Warehouse location (warehouse location) is not fully dependant on cust_no. i.e. it has non-key dependencies.
- We will go for 3 NF.

3 NF:

- Sales person and sales relations are as it is. [See Fig. 5.22 (a), Fig. 5.22(b)].
- Warehouse is decomposed into customer [Fig. 5.22 (c)] and warehouse [Fig. 5.22(d)].

Sales_person_no	Sales_person_name	Sales_area
501	Vinit	Aundh
345	Pritesh	Warje

Fig. 5.22 (a): Sales_person

Sales_person_no	Cust_no	Sales_amount
501	101	2 lakh
501	101	2.5 lakh
501	301	4 lakh
345	45	55 thousand
345	127	1.5 lakh

Fig. 5.22 (b): Sales

Cust_no	Cust_name	Warehouse_no
25	CMS	4
102	IBM	3
301	Lavasa	4
45	Infosys	2
127	Microland	1

Fig. 5.22 (c): Customer

warehouse_no	warehouse_location
4	Mumbai
3	Thane
2	Chennai
1	Bangalore

Fig. 5.22 (d): Warehouse

- Therefore, the sales_report relation is transformed into following 3NF relations:
- Sales_person – sales_person_no – primary key
  sales_person_name
  sales_area
- Sales – cust_no – foreign key
  sales_amount
  sales_person_no – foreign key
- Customer – cust_no – primary key
  cust_name
  warehouse_no – foreign key
- Warehouse – warehouse_no – primary key
  warehouse_location

Case 3: Normalize the following relation.

Land system
Soil type
Erodibility (decaying of plant)

Solution: Erodibility is uniquely determined by soil-type.

Let consider some record into unnormalized relations shown in Fig. 5.23.

Land_system	Soil_type	Erodibility
Faraway	Loamy sand	0.10
Limestone	Sandy loam	0.25
Mundooka	Loamy sand	0.10

Fig. 5.23: Unnormalized Relation

1 NF: Separate repeating groups

Soil table

land_system	primary key
soil_type	

Erosion table

soil_type	primary key
erodability	

- This is normalized relation so no need to go till 3 NF because at the start only it is normalized.

Case 4: Normalize the following data.

project_no
project_name
emp_no
emp_name
rate_category
hourly_rate

Solution: Above is the unnormalized data.

1 NF: Separate repeating groups

Employee-project table

project_no – primary key
project_name
emp_no – primary key
emp_name
rate_category
hourly_rate

2 NF: Find the elements depending upon primary keys or partial keys.

Employee Project Table

project_no – primary key
emp_no – primary key

Employee table

emp_no – primary key
emp_name
rate_category
hourly_rate

Project table

project_no – primary key
project_name

3 NF: Find non-key attributes. Emp_name is not dependent on either rate_category or hourly_rate. Same is applied to rate_category. But, hourly_rate is dependent on rate_category.

Employee project table

project_no – primary key
emp_no – primary key

Employee table

emp_no – primary key
emp_name
rate_category

Rate table

rate_category – primary key
hourly_rate

Project table

project_no – primary key
project_name

All above tables are in 3 NF and ready to implement.

Case 5: Normalize the following table of attributes upto 3 NF. The database is related to cricket association.

- The fields are –

Team_code	skill_name	club_city
player_code	club_code	club_district
team_name	umpire_code	
player_name	club_name	
player_age	umpire_name	
player_skill_code	club_address	

- Following assumptions to be made –
  1. A player can belong to one team and has only one skill.
  2. An umpire can belong to one club only.
  3. Club arranges matches between various teams and appoint umpires for the matches.

Solution: Unnormalized data:

club_code		team_code
club_name	team_name	
club_address	player_code	
club_district	player_name	
club_city		player_age
umpire_code	player_skill_code	
umpire_name		skill_name

1NF: Separate out the repeating group and find out key field.

Club table

club_code – primary key
team_code – secondary key
team_name
player_code
player_name
player_age
player_skill_code
skill_name

Umpire table

club_code – primary key
club_name
club_address
club_district
club_city
umpire_code
umpire_name

2NF: Find out the attributes depending upon the whole key and also the attributes depending upon the partial key.

CT table

club_code – primary key
team_code – secondary (partial key)

Team table

team_code – primary key
team_name

Player table

team_code
player_code – primary key
player_name
player_age
player_skill_code
skill_name

Club-table

club_code – primary key
club_name
club_address
club_district
club_city

Umpire table

club_code
umpire_code – primary key
umpire_name

3 NF: Find out the non-key elements depending upon some other non-key element. Extract the elements which can be directly computed.

From the Player table player_age, player_skill_code, skill_name can be separated out as they are non key elements depending upon the non key element namely player_name.

Similarly, club_name is a non key element.

table-1

player_name – primary key
player_age
player_skill_code
skill_name

and **table-2**

club_name – primary key
club_address
club_district
club_city

The key elements from the new structures are found out. In the Player table player_name is set out as a key field. In the club table, club_name is set out as a key field.

Case 6: Normalize the following data to 3 NF using appropriate relation.

Invoice

cust_code :	AC01	Inv. no :	011	Inv. date:	05/03/19
cust_name:	ABCL Ltd.	Po. no :	005	Po date:	05/03/19
cust_address :	Camp	Ch. no :	002	Ch_date:	05/03/19

Item no	Desc	Quantity	Rate	Amount
A005			**[Discount]**	——
			Inv.value	——

Solution: Unnormalized data:

Inv_no
Inv_date
Inv_value
Item_no
Desc
Rate
Qty
cust_code
cust_name
cust_address
po_no (purchase order no.)
po_date
Ch_no (Challan no.)
Ch_date
Amt

1 NF: Repeating group separated.

Invoice table

Inv_no – primary key

Item_no
Desc
Qty
Rate
Amt

Customer table

Inv_no – primary key
Inv_date
cust_code
cust_name
cust_address
po_no
po_date
Ch_no
Ch_date
Inv_value

2 NF: Find out elements depending upon the main key and partial key.

Item_1 table

Inv_no – primary key
Item_no
Qty

Item_2 table

Item_no – primary key
Desc
Rate
Amt

Invoice table

Inv_no – primary key
Inv_date
Inv_value

C_1 table

inv_no – primary key
cust_code

C_2 table

cust_code – primary key
cust_name
cust_address

C_3 table

Inv_no – primary key
Po_no

C_4 table

Po_no – primary key
Po_date

C_5 table

Inv_no – primary key
Ch_no

C_6 table

Ch_no – primary key
Ch_date

3 NF: We have to separate out the non-key elements depending upon some other non-key element.

In above example, only the cust_address is a non-key element which is depending upon other non-key element i.e. cust_name.

C_7 table

cust_name – primary key
cust_address

C_2 table

cust_code – primary key
cust_name

Case 7: Normalize the following data to 3NF using appropriate relation. Give brief explanation for each level.

Kaviman Ltd.					
cust_no	:	001	order no	:	1250
cust_name		Shekhar	order date	:	5/3/19
address	:	Camp	delivery date	:	10/3/19
item_no		description	rate		quantity
001		chair	65		100
014		table	1200		3
015		Washing machine	14000		2
Remark	:	Delivery at London H.O.			

Solution:

Step 0: Firstly, we have to write down the data in the unnormalized form. For this, the data elements related to each other are to be listed out one after another.

Unnormalized data:

cust_no
cust_name
address
order_no
order_date
delivery_date
item no
desc
rate
quantity
remark

Step 1:

- To achieve 1NF, remove all the repeating groups.
- A repeating group is actually another relation.
- Hence, it is removed from the record and treated as an additional record structure or relation.
- In this example item_no, description, rate and quantity, etc. repeats with each customer. Hence, we repeat it.
- The main key from the original structure is to be brought into repeating group structure.
- The new structure is named with appropriate name.

Item table

cust_no	primary key
item_no	partial key (secondary)
desc	
Rate, quantity	

Customer table

cust_no	primary key
cust_name	
address	
order_no	
order_date	
Delivery_date	
remark	

Step 2:

- As we know the above relation is in 1NF so we will go for 2NF we are looking for functional dependencies.
- For example, description of item is dependent on item_no, rate is dependent on item_no.
- Similarly, cust_name is dependent on cust_no., cust_address is dependent on cust_name and so on. We have to group such functionally dependent attributes and achieve 2NF.

So 2NF will have the following tables:

item_1 table

cust_no	primary key
order_no	
item_no	
quantity	

item_2 table

item_no	primary key
desc	
rate	

cust_1 table

cust_no	primary key
cust_name	
address	

cust_2 table

cust_no	primary key
order no	

cust_3 table

order_no	primary key
order_date	
delivery_date	
remark	

Step 3: In 3 NF we have to find out the non-key elements depending upon non-key elements. In the above structure address is a non-key element depends upon the cust_name which is also a non-key element. So, it can be defined as a new data structure and in which cust_name can be defined as a key field.

Summary

- In a relational model, objects are represented in tables. Each table is made out of rows and columns. Each row known as tuple or record. Each tuple is made out of fields known as attributes.
- Relations between tuples represent existing relationships between objects (tables).
- Integrity or consistency stands for the quality and reliability of data of a database system. A database is consistent if the data reflects the referenced objects correctly.
- Normalization is simply the process of distilling the structure of database to the point where user removes repeated group of data into separate tables.
- Decomposition means breaking up a relation schema into smaller relation schemas. In order to have a good database design it is necessary to decompose big relations into smaller and meaningful relations.
- When we apply a normalization rule, the database design takes the next logical form called the *Normal Form.*

Check your Understanding

1. Data dependencies are similar to .....
   (a) Constraints (b) Queries
   (c) Functions (d) Keys
2. Repetition of data again and again in any relations is .....
   (a) Dependency (b) Redundancy
   (c) Cardinality (d) Accuracy
3. Codd classified and defined the anomalies of normal form in the year .....
   (a) 1972 (b) 1973 (c) 1945 (d) 1947
4. BCNF is a more restrictive normal form than is .....
   (a) 1NF (b) 2NF (c) 3NF (d) 4NF
5. E-R model allows attributes that are .....
   (a) Multi-valued (b) Multi-layers
   (c) Multi-levels (d) Multi-relation

Ans.: (1) a (2) b (3) a (4) c (5) a

Practice Questions

Q.1 Answer the following questions in brief.

1. Which are needs of normalization.
2. What is normalization ?
3. What is data dependency?
4. What is decomposition in DBMS

5. Define the normalization rules:
   (a) BCNF (b) 3 NF (c) 2 NF

Q.2 Answer the following questions.

1. What are the desirable properties of decomposition ?
2. Explain different anomalies related with normalization.
3. Write short notes on:
   (a) BCNF (b) 1 NF (c) 2 NF (d) 3 NF
4. Normalize the following relation to 3 NF.
   (a) Ename, SSN, BDate, Address, D NUM (dept_num), D Name (dept_name)
   (b) plant_name, plant_code, plant_type, plant_price, cust_name, cust_code, no_plant, cust_address
   (c) doctor_code, doctor_name, patient_name, patient_code, ward_no, hospital_no, hospital_name
5. Consider a relational database customer
   customer (cno, cname, city, branch, br_no, loan_no, loan_amt, assets)
   Decompose this relation into subrelations.
6. Consider a relational database student and normalize it to 3 NF.
   student (rollno, name, address, phone_no, class_code, class_name, subject_code, subject_name)

Q.3 Define terms

Redundancy, normalization, data integrity, normal form, data dependency, decomposition.

Previous Exam Questions

Summer 2015

1. Write a note on normalization. **4 M**

Ans. Refer to Section 5.3.

Summer 2016

1. Explain the normalization process. **4 M**

Ans. Refer to Section 5.3.

Summer 2017

1. What is Normalization? Explain the 1NF with suitable example. **4 M**

Ans. Refer to Section 5.3 and 5.4.1

2. Write a short note on Normalization. **4 M**

Ans. Refer to Section 5.3.

3. Explain different anomalies related with Normalization. **4 M**

Ans. Refer to Section 5.2.

Summer 2018

1. What is Normalization ? Explain 1NF with suitable example. **4 M**

Ans. Refer to Section 5.3.

Winter 2016

1. Explain different anomalies related with normalization. **4 M**

Ans. Refer to Section 5.2.

2. Explain BCNF with suitable example. **4 M**

Ans. Refer to Section 5.4.4.

3. Explain update anomaly with suitable example. **4 M**

Ans. Refer to Section 5.3.1 (b).

Winter 2017

1. What are the anomalies of un-normalized database? **4 M**

Ans. Refer to Section 5.2.

2. Write a note on normalization. **4 M**

Ans. Refer to Section 5.3.

3. Define term: RDBMS. **1 M**

Ans. Refer to Section 5.1.

BIBLIOGRAPHY

1. Database System Concepts, 5th Edition : Silberschatz, Korth, Sudarshan.

2. Database Management System : V. K. Jain.

3. Database Management System, 3rd Edition : Raghu Ramkrishnan and Gehrke.

4. An Introduction to Database System : Bipin C. Desai.

5. Fundamental of Database System, 7th Edition : : Elmasri, Navathe.

6. SQl, PL/SQL The Programming Language Oracle: Ivan Bayross.

❖❖❖