Bachelor of Computer Application

(B.C.A.)

Introduction to Database Management System

Semester-II

Author- Dr. Chitra Desai (Atole)

SURESH GYAN VIHAR UNIVERSITY

Centre for Distance and Online Education
Mahal, Jagatpura, Jaipur-302025

Published by:

S. B. Prakashan Pvt. Ltd.

WZ-6, Lajwanti Garden, New Delhi: 110046

Tel.: (011) 28520627 | Ph.: 9205476295

Email: info@sbprakashan.com | Web.: www.sbprakashan.com

Designed & Graphic by : S. B. Prakashan Pvt. Ltd.

Printed at :

Syllabus

Data Base Management System

Learning Objective

- Students will be able to learn how to manage the data while using an IT enabled framework.
- Will be able to develop acumen about the system of data base i.e. what are the peripherals, how they work and even coordination.
- Will learn about a number of ERP systems that will enable them to manage the related components in real world.
- Student will be able to develop skills to manager data and use the same with utmost efficiency.

Unit 1

What is database system, purpose of database system, view of data, relational databases, database architecture, transaction management.

Unit 2

The importance of data models, Basic building blocks, Business rules, The evolution of data models, Degrees of data abstraction.

Unit 3

Database design and ER Model:overview, ER-Model, Constraints, ER-Diagrams, ERD Issues, weak entity sets, Codd's rules, Relational Schemas, Introduction to UML Relational database model: Logical view of data, keys, integrity rules. Relational Database design: features of good relational database design, atomic domain and Normalization (1NF, 2NF, 3NF, BCNF).

Unit 4

Relational algebra: introduction, Selection and projection, set operations, renaming, Joins, Division, syntax, semantics. Operators, grouping and ungrouping, relational comparison. Calculus: Tuple relational calculus, Domain relational Calculus, calculus vs algebra, computational capabilities.

Unit 5

What is constraints, types of constrains, Integrity constraints, Views: Introduction to views, data independence, security, updates on views, comparison between tables and views SQL: data definition, aggregate function, Null Values, nested sub queries, Joined relations. Triggers. Transaction management: ACID properties, serializability and concurrency control, Lock based concurrency control (2PL, Deadlocks),Time stamping methods, optimistic methods, database recovery management.

References

- A Silberschatz, H Korth, S Sudarshan, “Database System and Concepts”, fifth Edition McGraw-Hill
- An introduction to Database System – Bipin Desai, Galgotia Publications
- Database System: concept, Design & Application by S.K.Singh (Pearson Education)
- Database management system by leon &leon (Vikas publishing House).
- Database Modeling and Design: Logical Design by Toby J. Teorey, Sam S. Lightstone, and Tom Nadeau, “”, 4th Edition, 2005, Elsevier India Publications, New Delhi
- Fundamentals of Database Management System – Gillenson, Wiley India
- Rob, Coronel, “Database Systems”, Seventh Edition, Cengage Learning.

Contents

* * *

Chapter 1

FILE STRUCTURE AND ORGANIZATION

1. Introduction

Database is an organized collection of interrelated data. Databases are stored physically as files of records are typically stored on magnetic disks. Magnetic disk is one of the storage medium in computers.

In computers, a storage medium is any technology (including devices and materials) used to place, keep, and retrieve data. A medium is an element used in communicating a message on a storage medium, the "messages" - in the form of data - are suspended for use when needed. The plural form of this term is storage media. Although the term storage includes both primary storage (memory), a storage medium usually means a place to hold secondary storage such as that on a hard disk or tape.

In practice, almost all computers use a variety of memory types, organized in a storage hierarchy around the CPU, as a tradeoff between performance and cost. Generally, the lower a storage is in the hierarchy, the lesser its bandwidth and the greater its access latency is from the CPU. This traditional division of storage to primary, secondary, tertiary and off-line storage is also guided by cost per bit. *Figure 1.1* shows the various forms of storage areas.

1.1 Primary Storage

Primary Storage presently known as memory is the only one directly accessible to the CPU. The CPU continuously reads instructions stored there and executes them. Any data actively operated on is also stored there in uniform manner. As shown in the *figure 1.1,* traditionally there are two more sub-layers of the primary storage, besides main large-capacity RAM:

i. Processor registers are located inside the processor. Each register typically holds a word of data (often 32 or 64 bits). CPU instructions instruct the arithmetic and logic unit to perform various calculations or other operations on this data (or with the help of it). Registers are technically among the fastest of all forms of computer data storage, being switching transistors integrated on the CPU's chip, and functioning as electronic "flip-flops".

ii. Processor cache is an intermediate stage between ultra-fast registers and much slower main memory. It is introduced solely to increase performance of the computer. Most actively used information in the main memory is just duplicated in the cache memory, which is faster, but of much lesser capacity. On the other hand, it is much slower, but much larger than processor registers. Multi-level hierarchical cache setup is also commonly used—primary cache being smallest, fastest and located inside the processor secondary cache being somewhat larger and slower.

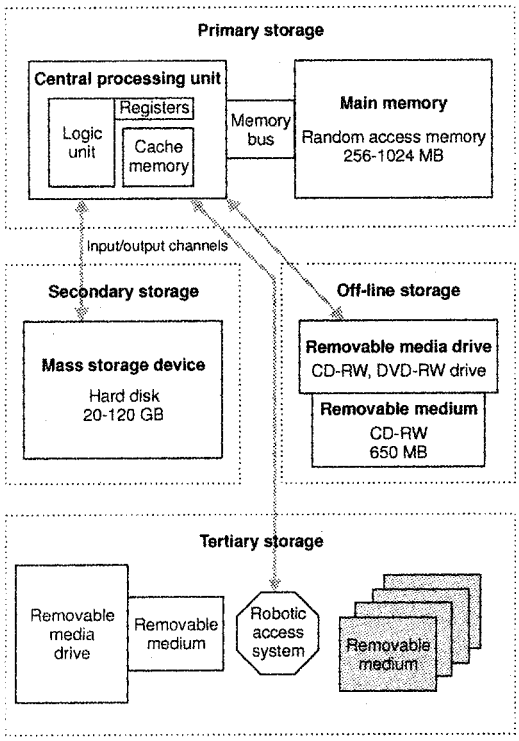


Figure 1.1: Various types of storage areas

As the RAM types used for primary storage are volatile (cleared at start up), a computer containing only such storage would not have a source to read instructions from, in order to start the computer. Hence, non-volatile primary storage containing a small startup program (BIOS) is used to bootstrap the computer, that is, to read a larger program from non-volatile secondary storage to RAM and start to execute it. A non-volatile technology used for this purpose is called ROM, for read-only memory (the terminology may be somewhat confusing as most ROM types are also capable of random access).

1.2 Secondary Storage

Secondary storage or storage in popular usage, differs from primary storage in that it is not directly accessible by the CPU.

The computer usually uses its input/output channels to access secondary storage and transfers desired data using intermediate area in primary storage. Secondary storage does not lose the data when the device is powered down—it is non-volatile. Per unit, it is typically also an order of magnitude less expensive than primary storage.

Consequently, modern computer systems typically have an order of magnitude more secondary storage than primary storage and data is kept for a longer time there.

In modern computers, hard disks are usually used as secondary storage. The time taken to access a given byte of information stored on a hard disk is typically a few thousands of a second, or milliseconds. By contrast, the time taken to access a given byte of information stored in random access memory is measured in thousand-millionths of a second, or nanoseconds.

This illustrates the very significant access-time difference which distinguishes solid-state memory from rotating magnetic storage devices: hard disks are typically about a million times slower than memory. Rotating optical storage devices, such as CD and DVD drives, have even longer access times.

Knowing how the data is likely organized can help in the construction of efficient database access applications.

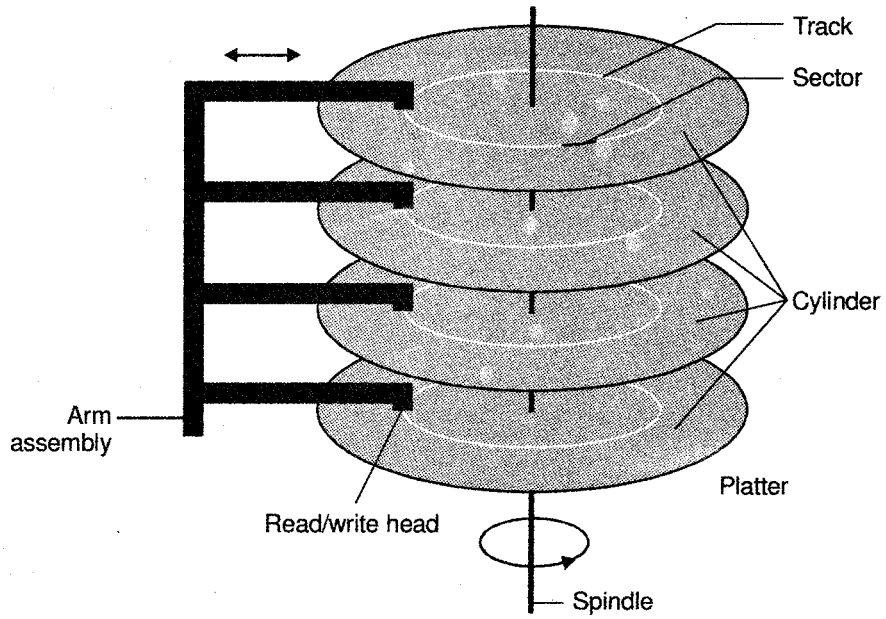


Figure 1.2 : Physical disk structure

Figure 1.2 shows a disk structure capable of storing large quantities of data cheaply.

- Megabytes (Mb) -- million (10^6 or 2^{20}) -- typical databases
- Gigabytes (Gb) -- billion (10^9 or 2^{30}) -- large databases
- Terabytes (Tb) -- trillion (10^{12} or 2^{40}) -- very large databases
- Petabytes (Pb) -- quadrillion (10^{15} or 2^{50})
- Exabytes (Xb) -- quintillion (10^{18} or 2^{60})

It is a non-volatile, i.e., memory is retained when power is removed. The hard disk is extremely slow compared with CPU speeds.

- Disk speeds are still rated in the milli-second range (thousandths).
- CPU speeds are rated in the gigaHertz or nano-second range (billionths).
- Million fold difference-- the CPU can carry out a million operations to each disk access.

Performance of a DBMS is largely a function of the number of disk I/O operations that must be performed. Buffering and disk caches can be used to speed up the access.

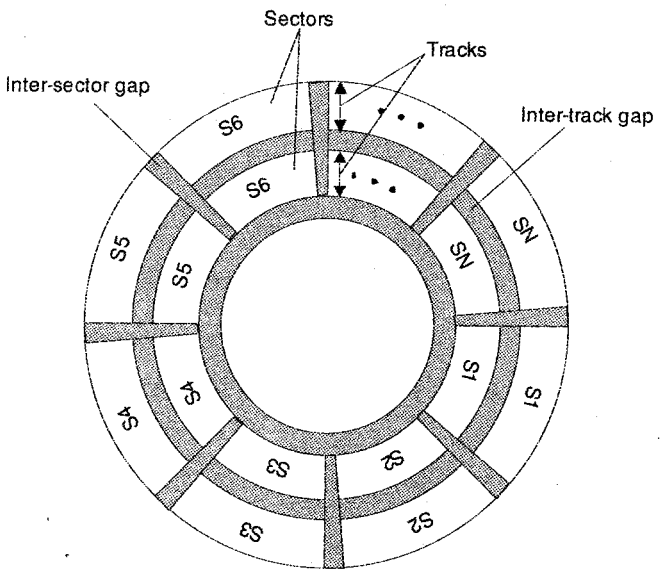


Figure 1.3: Disk data layout

▶ Data Files are Decomposed Into Pages

These are fixed size pieces of contiguous information in the file (usually multiples of 512, 1024, or 2048 bytes)

- Page is the unit of exchange between disk and main memory.

Disk are divided into page size blocks of storage.

- A page can be stored in any disk block.

An application's request for read item is satisfied by:

1. Read entire page containing item to buffer in DB (disk input/output operation in msec).
2. Transfer item from buffer to application (primary memory operation in nsec).

An application's request to change item is satisfied by

1. Read page containing item to buffer in DBMS (if it is not already there).
2. Update item in DBMS buffer (nsec).
3. (Eventually) copy buffer page to page on disk (msec).

Here the term "block" means a unit of storage in RAM, on disk. To allocate space on disk for fast access buffer pool management is done.

Buffer pool management means creating frames in RAM to hold blocks. One needs to decide the policy to move blocks between RAM & disk.

Databases are maintained to store the information for future retrieval, however care should be taken to see it that the retrieval is not time consuming and therefore one needs to understand the I/O time to access a page which can be as follows:

Seek Latency

Time to position heads over **cylinder** containing page (avg = ~10 - 20 ms)

- This mechanical motion is extremely hard to make faster.

Rotational Latency

Additional time for platters to rotate so that start of block containing page is under head (avg = ~5 - 10 ms). This depends on the rotational speed of the disk.

- 1/2 rotation is the average rotational latency.
- 7200 rpm = 120 rev/sec
- 1/120 sec/rev * 1/2 = 1/240 = 4.17 msec

Transfer Time

Time for platter to rotate block containing page (depends on size of block and rotational speed of the disk) over the read/write heads.

Latencies = Seek latency + Rotational latency

Total access time = Latencies + Transfer time

Goal - minimize average latency and reduce number of page transfers

Reducing Latency

For reducing the latency the following strategy and page trade off must be observed:

Strategy: Store pages containing related information close together on disk.

- **Justification**: If application accesses x, it will likely next access data related to x with high probability.
- "locality principle"

- Often this data might occur in another table--the data of a table are not necessarily kept to the locality principle.
- When a disk block is input, then related information may already be in the buffer as well.

Page size tradeoff

- Large page size - data related to x stored in same page; hence additional page transfer can be avoided.
- Small page size - reduce transfer time, reduce buffer size in main memory.
- Typical page size - 4096 bytes.

Some other examples of secondary storage technologies are: flash memory (*for example:* USB sticks or keys), floppy disks, magnetic tape, paper tape, punch cards, standalone RAM disks, and zip drives.

1.3 Tertiary Storage

Tertiary storage tertiary memory, provides a third level of storage. Typically it involves a robotic mechanism which will mount (insert) and dismount removable mass storage media into a storage device according to the system's demands; this data is often copied to secondary storage before use.

It is primarily used for archival of rarely accessed information since it is much slower than secondary storage (*for example* 5-60 seconds vs. 1-10 milliseconds). This is primarily useful for extraordinarily large data stores, accessed without human operators. Typical examples include tape libraries and optical jukeboxes.

1.4 Off-line Storage

Off-line storage also known as disconnected storage, is computer data storage on a medium or a device that is not under the control of a processing unit. The medium is recorded, usually in a secondary or tertiary storage device, and then physically removed or disconnected. It must be inserted or connected by a human operator before a computer can access it again. Unlike tertiary storage, it cannot be accessed without human interaction.

Off-line storage is used to transfer information, since the detached medium can be easily physically transported.

Additionally in case a disaster, *for example* a fire, destroys the original data, a medium in a remote location will be probably unaffected, enabling disaster recovery. Off-line storage increases a general information security, since it is physically inaccessible from a computer, and data confidentiality or integrity cannot be affected by computer-based attack techniques.

Also, if the information stored for archival purposes is accessed seldom or never, off-line storage is less expensive than tertiary storage.

2. Logical and Physical Files

In data processing, using an office metaphor, a file is a related to collection of records. *For example*, you might put the records you have on each of your customers in a file. In turn, each record would consist of fields for individual data items, such as customer name, customer number, customer address, and so forth. By providing the same information in the same fields in each record (so that all records are consistent), your file will be easily accessible for analysis and manipulation by a computer program.

This use of the term has become somewhat less important with the advent of the database and its emphasis on the table as a way of collecting record and field data. In mainframe systems, the term data set is generally synonymous with file but implies a specific form of organization recognized by a particular access method. Depending on the operating system, files (and data sets) are contained within a catalog, directory, or folder.

In any computer system but especially in personal computers, a file is an entity of data available to system users (including the system itself and its application programs) that is capable of being manipulated as an entity (*for example*, moved from one file directory to another). The file must have a unique name within its own directory. Some operating systems and applications describe files with given formats by giving them a particular file name suffix. (The file name suffix is also known as a file name extension.)

For example, a program or executable file is sometimes given or required to have an ".exe" suffix. In general, the suffixes tend to be as descriptive of the formats as they can be within the limits of the number of characters allowed for suffixes by the operating system.

Thus a ***file*** is a sequence of records. A ***record*** is a subdivision of a file, containing data related to a single entity. And a ***field*** in a record is a subdivision of a record containing a single attribute of the entity which the record describes.

Usually all records in a file are of the same record type (Fixed-length records). In general, a block contains one or more records specific to one file only:

- Spanned organization: Records can cross block boundaries.
- Unspanned organization: Records can't cross block boundaries.

From a users perspective a file can be a physical file or a logical file.

Physical File: A collection of bytes stored on a disk or tape.

Logical File: A "Channel" (like a telephone line) that hides the details of the file's location and physical format to the program.

When a program wants to use a particular file, "data", the operating system must find the physical file called "data" and make the hookup by assigning a logical file to it. This logical file has a logical name which is what is used inside the program.

3. Basic File Operations

Various file operations are as follows:
i. Opening file
ii. Closing files
iii. Reading and writing files
iv. Seeking from files

The different operation that are performed on a file, after a connection is obtained are opening, closing, reading writing and seeking we will see each of these operations one by one, in next few sections.

Asked. Number of questions 3

PU

Oct.2012 – *4M*
Explain any four file operations.

Apr.12, Oct. 11 – *4M*
List various file operations. Explain any three.

3.1 Opening Files

Once, we have a logical file identifier hooked up to a physical file or device, we need to declare the operations that we intend to do with the file.

In general there are two basic operations:

i. Open an existing file or
ii. Create a new file, deleting any existing contents in the file.

Opening a file makes it ready for use within the application program. The file pointer is placed at the beginning of the file and thus ready to start reading or writing. Creating a file also opens the file, in the sense that its ready for use after creation.

3.2 Closing Files

Closing a file is like terminating a phone call or hanging up the phone. When a file is closed the logical name or file descriptor is available for use with another file.

Closing a file that has been used for output also ensures that everything has been written to the file.

Data to and from secondary storage is always moved in terms of blocks rather than one byte at a time.

Thus, the operating system puts the byte written into a buffer rather than send them to the see storage. The buffer is then transferred as a block of data. Closing a file ensures that the buffer for that file has been flushed of data and that everything written has been sent to the file.

Whenever an application program terminates normally, the files used by it are automatically closed by the operating system.

A close statement is needed within a program, only to ensure that there is no accidental loss of data in case of the program is interrupted also to free up logical filenames for reuse.

3.3 Reading and Writing

Reading and writing are fundamental to file processing; they are the actions that make file processing an input/ output operation. Here we consider read and write at a relatively low level.

A low level read call requires three arguments or three pieces of information as given below.

```
Read (source_file, Destination_addr, size)
```

In the above read function,

Source_file →Specifies from where it should read from, i.e. the logical file name

Destination_addr → The address of the memory block, where the data read is to be stored.

Size →The amount of information to be read, given as a byte count.

The write function can be given as;

```
write (Destination_file, source_addr, size)
```

In the above the

Destination file → logical file name that is used for sending the data.

Source_addr → First address of the memory block where the data is stored.

Size → The amount of information to be written in number of bytes.

3.4 Seeking from a File

Seek is used to directly move to a particular position in a file. i.e. when we may want to read/ write directly from a particular position in a file in order to have the above, we should be able to control the read/ write pointer, so that we can directly read from any position or write to any position.

Thus, the action of moving directly to a certain position in a file is often called seeking.

A seek has minimum two arguments, as given below:

```
seek (source_file, offset)
```

In the above format

Source_file → The logical file name in which the seek will occur.

Offset→The number of position in the file, the pointer is to be moved from the start of the file.

4. File Organization

A **file** is organized logically as a sequence of records. Records are mapped onto disk blocks. Files are provided as a basic construct in operating systems, so we assume the existence of an underlying **file system**. Blocks are of a fixed size determined by the operating system. Record sizes vary. In relational database, tuples of distinct relations may be of different sizes.

One approach to mapping database to files is to store records of one length in a given file. An alternative is to structure files to accommodate variable-length records. (Fixed-length is easier to implement.)

4.1 Fixed Length Record

A record which is predetermined to be the same length as the other records in the file.

Record 1	Record 2	Record 3	Record 4	Record 5

- The file is divided into records of equal size.
- All records within a file have the same size.
- Different files can have different length records.

- Programs which access the file must know the record length.
- Offset, or position, of the nth record of a file can be calculated.
- There is no external overhead for record separation.
- There may be internal fragmentation (unused space within records.)
- There will be no external fragmentation (unused space outside of records) except for deleted records.
- Individual records can always be updated in place.

Consider a file of *deposit* of the form:

```
bname: char (22); account# : char(10); balance : real;
```

If we assume that each character occupies one byte, an integer occupies 4 bytes, and a real 8 bytes, our deposit record is 40 bytes long.

The simplest approach is to use the first 40 bytes for the first record, the next 40 bytes for the second, and so on. However, there are two problems with this approach.

1. It is difficult to delete a record from this structure. Space occupied must somehow be deleted, or we need to mark deleted records so that they can be ignored.
2. Unless block size is a multiple of 40, some records will cross block boundaries. It would then require two block accesses to read or write such a record.

When a record is deleted, we could move all successive records up one, which may require moving a lot of records. We could instead move the last record into the "hole" created by the deleted record.

This changes the order the records are in. It turns out to be undesirable to move records to occupy freed space, as moving requires block accesses. Also, insertions tend to be more frequent than deletions. It is acceptable to leave the space open and wait for a subsequent insertion. This leads to a need for additional structure in our file design.

So one solution is:

1. At the beginning of a file, allocate some bytes as a file header.
2. This header for now need only be used to store the address of the first record whose contents are deleted.
3. This first record can then store the address of the second available record, and so on.
4. To insert a new record, we use the record pointed to by the header, and change the header pointer to the next available record.
5. If no deleted records exist we add our new record to the end of the file.

Note: Use of pointers requires careful programming. If a record pointed to is moved or deleted, and that pointer is not corrected, the pointer becomes a dangling pointer. Records pointed to are called pinned.

Fixed-length file insertions and deletions are relatively simple because "one size fits all". For variable length, this is not the case.

4.2 Variable Length Record

A record which can differ in length from the other records of the file.

Delimited record

A variable length record which is terminated by a special character or sequence of characters.

▶ Delimiter

A special character or group of characters stored after a field or record, which indicates the end of the preceding unit.

Record 1	#	Record 2	#	Record 3	#	Record 4	#	Record 5	#

- The records within a file are followed by a delimiting byte or series of bytes.
- The delimiter cannot occur within the records.
- Records within a file can have different sizes.
- Different files can have different length records.
- Programs which access the file must know the delimiter.
- Offset, or position, of the nth record of a file cannot be calculated.
- There is external overhead for record separation equal to the size of the delimiter per record.
- There should be no internal fragmentation (unused space within records.)
- There may be no external fragmentation (unused space outside of records) after file updating.
- Individual records cannot always be updated in place.

Variable-length records: some possible schemes:

- The file records are of the same record type but one or more of the fields are of varying size.
- The file records are of the same record type but one or more of the fields may have multiple values for the individual records.
- The file records are of the same record type, but one or more of the fields are optional.
- The file include records of different types, each record will be preceded by a record type indication: if a relation exists between EMPLOYEE and DEPARTMENT, then their corresponding records are physically contiguous (clustered) in order to minimize I/O operations.

Asked. Number of questions 3

PU
Apr.12, Oct.12, 11– *4M*

Differentiate Fixed Length and Variable Length Records

Fixed length record	Variable length record
A file where all the records are of the same length is said to have fixed length records.	One or more of the fields can be of differing lengths in each record, called variable length records.
Advantage: Access is fast because the computer knows where each record starts. e.g., if each record is 120 bytes long then • The 1st record starts at [Start of File] + 0 bytes. • The 2nd record starts at [Start of File] + 120 bytes • The 3rd record starts at [Start of File] + 240 bytes etc.....	**Advantage:** • The records will be smaller and will need less storage space.
Disadvantage: Using fixed length records, the records are usually larger and therefore need more storage space and are slower to transfer (load or save).	**Disadvantage:** The computer will be unable to determine where each record starts so processing the records will be slower.
In fixed length, user cannot declare the record with his/her convenience.	Variable length is nothing but user can declare record with his/her convenience.
Fixed length is mentioned in static variable creation.	Variable length is changed in dynamic variable creation.
For fixed length files, we have to code actual length.	We have to code actual length + 4 bytes for variable length files.
When we declare array with its size it is called as fixed length.	When we declare array with pointer it is called variable length record.

Example in c
`struct book`
`{`
`  int bno;`
`  char bookname[10];`
`};`
`book b1[10];`
b1 is array of 10 records.
	Example
	`struct course`
	`{`
	`  int courseno;`
	`  char coursename[10];`
	`}*ptr;`
	ptr is pointer to structure course.

5. Types of File Organization

Many alternatives for file organization exist, each ideal for some situation, and not so good in others:

- **Heap Files:** Suitable when typical access is a file scan retrieving all records.
- **Sorted Files:** Best if records must be retrieved in some order, or only a `range' of records is needed.
- **Hashed Files:** Good for equality selections.

Three Types of File Organization
i. Heap files
ii. Sorted files
iii. Hashed file

5.1 Files of Unordered Records (Heap Files)

Asked. Number of questions asked 1
PU
Oct.2012 – *4M*
Describe Heap file Organization.

An unordered file, sometimes called a heap file, is the simplest type of file organization.

Records are placed in file in the same order as they are inserted. A new record is inserted in the last page of the file; if there is insufficient space in the last page, a new page is added to the file.

This makes insertion very efficient. However, as a heap file has no particular ordering with respect to field values, a linear search must be performed to access a record. A linear search involves reading pages from the file until the required is found. This makes retrievals from heap files that have more than a few pages relatively slow, unless the retrieval involves a large proportion of the records in the file.

To delete a record, the required page first has to be retrieved, the record marked as deleted, and the page written back to disk. The space with deleted records is not reused. Consequently, performance progressively deteriorates as deletion occurs. This means that heap files have to be periodically reorganized by the Database Administrator (DBA) to reclaim the unused space of deleted records.

Heap files are one of the best organizations for bulk loading data into a table, as records are inserted at the end of the sequence; there is no overhead of calculating what page the record should go on.

5.2 File of Ordered Records (Sorted Files)

Asked. Number of questions 1

PU

Apr.2011 – *4M*

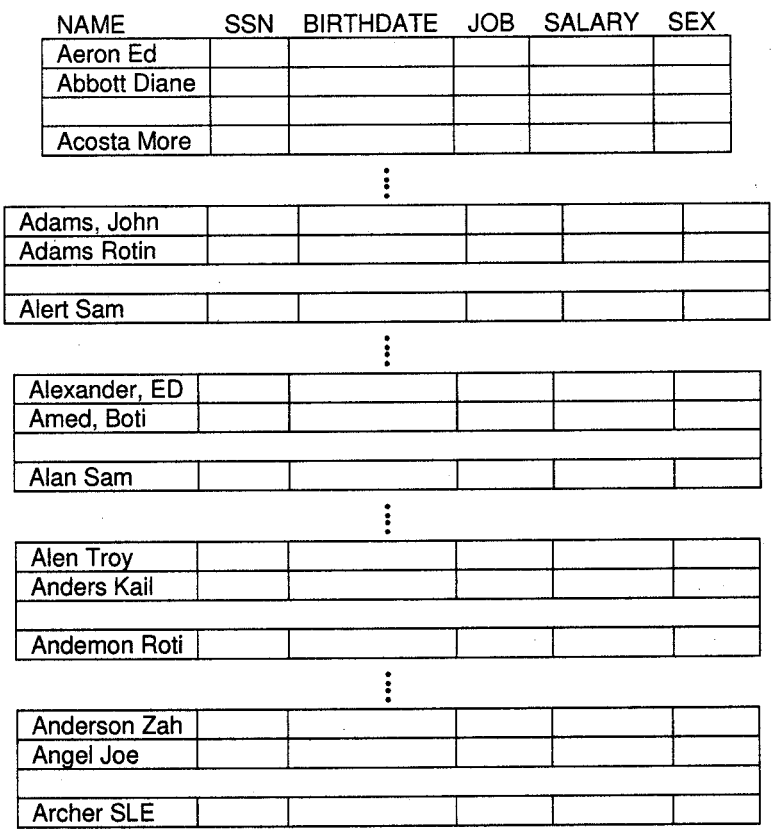
Explain Sorted File Organization Technique in detail.

Organization that physically order the records of a file on disk based on the values of one of the fields called the ordering field.

If the ordering field is also a key field of the file then the field is called the ordering key for the file. *Figure 1.4* shows an ordered file with NAME as the ordering key field (assuming that employees have distinct names). Reading the records in order of the ordering key values becomes extremely efficient, because no sorting is required.

Using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used.

Ordering does not provide any advantage for random or ordered access of the records based on values for the other non-ordering fields of the file. In this case, do a linear search for random access.



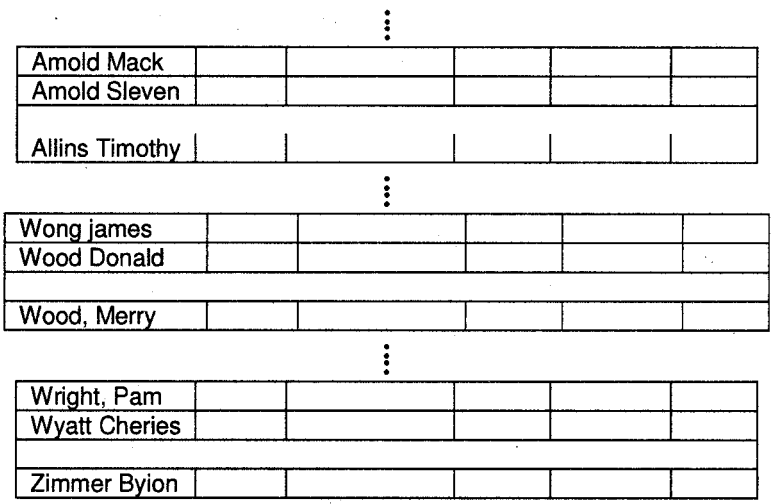


Figure 1.4 : Sorted file

5.3 Hash Files

Asked. Number of questions 2
PU
Apr.12, Oct. 11 – *4M*
Explain Hashed File Organization Technique in detail.

Provides very fast access to records on certain search conditions. The search condition must be an equality condition on a hash field of the file. In most cases, the hash field is also a key field of the file (hash key).

▶ Hashing

To provide a function h, called a hash function, that is applied to the hash field value of a record and yields the address of the disk block in which the record is stored. A search for the record within the block can be carried out in a main memory buffer.

▶ Internal Hashing

Hashing is also used as an internal search structure within a program whenever a group of records is accessed exclusively by using the value of one field. Hashing is implemented as a hash table through the use of an array of records. Suppose that the array index range is from 0 to N-1; then we have N 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 N-1. One common hash function is the h(K) = K mod N function, this value is used for the record address.

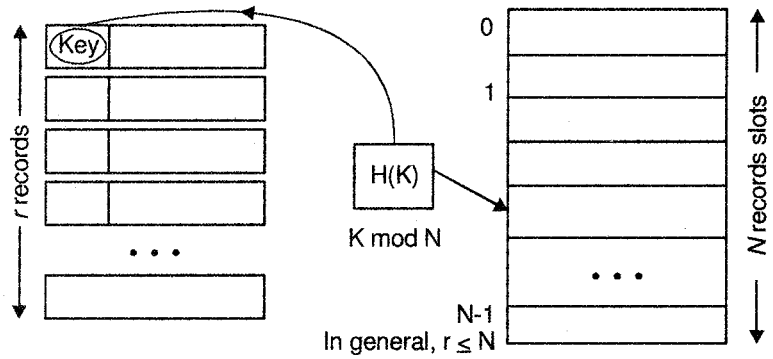


Figure 1.5 : Internal Hashing

For example, consider a key,student id (six digits). Assume we have N = 100,000 record slots numbered 00000 – 99999.

Then the hash function H(K): student_id mod 100000

- 085768 → 085768 mod 100000 = 85768
- 134281 →134281 mod 100000 = 34281
- 101004 → 101004 mod 100000 = 1004
- 100000 → 100000 mod 100000 = 0
- 601004 → 601004 mod 100000 = 1004 (collision)

▶ Collision

A collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. The process of finding another position (after collision) is called collision resolution.

Methods for collision resolution: Open addressing – Chaining – Multiple hashing

▶ External Hashing

Hashing for disk files is called external hashing. The target address space is made of buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of contiguous blocks. The hashing function maps a the indexing field's value into a relative bucket number.A table maintained in the file header converts the bucket number into the corresponding disk block address.

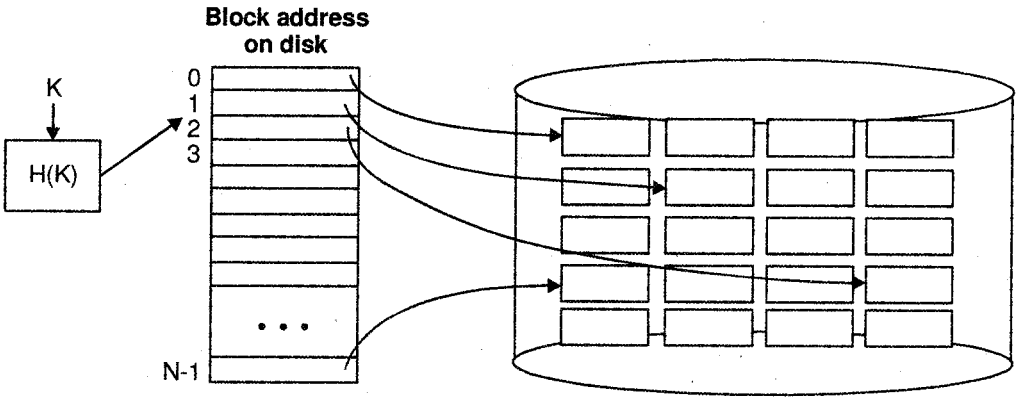


Figure 1.6 : External Hashing

Hash file with relative bucket numbers 0 through N-1 can be seen in *figure 1.6*. Bucket number to absolute disk block addresses are mapped. The disk blocks are the buckets that hold several data records each.

6. Overview of Indexes

Index searching techniques were invented early on by computer programmers as a means of reducing disk I/O, and they were incorporated into several proprietary file systems prior to the advent of relational databases.

The classic analogy to help you understand database indexes is the index in the back of reference books. Sure, if you wanted to find everything in the book about a particular subject you could start at the beginning and scan every page, but it is much faster to look in a smaller, alphabetized subject index that directs you to a list of pages. Then you need to scan only those pages to find information about your chosen subject. Not everything in the book is indexed, however, so if your subject is not mentioned in the index, you must still scan for it. Likewise, a database index is a look-up mechanism that helps a DBMS find the information you request faster than it could with a full scan. As with book indexes, not everything in the database is indexed, so an occasional scan may still be necessary.

The primary reason to build an index is to improve performance. But it is not the only reason to build an index. The second reason has to do with enforcing uniqueness among rows stored in a database table. Tables in a SQL database are usually designed with a primary key; that is, a set of columns with a unique value that identifies a row in the table. When a new row is inserted into a table defined with a primary key, it is up to the DBMS to ensure that the primary key value for that row is unique. Performance would be unacceptable if the DBMS had to scan the entire table each time a new row

was inserted. Therefore, the accepted solution is to build a unique index on the primary-key columns and let the DBMS use that as the physical enforcement mechanism for the primary key uniqueness requirement. Some SQL products (such as Oracle7) implicitly create a unique index when you define a primary key during table creation, while others (such as SQLBase) require you to create a unique index before using a table created with a primary key.

The best things in file may be free, but indexes have their price: increased overhead for index maintenance during the processing of insert, update, and delete statements. Not only will changes to the database be slower, but if every column in a database is indexed, query performance might suffer because index maintenance could consume the bulk of the CPU cycles. Plus, you might have locking conflicts. Thus, the challenge is to find a happy medium between too few indexes and too many indexes.

One way to meet this challenge is to look for situations in which indexes should be avoided. First, you should avoid indexes on small tables. If a few disk reads is all it takes to scan an entire table, then an index can actually slow things down because it requires an extra disk read or two. What qualifies as a small table? The size limit varies depending on the hardware, operating system, DBMS, and row size, but any table under 100 rows is a candidate. If there is any question in your mind about whether to create an index on a small table, test the performance with and without the index.

Second, you should avoid indexes on tables that will always be accessed by scans (such as transaction tables in which each row is fetched by a cursor, or statistical tables that summarize with aggregate functions). In these cases, an index would be redundant.

Third, you should avoid placing indexes on columns that have few distinct values. The best candidates for indexing are columns with unique values; whereas the worst candidates are columns with only a few possible values (the classic example is a column named "sex," which can have the values "F" or "M"). It's more efficient for a DBMS to scan the entire table than to use an index to find 50 percent of the rows. (I've even seen someone index a column with one distinct value, but that's another story.)

It's worth noting that binary large object (BLOB) columns (also known as "long" or "raw" columns) typically cannot be indexed. And for good reason: An index on a BLOB column would take up almost as much disk space as the table. Therefore, performance would suffer.

So when should you create indexes? This is an easy decision to make. If a table will be populated by interactive data entry, you should create indexes on the table immediately after you create the table. On the other hand, if the table will be populated by bulk loading of data from a file or from another table, you should create the indexes after the data is loaded. The rationale for delaying index creation in the latter case is that it is much more efficient to build an index all at once instead of updating the index every time a row is added. With interactive data entry, the index maintenance inefficiency is generally transparent to the user, but with bulk loading it is very noticeable.

In its simplest form, the create index command looks like the following:

CREATE INDEX sales_1996_idx ON sales_1996 (customer, product);

In this *example*, the name of the index is "sales_1996_idx," the name of the table being indexed is "sales_1996," and the columns that make up the index are "customer" and "product."

▶ Index Structure and Access

1. The top level of an index is usually held in memory. It is read once from disk at the start of queries.
2. Each index entry points to either another level of the index, a data record, or a block of data records.
3. The top level of the index is searched to find the range within which the desired record lies.
4. The appropriate part of the next level is read into memory from disc and searched.
5. This continues until the required data is found.
6. The use of indices reduce the amount of file which has to be searched.

▶ Costing Index and File Access

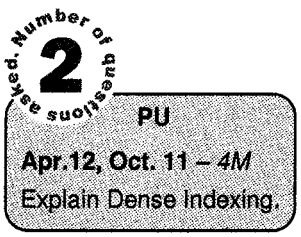
1. The major cost of accessing an index is associated with reading in each of the intermediate levels of the index from a disk (milliseconds).
2. Searching the index once it is in memory is comparatively inexpensive (microseconds).
3. The major cost of accessing data records involves waiting for the media to recover the required blocks (milliseconds).
4. Some indexes mix the index blocks with the data blocks, which means that disk accesses can be saved because the final level of the index is read into memory with the associated data records.

The most common physical storage structure for SQL indexes is the B-tree. Almost every SQL DBMS on the market supports B-tree indexes. Plus, some DBMSs support additional physical index structures such as hashing and Index Sequential Access Method (ISAM). B-tree indexes are popular because of their adaptability (the tree structure balances itself dynamically as a table grows, which maintains an efficient index structure by minimizing the number of disk reads to find a given value in the index).

6.1 Types of Indexes

- Indexes on Ordered vs. Unordered files
- Dense vs. Non-dense (i.e., sparse) indexes
  - **Dense:** An entry in the index files for each record of the data file.
  - **Sparse:** Only some of the data records are represented in the index, often one index entry per block of the data file.
- Primary Indexes vs. Secondary Indexes
  - **Primary index**: It is an index whose search key defines the sequential order of the file.
  - **Secondary index**: Indices whose search key specifies an order different from sequential order of the file.
- Ordered Indexes vs Hash Indexes
  - **Ordered Indexes:** Indexing fields stored in sorted order.
  - **Hash Indexes:** Indexing fields stored using a hash function.
- Single-level vs. Multi-level
  - **Single-level** index is an ordered file and is searched using binary search.
  - **Multi-level** ones are tree-structured that improve the search and require a more elaborate search algorithm.
- Index on a single indexing field – Index on multiple indexing fields (i.e., Composite Index).
  - If a certain combination of fields is used frequently, set an index on multiple fields.

▶ Dense and Sparse Indices



An index record, or index entry, consists of a search key value and pointers to one or more records with that value as their search key value. The pointer to a record consists of the identifier of a disk block and an offset within the disk block to identify the record within the block.

There are two types of ordered indices:

Dense Index

An index record appears for every search-key value in the file. In a dense clustering index, the index record contains the search-key value and a pointer to the first data record with that search-key value.

The rest of the records with the same search-key value would be stored sequentially after the first record, since because the index is a clustering one, records are sorted on the same search key. Dense index implementations may store a list of pointers to all records with the same search-key value; doing so is not essential for clustering indices.

Asked Number of questions 2

PU

Oct.2012 – *4M*
Define Index. Explain Sparse Index.

Apr.2011 – *4M*
Explain Sparse Index with example.

▶ Sparse Index

- Index records are created only for some of the records.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along the pointers in the file (that is, sequentially) until we find the desired record.

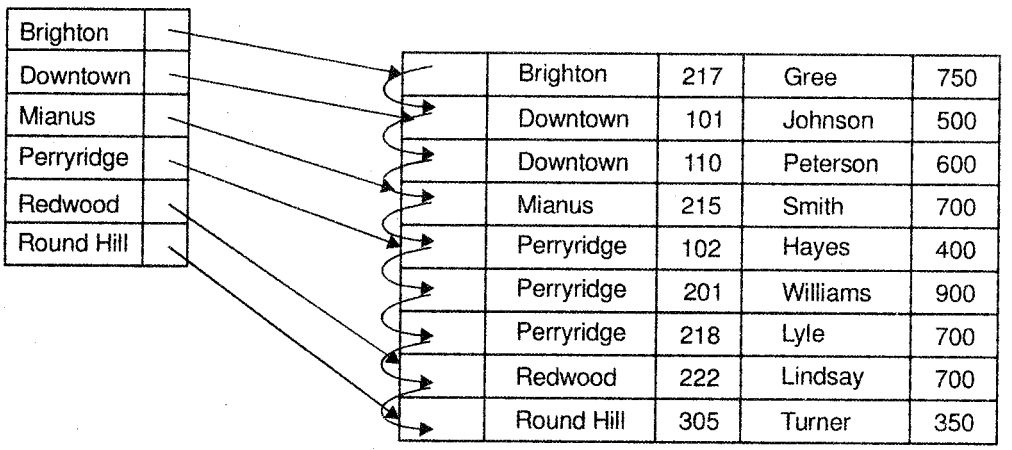


Figure 1.7: Dense index

Figures 1.7 and *1.8* show example for dense and sparse indices for account file. Suppose that we are looking up for the Perryridge branch. Using the dense index of *figure 1.7*, we follow the pointer directly to the first Perryridge record. We process this record, and follow the pointer in that record to locate the next record in the search key (branch_name) order.

We continue processing records until we encounter a record for a branch other than Perryridge. If we are using the sparse index as in *figure 1.8* , we do not find an index entry for "Perryridge" . Since the last entry before Perryridge is Mianus, we follow that pointer. We then read the account file in sequential order until we find the first Perryridge record, and begin processing at that point.

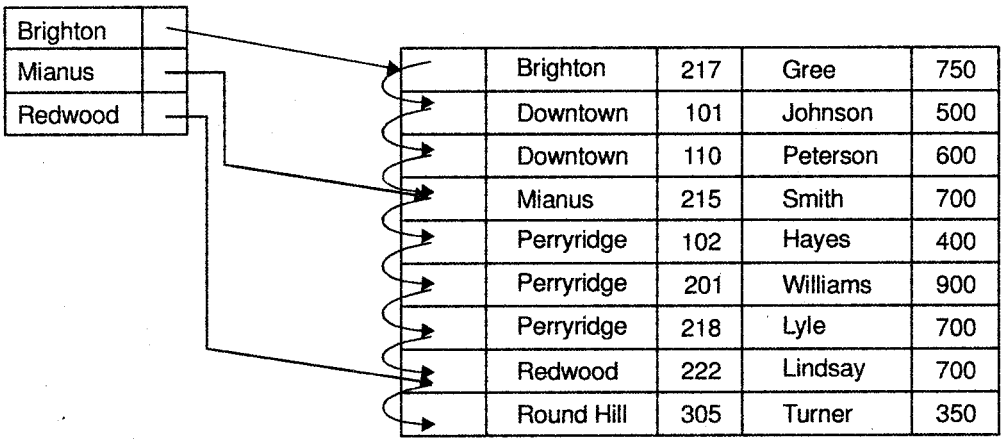


Figure 1.8: Sparse index

Dense indices are faster in general, but sparse indices require less space and impose less maintenance for insertions and deletions.

There is a trade off that a system designer must make between access time and space overhead. Although the decision regarding this trade off depends on the specific application, a good compromise is to have a sparse index with one index entry per block.

▶ Why is this good?

- Biggest cost is in bringing a block into main memory.
- We are guaranteed to have the correct block with this method, unless record is on an overflow block (actually could be **several** blocks).
- Index size still small.

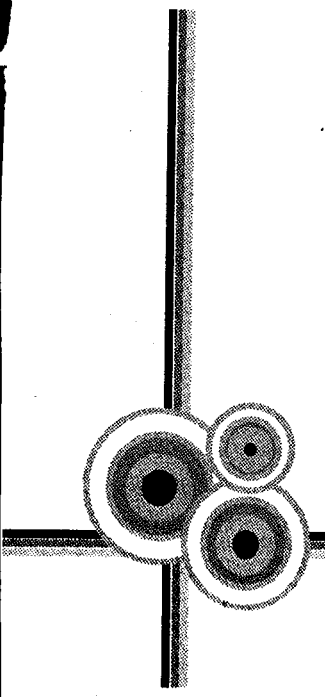
Summary

- In computer system there are several types of data storage. They are classified by the speed with which they can access data, by their cost per unit of data to buy the memory, and by their reliability. Among the media available are cache, main memory, flash memory, magnetic disk, optical disk and magnetic tapes.
- We can organize a file as a sequence of records mapped onto disk blocks. One approach to mapping the database files is to use several files, and to store records of only one fixed length in any given file. An alternative is to structure files so that they can accommodate multiple lengths for records.
- Many alternatives for file organization exist, each ideal for some situation, and not so good in others: Heap files which are suitable when typical access is a file scan retrieving all records. Sorted Files, best if records must be retrieved in some order, or only a 'range' of records is needed. Hashed Files, good for equality selections.
- Index sequential files are one of the oldest index schemes used in database system. To permit fast retrieval of records in search key order, records are stored sequentially, and out of order records are chained together. To allow fast random access, we use an index structure.
- There are two types of indices that we can use, i.e., dense indices and sparse indices. Dense indices contain entries for every search key value, whereas sparse indices contain entries only for some search key values.

PU Questions

[Oct.2012 – 4M]	1.	Explain any four file Operations.
[Oct.2012 – 4M]	2.	Describe Heap File Organization.
[Oct.2012 – 4M]	3.	Define Index. Explain Sparse Index.
[Apr.12,Oct.11 – 4M]	4.	Explain Hashed File Organization Technique in detail.
[Apr.12,Oct.11 – 4M]	5.	Explain Dense Indexing.
[Apr.12,Oct.11– 4M]	6.	List various file operations. Explain any three.
[Apr.12,Oct.12,11– 4M]	7.	Differentiate between Fixed Length and Variable Length Records.
[Apr.2011 – 4M]	8.	Explain Sorted File Organization Technique in detail.
[Apr.11, Oct.09 – 4M]	9.	Explain Sparse Index with example.
[Oct.10, Oct.09 – 4M]	10.	Write a short note on heap file.
[Oct.10, 09 – 4M]	11.	What are the different file organization techniques? Explain any one technique in detail.
[Oct.10,09 – 4M]	12.	Explain Basic File Operations.
[Apr.10,09 – 4M]	13.	List types of File Organisation Techniques. Explain any one in detail.
[Apr.2010 – 4M]	14.	What do you mean by Physical Files and Logical Files?
[Oct.09,Apr.09 – 4M]	15.	Explain Fixed Length and Variable Length Record with example.
[Apr.2009 – 4M]	16.	Explain any four File Operations.

VISION



Chapter 2

DATABASE MANAGEMENT SYSTEM

1. Introduction

A need to store huge amount of data in a database for future retrieval is an obligatory requirement of any organization in this era of information technology. An organization may be a single venture such as a firm with all its units located at a single campus governed by only one board of Directors or it may consists of number of units which could be considered a separate organization.

Typically an organization needs to collect, process, store and distribute data for its human, financial and material resources and functions. The functions may include: admission process of a college or university, maintaining stock record and performing inventory control, sales report and forecast, etc. All this information can be stored into the database and effectively managed by an efficient Database Management System (DBMS). Thus a Database management system is a software system.

Database systems are all-pervading today and most people interact, either directly or indirectly, with databases many times every day.

A major purpose of the database system is to provide users with an abstract view of the data. That is the system hides certain details of how the data are stored and maintain. Depending upon the creation, maintenance and users of database the users of DBMS can be classified as in section 3.5. Before getting in to more details of DBMS, let us first see what exactly is database?

Over the course of last four decades of the twentieth century, use of database grew in all enterprises. In early days very few users interacted with the database directly, but with the advent of Internet revolution of the late 1990 sharply increased direct user access to database.

2. Basic Concept and Definitions

The concept of maintaining and managing data is not a new one.

Data can be defined as a collection of facts and *figures*. It is generated in any type of transaction of any organization.

For example, An amount withdrawn from a particular account on a particular day is a data which is generated in any bank. Each organization has to record the details of such transactions and use it for further processing like preparing balance sheet, profit and loss account etc. When such a processing is done on data, it is called as *information*. So information is a processed form of data, which helps the managers to make decision.

Data and Information can broadly be differentiated as follows:

	Data	Information
1.	Data is a collection of facts and figures.	Information is a processed form of data.
2.	No conclusion can be drawn on data.	Information can be used to draw conclusion.
3.	Data cannot be used for decision-making.	Information plays an important role in decision making.
4.	Data is a raw material for processing. It is processed further.	Information can also be further utilized to produce highly identified information.
5.	It is not time bound.	It is time bound.

The group of inter-related data referring to a particular type forms a record. The group of records referring to a specific common type forms a database.

A collection of data designed to be used by different people is called a database. It is a collection of interrelated data stored together with controlled redundancy to serve one or more applications. It is organized in such a way that a computer program can quickly select desired pieces of data.

To access information from a database, you need a database management system (DBMS).

Databases are widely used and some of the representative examples are as below:

1. **Banking:** For customer information, accounts, loans, and banking transaction.
2. **Airlines:** For reservation and schedule information.
3. **Universities and big educational organizations:** For student information, course registration, libraries and grades.

4. **Credit card transactions:** For purpose on credit cards and generation of monthly statements.
5. **Telecommunications:** For keeping records of call made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication network.
6. **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
7. **Sales:** For customer product – sales and purchase information.
8. **On-line retailers:** For online order tracking.
9. **Manufacturing:** For management of Supply chain and for tracking production of items.
10. **Human resources:** For employee information and payrolls.

3. Definition of DBMS

A Database Management System (DBMS) is a collection of interrelated data and a set of programs to access those data.

The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Database is designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanism for the manipulation of information. In addition the database system must ensure the safety of information stored, despite system crashes or attempts at unauthorized access. If data is to be shared among several users, the system must avoid possible anomalous results.

More specifically, a DBMS is a *general purpose* software system facilitating each of the following (with respect to a database):

1. **Definition:** Specifying data types (and other constraints to which the data must conform) and data organization.
2. **Construction:** Insertion, updation and deletion of data as per need of transaction.
3. **Manipulation:** Querying, updating, report generation.
4. **Sharing:** Allowing multiple users and programs to access the database simultaneously.

5. **System protection:** Preventing database from becoming corrupted when hardware or software failures occur.

6. **Security protection:** Preventing unauthorized or malicious access to database.

A database together with the DBMS software is referred to as a **database system**.

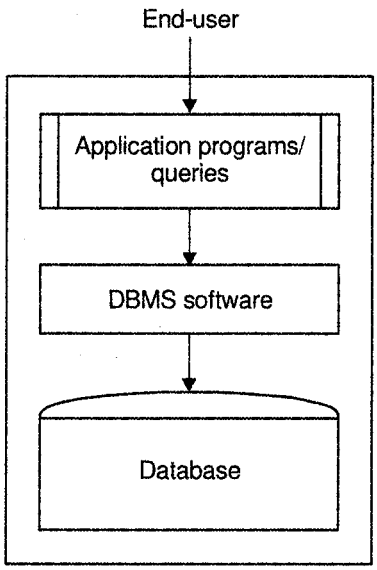


Figure 2.1: Database System

A DBMS is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. DBMS are categorized according to their data structures or types, some time DBMS is also known as Database Manager. It is a set of prewritten programs that are used to store, update and retrieve a Database. A DBMS includes:

A modeling language is used to define the schema of each database hosted in the DBMS, according to the DBMS data model. The four most common types of organizations are the hierarchical, network, relational and object models. Inverted lists and other methods are also used.

A given database management system may provide one or more of the four models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements (which include transaction rate (speed), reliability, maintainability, scalability, and cost).

Data structures (fields, records, files and objects) optimized to deal with very large amounts of data stored on a permanent data storage device (which implies relatively slow access compared to volatile main memory).

A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.

It also controls the security of the database. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. *For example*, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.

If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.

A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity, despite concurrent user accesses (concurrency control) and faults (fault tolerance).

It also maintains the integrity of the data in the database. The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database.

The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

Database servers are specially designed computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with RAID disk arrays used for stable storage. Connected to one or more servers via a high-speed channel, hardware database accelerators are also used in large volume transaction processing environments.

DBMSs are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system.

4. File Processing System Vs DBMS

Conventional File Systems Vs DBMS

▶ File Systems

In File Processing System, system allows permanent records in various files and it needs different application programs to extract records from and add records to, the appropriate files. Each time when need arises system programmers write these application programs to meet the needs so that system acquires more application programs, more files which will be time consuming always. It creates complexity, reduces efficiency of the system. File processing system has a number of many disadvantages, which we are going to discuss here.

Database system arouses in response to early methods of computerized management of commercial data. The earlier ways to keep the information on a computer was to use operating system files. These files could be manipulated by number of application programs. Thus a typical file-processing system is supported by a conventional operating system wherein a system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.

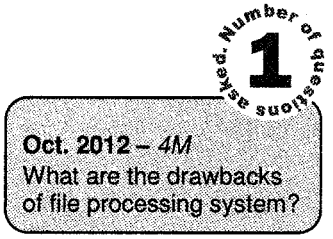
Keeping the organizational information in a file processing system has a number of major disadvantages as discussed in next section.

4.1 Limitations of File Processing System

Limitations of file processing system are
i. Data redundancy and inconsistency
ii. Difficulty in accessing data
iii. Data isolation
iv. Integrity problem
v. Atomicity problem
vi. Concurrent access anomalies
vii. Security problem

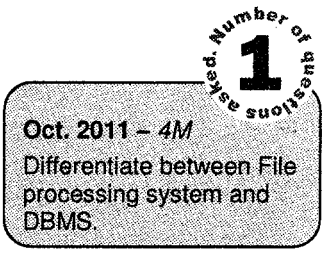
The various limitations of file processing system are:

1. **Data redundancy and inconsistency**: Different programmers may create files which may be stored at different locations in different data structures and there is a possibility of information duplication. *For example,* the information of student name and roll no. may appear in administration section storing student information for official records and it may also appear in library section for book issue and return purpose. This redundancy leads to higher storage and access cost. In addition it may lead to data inconsistency.



2. **Difficulty in accessing data:** Conventional file processing environment do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data retrieval systems are required for general use.
3. **Data isolation:** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
4. **Integrity problem**: The data values stored in the database must satisfy certain types of consistency constraints because when new constraints are added, it is difficult to change the program to enforce them. The problem is compounded when constraints involve several data items from different files.
5. **Atomicity problem:** In case of failure occurrence it is necessary that the data be restored to the consistent state that existed prior to failure. It is difficult to ensure atomicity in a conventional file processing system.
6. **Concurrent access anomalies:** In multiple user environments, interaction of concurrent updates is possible and may result in inconsistent data due to lack of supervision. In conventional file processing system, it is difficult to provide supervision but data may be accessed by many different application programs that have not been coordinated previously.
7. **Security problem:** In database system every user should not have all the privileges to access the data however as the application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraint is difficult.

4.2 Comparison of File Processing System and DBMS



The difficulties in the file-processing system propped up the development of database management system. Almost the entire shortfall of file-processing system was covered by database management system. Consider an organization/enterprise that is organized as a collection of departments/offices.

Each department has certain data processing "needs", many of which are unique to it. In the file processing approach, each department would control a collection of relevant data files and software applications to manipulate that data.

▶ Difference between a File Processing System and a DBMS

i. **Data Redundancy and Inconsistency:** In file processing system various files having various formats & program may be written in different languages so same information is stored in number. of locations. This is wastage of space and storing garbage data also. At that time redundancy occurs so data becomes inconsistent.

But this is avoided in DBMS.

ii. **Difficulty in Accessing Data:** In file processing system, if user wants a new application /report then he has to do such work separately. It means file processing system do not allow to retrieve data in efficient manner. But this is allowed in DBMS.

iii. **Data Isolation:** In file processing system, data are scattered, i.e., it is not isolated so it becomes very difficult to access data.

But in DBMS data is isolated so we can easily access the data.

iv. **Integrity Problem**: In file processing system we cannot enforce data integrity on any data values so integrity problem arises. But in DBMS it is possible. We enforce it using integrity constraints.

v. **Atomicity:** Atomicity problem occurs in file processing system, but it does not occur in DBMS.

vi. **Security:** It is not possible to assign privileges on data and user(s) in file processing system, which is easily possible in DBMS.

5. Advantages and Disadvantages of DBMS

5.1 Advantages of DBMS

Asked. Number of questions asked 1.

Apr. 2012 – *4M*
What are the advantages of DBMS?

1. **Controlling Redundancy:** Data redundancy (such as tends to occur in the "file processing" approach) leads to wasted storage space, duplication of effort (when multiple copies of a datum need to be updated), and a higher likelihood of the introduction of inconsistency.

   On the other hand, redundancy can be used to improve performance of queries. Indexes, *for example,* are entirely redundant, but help the DBMS in processing queries more quickly.

Another *example* of using redundancy to improve performance is to store an "extra" field in order to avoid the need to access other tables. A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated.

Advantages of DBMS

i. Controlling Redundancy
ii. Restricting Unauthorized Access
iii. Providing Persistent Storage for Program Objects
iv. Providing Storage Structures for Efficient Query Processing
v. Query processing and optimization
vi. Providing Backup and Recovery
vii. Providing Multiple User Interfaces
viii. Representing Complex Relationships among Data
ix. Enforcing Integrity Constraints
x. Permitting Inferencing and Actions via Rules

2. **Restricting Unauthorized Access:** A DBMS should provide a security and authorization subsystem, which is used for specifying restrictions on user accounts. Common kinds of restrictions are to allow read-only access (no updating), or access only to a subset of the data.

3. **Providing Persistent Storage for Program Objects:** Object-oriented database systems make it easier for complex runtime objects (e.g., lists, trees) to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.

4. **Providing Storage Structures for Efficient Query Processing:** The DBMS maintains indexes (typically in the form of trees and/or hash tables) that are utilized to improve the execution time of queries and updates. (The choice of which indexes to create and maintain is part of *physical database design and tuning* and is the responsibility of the DBA.

5. The **query processing and optimization** module is responsible for choosing an efficient query execution plan for each query submitted to the system.

6. **Providing Backup and Recovery:** The subsystem having this responsibility ensures that recovery is possible in the case of a system crash during execution of one or more transactions.

7. **Providing Multiple User Interfaces:** *For example*, query languages for casual users, programming language interfaces for application programmers, forms and/or command codes for parametric users, menu-driven interfaces for stand-alone users.

8. **Representing Complex Relationships among Data:** A DBMS should have the capability to represent such relationships and to retrieve related data quickly.

9. **Enforcing Integrity Constraints:** The data values stored in the database must satisfy certain types of consistency constraints. *For example,* the balance of a bank account should never fall below a particular amount say Rs. 1000/- or if the value of rate-of-interest is not fed by the operator it should be taken to be 12%. The DBMS provides the capabilities for defining and enforcing these types of constraints.

10. **Permitting Inferencing and Actions via Rules:** In a deductive database system, one may specify *declarative* rules that allow the database to infer new data. *Example, figure* out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.

    Active database systems go one step further by allowing "active rules" that can be used to initiate actions automatically.

5.2 Disadvantages of DBMS

A significant disadvantage of the DBMS system is cost. In addition to the cost of purchasing or developing the software, the hardware has to be upgraded to allow for the extensive programs and the work spaces required for their execution and storage. The processing overhead introduced by the DBMS to implement security, integrity and sharing of data causes a degradation of the response and through-put times. An additional cost is that of migration from a traditionally separate application environment to an integrated one.

While centralization reduces duplication, the lack of duplication requires that the database be adequately backed up so that in the case of failure the data can be recovered. Backup and recovery operations are fairly complex in a DBMS environment, and this becomes worse in a concurrent multi-user database system. Further more a database system requires a certain amount of controlled redundancies to enable access to related data items.

Centralization also means that the data is accessible from a single source, namely the database. This increases the potential severity of security breaches and disruption of the operation of the organization because of downtime and failures.

6. Users of DBMS

The basic objective of a database system is to store new information and retrieve stored information as far as possible at the lower processing cost. Thus the following are the users of the database. These apply to "large" databases, not "personal" databases that are defined, constructed, and used by a single person via, say, Microsoft Access.

1. **Database Administrator (DBA):** This is the chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc. In large organizations, the DBA might have a support staff.

Users of DBMS
i. Database Administrator(DBA)
ii. Database Designer
iii. End Users
iv. System Analysts, Application Programmers, Software Engineers

2. **Database Designers**: They are responsible for identifying the data to be stored and for choosing an appropriate way to organize it. They also define **views** for different categories of users. The final design must be able to support the requirements of all the user sub-groups.

3. **End Users:** These are persons who access the database for **querying, updating,** and **report generation**. They are main reasons for database's existence.

Number of questions asked. 1

Apr. 2011 – *4M*
List various Users of DBMS and specify their roles.

- *Casual end users:* Use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
- *Naive/Parametric end users*: Typically the biggest group of users; frequently query/update the database using standard **canned transactions** that have been carefully programmed and tested in advance. *Examples:*
  - Bank tellers check account balances, post withdrawals/deposits
  - Reservation clerks for airlines, hotels, etc., check availability of seats/rooms and make reservations.
  - Shipping clerks (e.g., at UPS) who use buttons, bar code scanners, etc., to update status of in-transit packages.
- *Sophisticated end users:* Engineers, scientists, business analysts who implement their own applications to meet their complex needs.
- *Stand-alone users:* Use personal databases, possibly employing a special-purpose (*for example:* financial) software package.

4. **System Analysts, Application Programmers, Software Engineers**

- *System Analysts*: determine needs of end users, especially naive and parametric users, and develop specifications for canned transactions that meet these needs.
- *Application Programmers*: These people write the codes of the programs according to the designs suggested by the database designers. They also test, debug, document and maintain these programs.

7. View of Data

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data stored and maintained.

7.1 Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interactions with the system.

8. The Three Level Architecture of DBMS

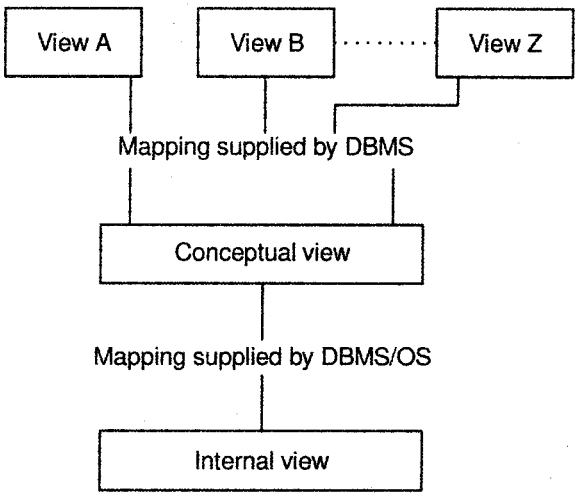


Figure 2.2: Three level architecture of DBMS

The architecture shown in *figure 2.2* of DBMS is divided into three levels: the external level, the conceptual level and the internal level. The view at each of these levels is described by a scheme.

A scheme is an outline or a plan that describes the records and relationships existing in a view. It describes the way in which entities at one level of abstraction can be mapped to the next level.

8.1 External or User View

The external or user view is at the highest level of database abstraction where only those portions of the database of concern to the user or application program are included.

Any number of user views may exist for a given global or conceptual view. Each external view is described by means of a scheme called an external schema. The external schema consists of the definition of the logical records and the relationship in the external view.

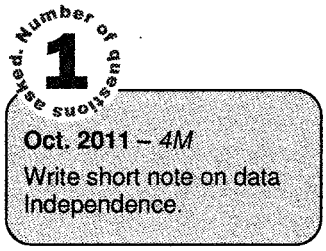
8.2 Conceptual or Global View

One conceptual view represents the entire database and there is only one conceptual scheme per database. The description of data at this level is in a format independent of its physical representation. It also includes features that specify the checks to retain data consistency and integrity.

8.3 Internal View

It is at the lowest level of abstraction, closest to the physical storage method used. It indicates how data will be stored and describes the data structure and access methods to be used by the database. The internal view is expressed by the internal schema, which contains the definition of the stored record, the method of representing the data fields, and access aids used.

8.4 Data Independence



Data Independence, basically concerned with storing and accessing of data, is independent of storage devices or locations or we can say logical data independence and physical data independence is there.

There are two distinct levels of data independence:

i. *Logical Data Independence*

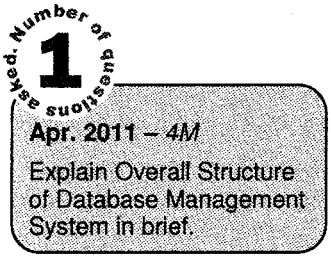
ii. *Physical Data Independence*

Three levels of abstraction, along with the mappings from internal to conceptual and from conceptual to external; provide two distinct levels of data independence: logical data independence and physical data independence.

Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schema. The change would be absorbed by the mapping between the external and conceptual level. It is achieved by providing the external level or user view of the database.

Physical data independence indicates that the physical storage structure or device used for storing data could be changed without necessitating a change in the conceptual view or any of the external view. The change would be absorbed by mapping between the conceptual and internal levels.

9. Overall System Structure



Query processor and Data manager together form the core of DBMS. Data manager (transaction manager, lock manager, files and index structures, buffer manager, disk space manager and recovery manager) is sometimes referred to as a database control system.

One of the functions of the data manager is to convert operations in the user's queries coming directly via the query processor or indirectly via an application program from the user's logical view to a physical file system. The data may come from a web forms, several applications or SQL interface in the form of SQL command.

The data manager is then responsible for interfacing with the file system. In addition, the tasks of enforcing constraints to maintain the consistency and integrity of the data, as well as its security, are

also performed by the concurrent users is under the control of the data manager. It is also entrusted with back up and recovery option.

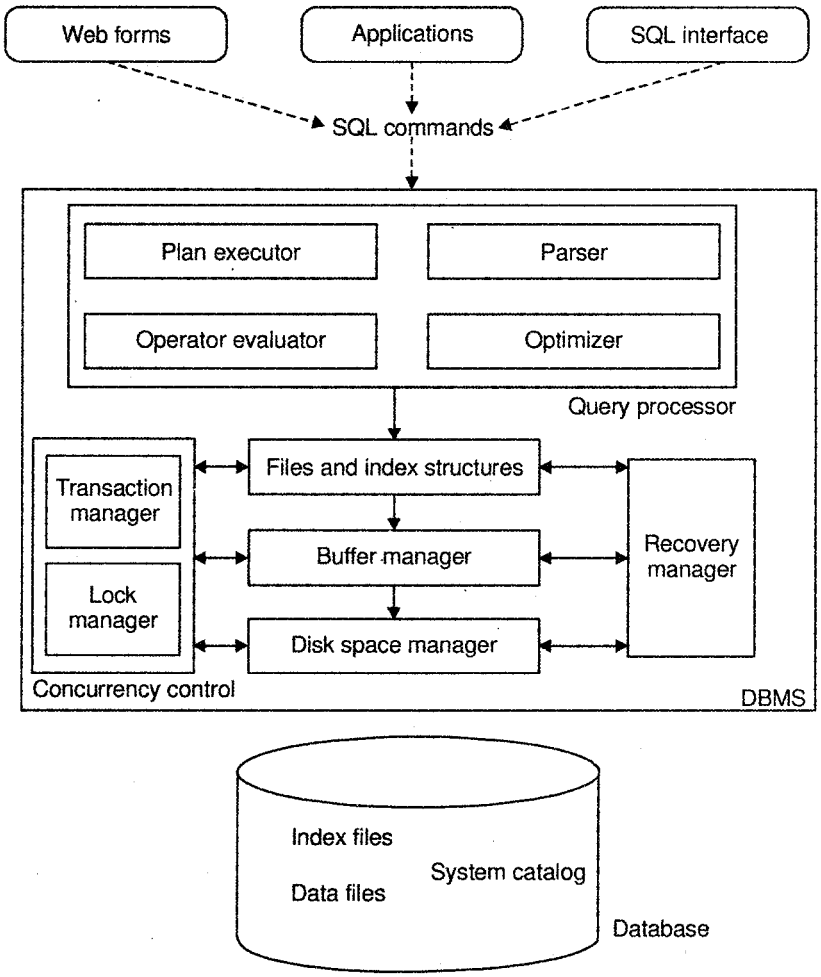


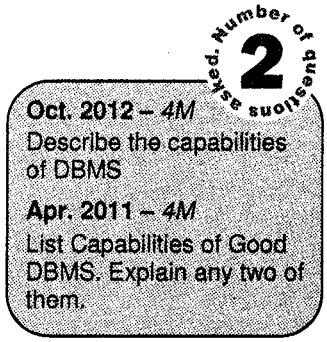
Figure 2.3: Overall database system structure

Responsibilities for the structure of the files and managing the file space rests with the file manager. It is also responsible for locating the block containing the required record, requesting this block from the disk manager, and transmitting the required record to the data manager.

Once the data is retrieved by formulating a query in the data manipulation language, the query processor is used to interpret the online user's query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution.

The disk manager is the part of the operating system of the host computer and all physical input and output operations are performed by it. The disk manager transfers the block or page requested by the file manager so that the later need not be concerned with the physical characteristics of the underlying storage media.

10. Features and Capabilities of DBMS



One can characterize a DBMS as an "attribute management system" where attributes are small chunks of information that describe something. *For example,* "colour" is an attribute of a car. The value of the attribute may be a color such as "red", "blue" or "silver".

Alternatively, and especially in connection with the relational model of database management, the relation between attributes drawn from a specified set of domains can be seen as being primary. For instance, the database might indicate that a car that was originally "red" might fade to "pink" in time, provided it was of some particular "make" with an inferior paint job. Such higher parity relationships provide information on all of the underlying domains at the same time, with none of them being privileged above the others.

Throughout recent history specialized databases have existed for scientific, geospatial, imaging, document storage and like uses. Functionality drawn from such applications has lately begun appearing in mainstream DBMSs as well. However, the main focus there, at least when aimed at the commercial data processing market, is still on descriptive attributes on repetitive record structures.

Thus, the DBMSs of today roll together frequently-needed services or features of attribute management. By externalizing such functionality to the DBMS, applications effectively share code with each other and are relieved of much internal complexity. Features commonly offered by database management systems include:

▶ Query ability

Querying is the process of requesting attribute information from various perspectives and combinations of factors. *Example:* "How many 2-door cars in Texas are green?"

A database query language and report writer allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data. It also controls the security of the database. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. *For example,* an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only

also performed by the concurrent users is under the control of the data manager. It is also entrusted with back up and recovery option.

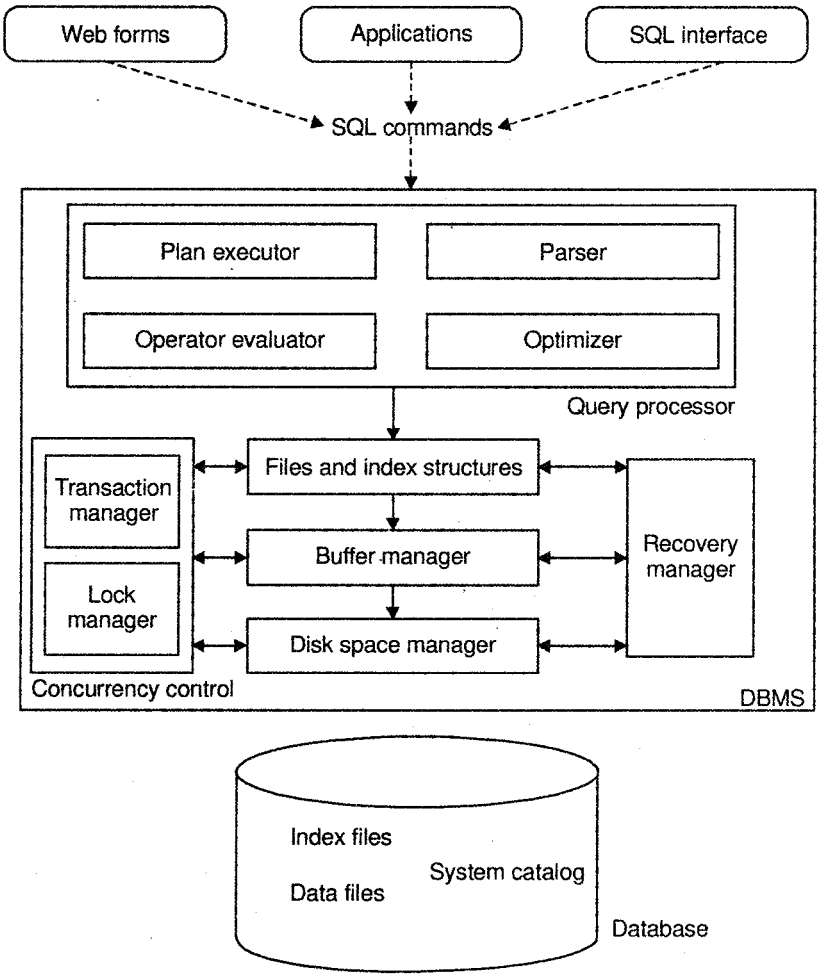


Figure 2.3: Overall database system structure

Responsibilities for the structure of the files and managing the file space rests with the file manager. It is also responsible for locating the block containing the required record, requesting this block from the disk manager, and transmitting the required record to the data manager.

Once the data is retrieved by formulating a query in the data manipulation language, the query processor is used to interpret the online user's query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution.

The disk manager is the part of the operating system of the host computer and all physical input and output operations are performed by it. The disk manager transfers the block or page requested by the file manager so that the later need not be concerned with the physical characteristics of the underlying storage media.

10. Features and Capabilities of DBMS

One can characterize a DBMS as an "attribute management system" where attributes are small chunks of information that describe something. *For example,* "colour" is an attribute of a car. The value of the attribute may be a color such as "red", "blue" or "silver".

Number of questions asked 2

Oct. 2012 – *4M*
Describe the capabilities of DBMS

Apr. 2011 – *4M*
List Capabilities of Good DBMS. Explain any two of them.

Alternatively, and especially in connection with the relational model of database management, the relation between attributes drawn from a specified set of domains can be seen as being primary. For instance, the database might indicate that a car that was originally "red" might fade to "pink" in time, provided it was of some particular "make" with an inferior paint job. Such higher parity relationships provide information on all of the underlying domains at the same time, with none of them being privileged above the others.

Throughout recent history specialized databases have existed for scientific, geospatial, imaging, document storage and like uses. Functionality drawn from such applications has lately begun appearing in mainstream DBMSs as well. However, the main focus there, at least when aimed at the commercial data processing market, is still on descriptive attributes on repetitive record structures.

Thus, the DBMSs of today roll together frequently-needed services or features of attribute management. By externalizing such functionality to the DBMS, applications effectively share code with each other and are relieved of much internal complexity. Features commonly offered by database management systems include:

▶ Query ability

Querying is the process of requesting attribute information from various perspectives and combinations of factors. *Example:* "How many 2-door cars in Texas are green?"

A database query language and report writer allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data. It also controls the security of the database. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. *For example,* an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only

work history and medical data. If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.

▶ Backup and Replication

Copies of attributes need to be made regularly in case primary disks or other equipment fails. A periodic copy of attributes may also be created for a distant organization that cannot readily access the original. DBMS usually provide utilities to facilitate the process of extracting and disseminating attribute sets.

When data is replicated between database servers, so that the information remains consistent throughout the database system and users cannot tell or even know which server in the DBMS they are using, the system is said to exhibit replication transparency.

▶ Rule enforcement

Often one wants to apply rules to attributes so that the attributes are clean and reliable.

For example, we may have a rule that says each car can have only one engine associated with it (identified by Engine Number). If somebody tries to associate a second engine with a given car, we want the DBMS to deny such a request and display an error message. However, with changes in the model specification such as, in this example, hybrid gas-electric cars, rules may need to change. Ideally such rules should be able to be added and removed as needed without significant data layout redesign.

▶ Security

Often it is desirable to limit who can see or change which attributes or groups of attributes. This may be managed directly by individual, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements.

▶ Computation

There are common computations requested on attributes such as counting, summing, averaging, sorting, grouping, cross-referencing, etc. Rather than have each computer application implement these from scratch, they can rely on the DBMS to supply such calculations.

Often one wants to know who accessed what attributes, what was changed, and when it was changed. Logging services allow this by keeping a record of access occurrences and changes.

▶ Automated Optimization

If there are frequently occurring usage patterns or requests, some DBMS can adjust themselves to improve the speed of those interactions. In some cases the DBMS will merely provide tools to monitor performance, allowing a human expert to make the necessary adjustments after reviewing the statistics collected.

11. Data Model

Beneath the structure of database is the data model. A collection of conceptual tools for describing data, data relationships, data semantics, and data consistency constraints. The model provides the way to describe the design of a database at the physical, logical and view level.

A **data model** is therefore an abstract model that describes how data is represented and accessed.

The term **data model** has two generally accepted meanings:

1. A data model *theory*, i.e., a formal description of how data may be structured and accessed.
2. A data model *instance*, i.e., applying a data model *theory* to create a practical data model *instance* for some particular application.

A *data model theory* has three main components:

- The *structural* part: A collection of data structures which are used to create databases representing the entities or objects modeled by the database.
- The *integrity* part: A collection of rules governing the constraints placed on these data structures to ensure structural integrity.
- The *manipulation* part: A collection of operators which can be applied to the data structures, to update and query the data contained in the database.

For example, in the relational model, the structural part is based on a modified concept of the mathematical relation; the integrity part is expressed in first-order logic and the manipulation part is expressed using the relational algebra, tuple calculus and domain calculus. Data modeling is the process of creating a data model instance by applying a data model theory. This is typically done to solve some business enterprise requirement.

Business requirements are normally captured by a semantic logical data model. This is transformed into a physical data model instance from which is generated a physical database.

For example, a data modeler may use a data modeling tool to create an Entity Relationship Diagram (ERD) of the Corporate data repository of some business enterprise. This model is transformed into a relational model, which in turn generates a relational database.

A data model *instance* may be one of three kinds:

Three types of Schema
i. A conceptual schema
ii. A logical schema
iii. A physical schema

i. **A conceptual schema**: (data model) Describes the semantics of a domain, being the scope of the model. *For example,* it may be a model of the interest area of an organization or industry. This consists of entity classes (representing kinds of things of significance in the domain) and relationships (assertions about associations between pairs of entity classes).

A conceptual schema specifies the kinds of facts or propositions that can be expressed using the model. In that sense, it defines the allowed expressions in an artificial 'language' with a scope that is limited by the scope of the model.

ii. **A logical schema**: (data model) Describes the semantics, as represented by a particular data manipulation technology. This consists of descriptions of tables and columns, object oriented classes, and XML tags, among other things.

iii. **A physical schema**: (data model) Describes the physical means by which data are stored. This is concerned with partitions, CPUs, tablespaces, and the like.

12. Object Based Logical Model

Object-based logical models:

- Describe data at the conceptual and view levels.
- Provide fairly flexible structuring capabilities.
- Allow one to specify data constraints explicitly.
- Over 30 such models, including
  - Entity-relationship model
  - Object-oriented model
  - Binary model
  - Semantic data model
  - Infological model
  - Functional data model

13. Record Based Logical Model

▶ Record Based Logical Model

In record based logical model, the database is structured in fixed format records of several types. Each record type is made up of a fixed number of fields or attributes and each field is usually of fixed length.

These models do not include a mechanism for the direct representation of code in the database as in case of object-based model.

Instead, there are separate languages that are associated with models to express database queries. These data models are:

- Entity Relationship Model
- Network Model
- Hierarchical Model

13.1 Relational Model

Oct. 2012 – *4M*
What is data model?
Explain relational model.

RDBMS (Relational Database Management System) is a database based on the relational model developed by E.F. Codd.

A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

- Values are atomic.
- Each row is unique.
- Column values are of the same kind.
- The sequence of columns is insignificant.
- The sequence of rows is insignificant.
- Each column has a unique name.

Certain fields may be designated as keys, which mean that searches for specific values of that field will use indexing to speed them up.

Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables.

For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables.

This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retreival time, relational databases are classed as dynamic database management system.

The Relational database model is based on the Relational Algebra.

Consider a database for the Universal Hockey League (UHL), a professional ice hockey league with teams worldwide. It consists of a number of divisions and numerous franchises under each division. The database records statistics on teams, players, and divisions of the league.

A franchise may relocate to another city and may become part of a different division. Players are under contract to a franchise and are obliged to move with it. This relationship between a franchise and a division is called a team. We use the word team synonymously with franchise.

Consequently, we can view a franchise as consisting of a collection of players, coaches, and general manager. Players are required to play for a given franchise for the entire season.

Using the relational model, each of the entities in the UHL can be represented by a relation as shown in *figure.*

The description of the relation is given by a relation scheme. A relation scheme is like a type declaration in a programming language. It indicated the attributes included in the scheme, their order, and their domain.

However, we will ignore the domain for the present.

Player

Name	Birth Place	Birth Date
Zax Viviteer	Prague, Czec	1962-04-29
Barn Kurri	Dtroit, Mich	1964-07-17
Todd Smith	Roseu, Minn	1963-05-09
Dave Fisher	Edmonton, Canada	1959-10-28
Ozzy Xavier	Kiruna, Sweden	1965-02-19
Gaston Vabr	Montreal, Canada	1958-05-12
Ken Dorky	Chicago, I11	1958-05-13
Brain Lafontaine	Paris, France	1960-07-03
Bruce McTavish	Rio, Brazil	1966-10-27
Dave O'Connell	Dublin, Ireland	1967-03-16
Johny Brent	Boston, Mass	1964-12-23

Franchise

Franchise_Name	Year_Establishment
Bullets	1975
Rodeos	1921
Zippers	1917
Blades	1982
Flashers	1967

Division

Division_Name
Northern
Southern
European
World

Forward

Name	Franchise_Name	Year	Goals	Assists
Barn Kurri	Bullets	1986	40	67
Bruce Mc Tavish	Bullets	1986	30	37
Todd Smith	Rodeos	1986	17	24
Ozzy Xavier	Blades	1986	56	119
Ozzy Xavier	Flashers	1985	36	49
Gaston Vabr	Flashers	1986	16	22
Zax Viviteer	Blades	1986	80	162
Dave O'Connell	Zippers	1986	12	59
Brain Lafontaine	Zippers	1985	10	40
Brain Lafontaine	Zippers	1986	22	73

Goal

Name	Franchise_Name	Year	Goals_Against_Avg	Shoutouts
Davy Fisher	Blades	1986	1.21	7
Ken Dorky	Zippers	1986	4.02	4
Johny Brent	Flashers	1986	7.61	0
Dave Fisher	Flashers	1985	3.05	5

Team

Franchise_Name	Division_Name	Year	City	Points
Flashers	Northern	1986	Sr. Loues	93
Blades	Northern	1986	Edmonton	97
Zippers	European	1985	Paris	82
Zippers	Northern	1986	Montreal	99
Rodeos	Southern	1986	Rio	65
Bullets	World	1986	Tokyo	79

Figure 2.4: Relational model for UHL

13.2 Network Model

Apr. 2012 – *4M*
Explain Network Model.

Oct.11, Apr.11 – *4M*
Explain Network Model with example.

The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data. In 1971, the Conference on Data Systems Languages (CODASYL) formally defined the network model. The basic data modeling construct in the network model is the set construct.

A set consists of an owner record type, a set name, and a member record type. A member record type can have that role in more than one set, hence the multiparent concept is supported. An owner record type can also be a member or owner in another set. The data model is a simple network, and link and intersection record types (called junction records by IDMS) may exist, as well as sets between them. Thus, the complete network of relationships is represented by several pairwise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, a set defines a 1:M relationship, although 1:1 is permitted. The CODASYL network model is based on mathematical set theory.

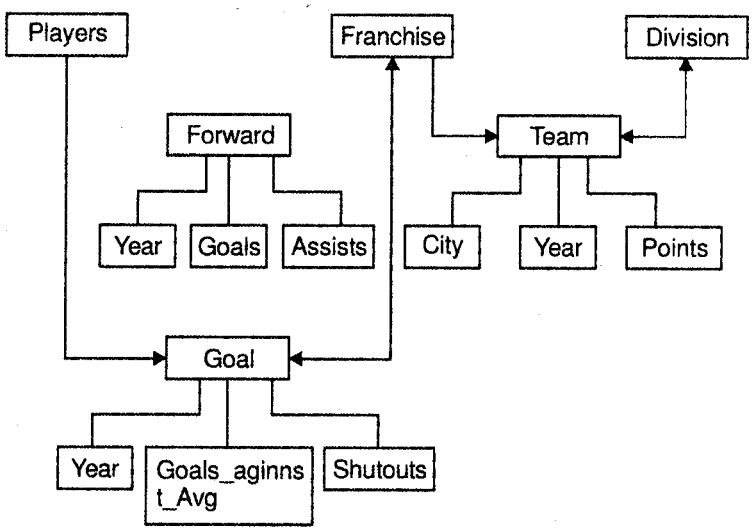


Figure 2.5: Network model for UHL

13.3 Hierarchical Model

The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type.

These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows.

To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1: N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths.

For example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth.

The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database, the parent-child relationship is one to many.

This restricts a child segment to having only one parent segment. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

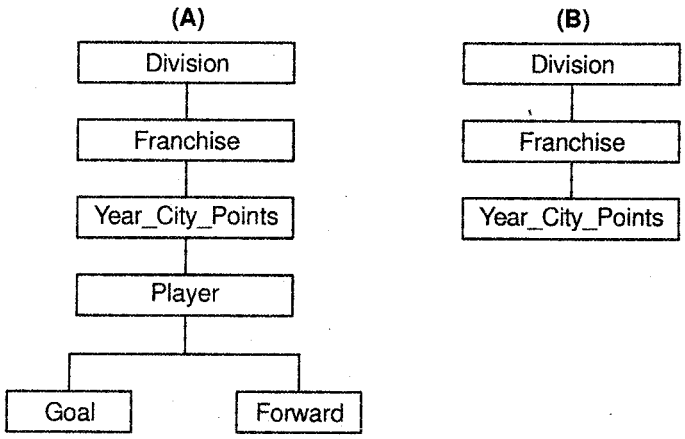


Figure 2.6: Hierarchical model for UHL

As we see above *figure* representing the many-to-many relationship between the players and the franchise requires the introduction of certain redundancies and inefficiencies. Furthermore, we cannot follow the player hierarchy to find out the players score in a given year. This involves, first finding the franchises to which a player belonged from the PLAYER hierarchy, second, we have to refer to the DIVISION hierarchy to find this FRANCHISE and, for the required year, find the player and his score.

In the hierarchical model, we can have duplications of certain record occurrences as well.

14. Entity Relationship Model

The entity-relationship model (or ER model) is a way of graphically representing the logical relationships of entities (or objects) in order to create a database. The ER model was first proposed by Peter Pin-Shan Chen of Massachusetts Institute of Technology (MIT) in the 1970s.

In ER modeling, the structure for a database is portrayed as a diagram, called an entity-relationship diagram (or ER diagram), that resembles the graphical breakdown of a sentence into its grammatical parts.

Entities are rendered as points, polygons, circles, or ovals. Relationships are portrayed as lines connecting the points, polygons, circles, or ovals.

Any ER diagram has an equivalent relational table, and any relational table has an equivalent ER diagram. ER diagramming is an invaluable aid to engineers in the design, optimization, and debugging of database programs.

In a logical sense, entities are the equivalent of grammatical nouns, such as employees, departments, products, or networks. An entity can be defined by means of its properties, called attributes.

Relationships are the equivalent of verbs or associations, such as the act of purchasing, the act of repairing, being a member of a group, or being a supervisor of a department.

A relationship can be defined according to the number of entities associated with it, known as the degree.

A *database* can be modeled in ER as:

- A collection of entities.
- Relationship among entities.

An *entity* is an object that exists and is distinguishable from other objects.

Example: specific person, company, event, plant

14.1 Entity Sets

An *entity set* is a set of entities of the same type that share the same properties or attributes. For example consider the entity sets i.e., customer and loan shown in *figure 2.7.*

The set of all persons who are customers at a given bank is defined as an entity set customer. Similarly the entity loan represents the set of all loans awarded by a particular bank.

The individual entities that constitute a set are said to be the extension of the entity set. Thus all the individual bank customers are the extensions of the entity set customer.

Customer			
321-123123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupont	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassai	Princeton
335-57-7991	Adams	Spring	Pittsfiels

Loan	
L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

Figure 2.7: Entity set

14.2 Entity Attributes

▶ Definition of attributes

An entity is represented by a set of attributes that is descriptive properties possessed by all members of an entity set.

Example: Consider the following two entities that is customer and loan, then the attributes for each respectively are:

Customer = (customer-id, customer-name, customer-street, customer-city)

loan = (loan-number, amount)

➢ Attribute types

Asked. Number of questions 1

Apr. 2011 – 4M

What is an Attribute? Explain its types.

An attribute, as used in the E-R model, can be characterized by the following attribute types:

1. **Simple and composite attributes:** Composite attributes are those which can be divided into sub parts. For example, as shown in *figure 2.8* the attribute name can be structured into attributes consisting of first-name, middle-name and last-name.

Also it can be observed in the *figure* that the composite attribute address consisting of street can further be structured into street-number, street-name and apartment-number, it is known as component attribute.

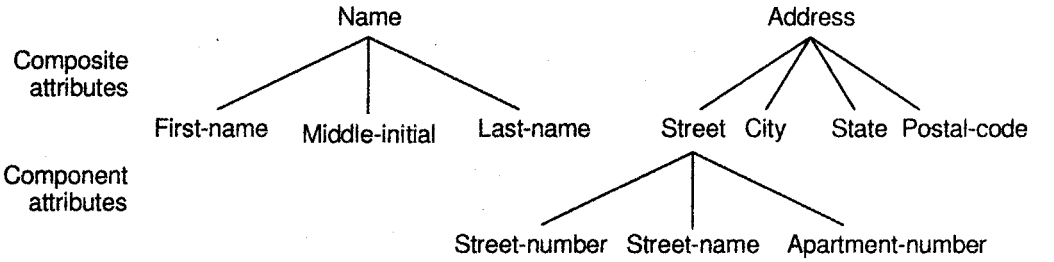


Figure 2.8: Composite attributes customer_name and customer_address

2. **Single-valued and multi-valued attributes:** There are instances that where an attribute has set of values for a specific entity. *For example* phone number is a multivalued attribute as there can be multiple phone numbers associated to an individual.
3. **Derived attributes:** The value for this type of attribute can be derived from the values of other related attributes or entities. *For example* to compute age, given date of birth.

14.3 Relationship Sets

A relationship is an association among several entities.

Example:

Hayes *depositor* A-102
customer entity relationship set *account* entity

A *relationship* set is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$\{(e1, e2, \ldots en) \mid e1 \in E1, e2 \in E2, \ldots, en \in En\}$ where $(e1, e2, \ldots, en)$ is a relationship

Example: (See figure 2.9)

(Hayes, A-102) $\in$ *depositor*

An *attribute* can also be property of a relationship set. For instance, the *depositor* relationship set between entity sets *customer* and *account* may have the attribute *access-date.*

Relationship sets that involve two entity sets are *binary* (or degree two).

Generally, most relationship sets in a database system are binary. Relationship sets may involve more than two entity sets.

For example, suppose employees of a bank may have jobs (responsibilities) at multiple branches, with different jobs at different branches.

Then there is a ternary relationship set between entities sets *employee, job and branch.* Relationships between more than two entity sets are rare. Most relationships are binary.

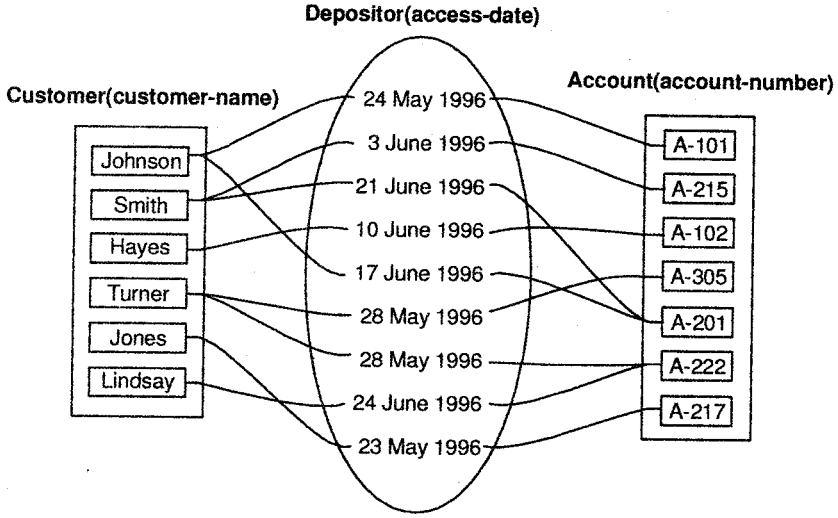


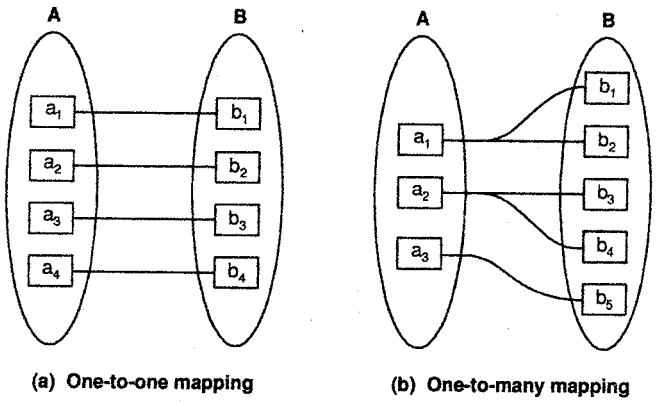
Figure 2.9: Example of relationship set

14.4 Mapping Cardinalities

Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

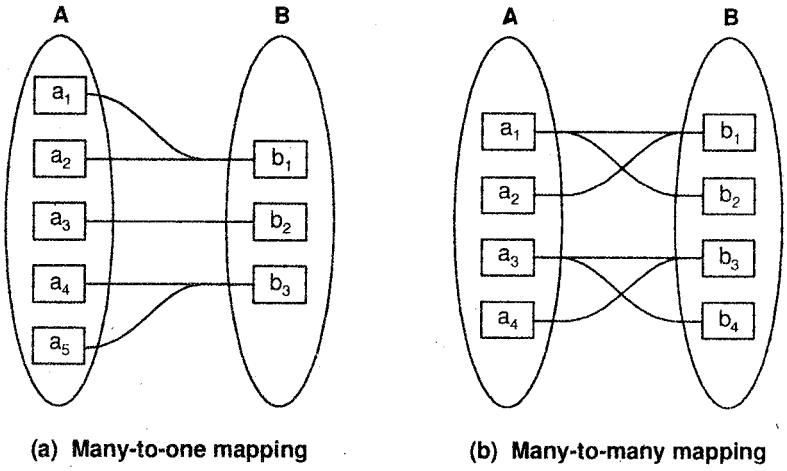
Number of questions asked: 1

Oct. 2012 – *4M*
Define:
i. Relation
ii. Cardinality



(a) One-to-one mapping (b) One-to-many mapping

Figure 2.10: (a) One-to-One Mapping (b) One-to- Many Mapping



(a) Many-to-one mapping (b) Many-to-many mapping

Figure 2.11: (a) Many-to-One Mapping (b) Many-to-Many Mapping

i. **One – to – One:** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

ii. **One – to – Many:** An entity in A is associated with any number of entities in B. An entity in B is however, can be associated with at most one entity in A.

iii. **Many – to – One:** An entity in A is associated with at most one entity in B. An entity in B can however be associated with any number of entities in A

iv. **Many – to – Many:** An entity in A is associated with any number of entities in B, and an entity in B is associated with any number of entities in A.

15. Entity Relationship Diagram (ERD)

An E-R diagram can express the overall structure of a database graphically. E-R diagrams can be represented with the following major components:

- **Rectangles** represent entity sets.
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Ellipses** represent attributes

- *Double ellipses* represent multivalued attributes.
- *Dashed ellipses* denote derived attributes.

- **Underline** indicates primary key attributes.
- **Double lines** represents total participation of an entity in relatiohsip set.
- **Double rectangles** represent weak entity set.

Consider the entity relationship diagram in *figure 2.12*, which consists of two entity sets, customer and loan, related through a binary relationship set borrower.

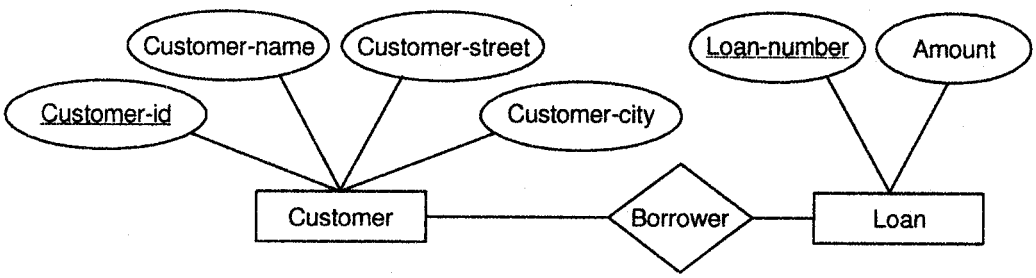


Figure 2.12: E-R diagram corresponding to customers and loans

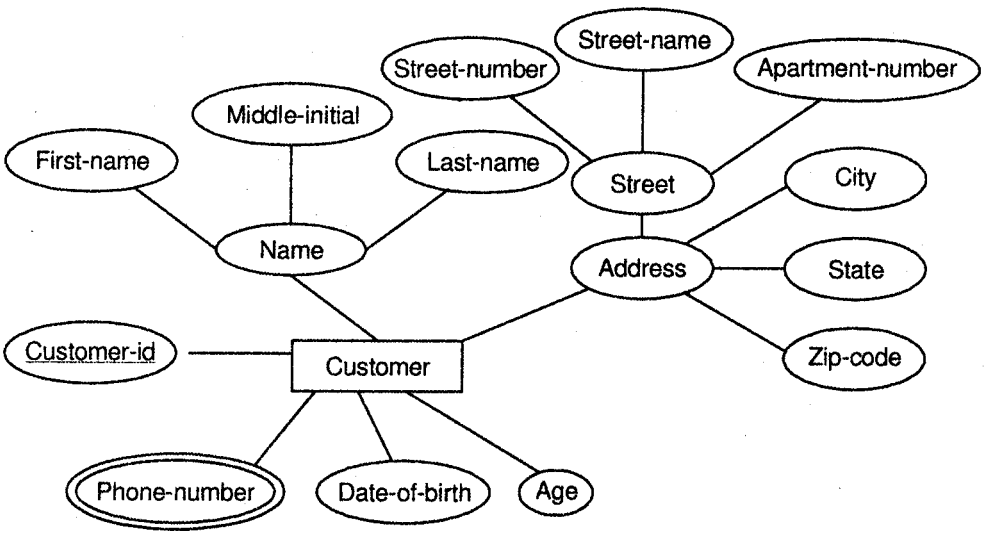


Figure 2.13: E-R diagram with composite, multi-valued and derived attributes

Figure 2.13 illustrates a multivalued attribute phone_number, depicted by double ellipse, and the derived attribute age, depicted by a dashed ellipse.

If a relationship set has also some attributes associated with it, then we link these attributes to that relationship set.

For example, in figure *2.14* we have the access_date descriptive attribute attached to the relationship set depositor to specify the most recent date on which the customer accessed the account.

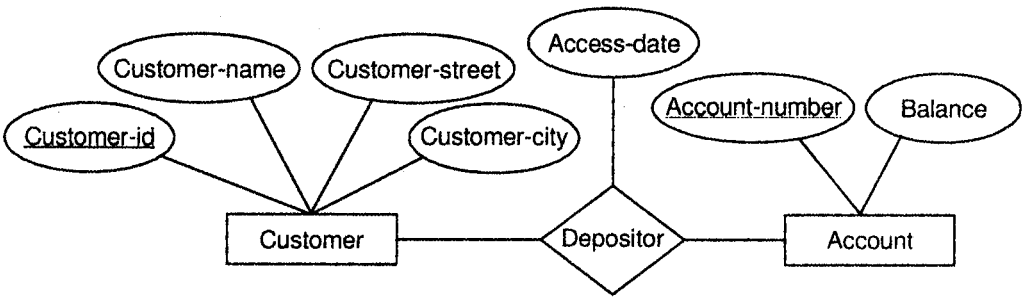


Figure 2.14: E-R diagram with an attribute attached to a relationship set

We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

For example, one-to-one relationship (*See figure 2.15*):

- A customer is associated with at most one loan via the relationship *borrower.*
- A loan is associated with at most one customer via *borrower.*

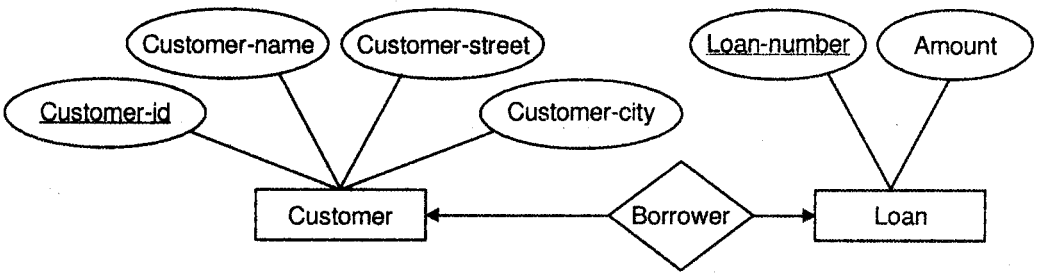


Figure 2.15: One-to-one relationship

In one-to-many relationship, a loan is associated with atmost one customer via *borrower,* a customer is associated with several (including 0) loans via *borrower.* See *figure 2.16.*

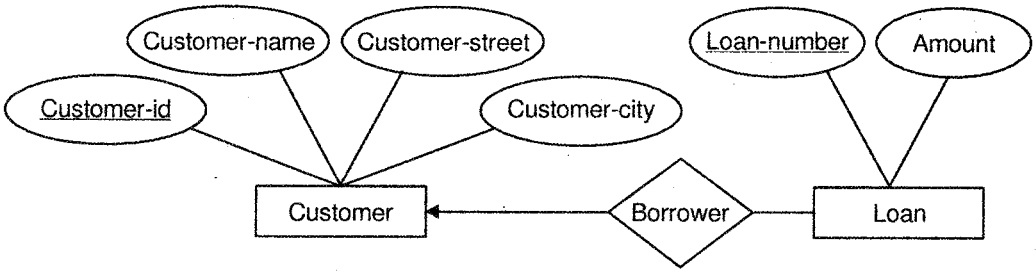


Figure 2.16: One-to-many relationship

In many-to-one relationship, a loan is associated with several (including 0) customers via *borrower*, a customer is associated with at most one loan via *borrower*. *See figure 2.17.*

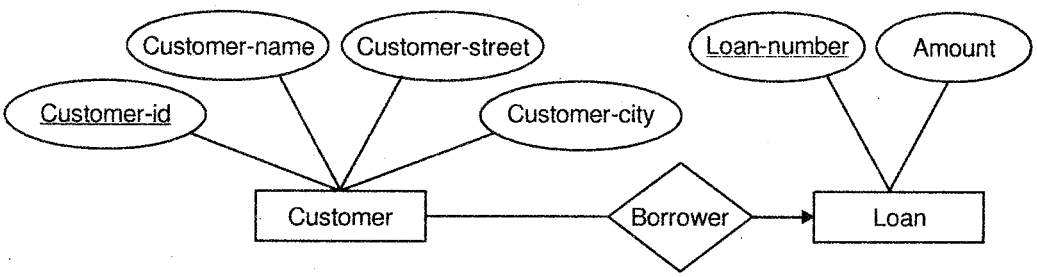


Figure 2.17: Many-to-one relationship

In many – to – many, a customer is associated with several (possibly 0) loans via borrower and a loan is associated with several (possibly 0) customers via borrower. *See figure 2.18.*

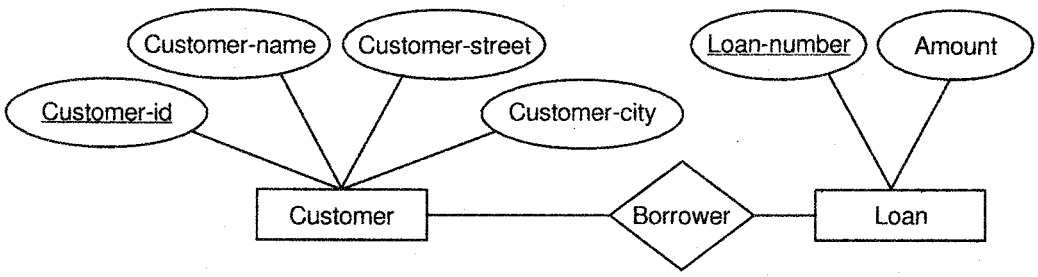


Figure 2.18: Many-to-Many relationship

Participation of an entity set in a relationship set can be partial or total.

Total participation (indicated by double line): Every entity in the entity set participates in at least one relationship in the relationship set.

For example, participation of *loan* in *borrower* is total and every loan must have a customer associated to it via borrower.

Partial participation: Some entities may not participate in any relationship in the relationship set.

For example, participation of *customer* in *borrower* is partial.

E-R diagrams also provide a way to indicate more complex constraints on the number of times each entity participates in relationships in a relationship set. An edge between an entity set and a binary relationship set can have an associated minimum and maximum cardinality, shown in the form *l....h* where *l* is the minimum and *h* the maximum cardinality.

A minimum value of 1 indicates total participation of the entity set in the relationship set. A maximum value of 1 indicates that the entity participates in atmost one relationship, while a maximum value * indicates no limit. Note that a label 1..* on an edge is equivalent to a double line.

For example consider *figure 2.20.* The edge between *loan* and *borrower* has a cardinality constraint 1...1, meaning the minimum and the maximum cardinality are both 1. That is, each loan must have exactly one associated customer. The limit 0...* on the edge from customer to borrower indicates that a customer can have zero or more loans.

Thus, the relationship borrower is one-to-many from customer to loan, and further the participation of loan in borrower is total.

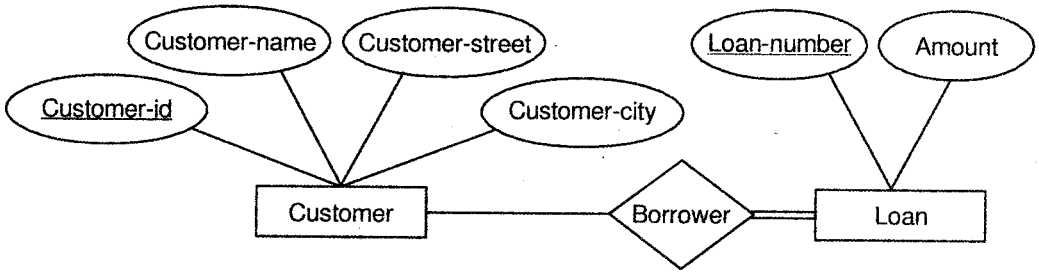


Figure 2.19: Participation of an entity set in relationship set

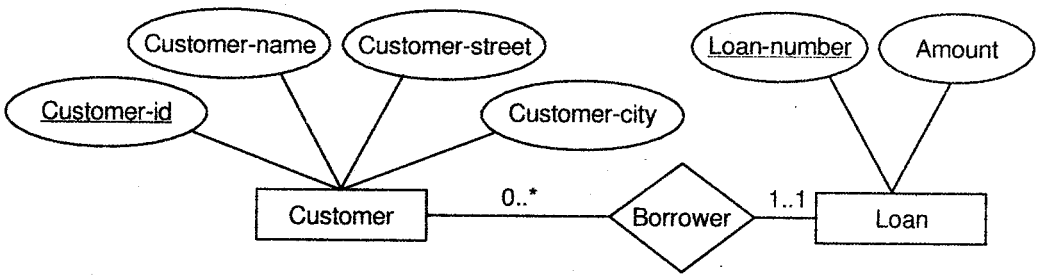


Figure 2.20: Cardinality limits on a relationship set

Entity sets of a relationship need not be distinct. The labels "manager" and "worker" in *figure 2.21* are called roles; they specify how employee entities interact via the works-for relationship set.

Roles are indicated in E-R diagrams by labeling the lines that connect diamonds to rectangles. Role labels are optional, and are used to clarify semantics of the relationship.

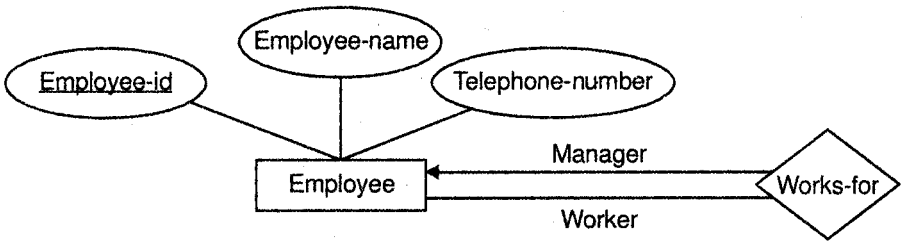


Figure 2.21: Role indicators

16. Extended Features of ERD

Number of questions asked. 1

Oct. 2012 – *4M*
Write short note on extended features of ERD?

Apart from the various features of ER diagram discussed above there can be certain extensions to the basic E-R diagram. The extended features are specialization, generalization, attribute inheritance and aggregation.

16.1 Specialization and Generalization

▶ Specialization

Number of questions asked. 1

Apr. 2011 – *4M*
Differentiate between Specialization and Generalization with example.

An entity set may include subgroupings of entities that are distinct in some way from the other entities in the set.

For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The ER model provides the means for representing these distinctive entity groupings.

As an example consider an entity set person, with attributes person_id, name, street and city. A person may be further classified as one of the following:

- Customer
- Employee

Each of this person type is described by a set of attributes that include all the attributes of entity set person plus possible additional attributes. *For example*, customer entities may be described further by an attribute credit_rating, whereas entities may be described further by the attribute salary. The process of designing subgroupings within an entity set is called specialization.

In terms of an E-R diagram, specialization is depicted by a triangle component labeled ISA, as shown in *figure 2.22*. The labeled ISA stands for "is a" and represents, for example, that a customer "is a" person. The ISA relationship may also be referred to as a super class – sub class relationship. Higher and lower entity sets are depicted as regular entity sets - that is, as rectangles containing the name of the entity set.

▶ Generalization

Number of questions asked. 1

Apr. 2012 – *4M*
Write short note on Generalization.

The refinement from an initial entity set into successive levels of entity subgroupings represent a top down design process in which distinctions are made explicit. The design process may also proceed in a bottom up manner, in which multiple entity sets are synthesized into a higher level entity set on the basis of common features.

The database designer may have first identified a customer entity set with the attributes customer_id, customer_name, customer_city and credit_rating and employee entity set with the attributes employee_id, employee_name, employee_city and employee_salary.

There are similarities between the customer entity set and the employee entity set in the sense that they have several attributes that are conceptually the same across the two entity sets: namely, the identifier, name, street, and city attributes. This commonality can be expressed by generalization, which is a containment relationship that exists between the higher level entity set and one or more lower level entity set. In our *example* person is the higher level entity set and customer and employee are lower level entity sets.

16.2 Attribute Inheritance

A crucial property of the higher and lower level entities created by specialization and generalization is attribute inheritance. The attributes of the higher entity sets are said to be inherited by the lower level entity sets.

Figure 2.22 depicts the hierarchy of entity sets.

In the *figure*, employee is a lower level entity set of person and a higher level entity set of the officer, teller and secretary entity sets.

In a hierarchy, a given entity set may be involved as a lower level entity set in only one ISA relationship; that is entity sets in this diagram have only single inheritance. If an entity set is a lower level entity set is more than one ISA relationship then the entity set has multiple inheritance, and the resulting structure is said to be lattice.

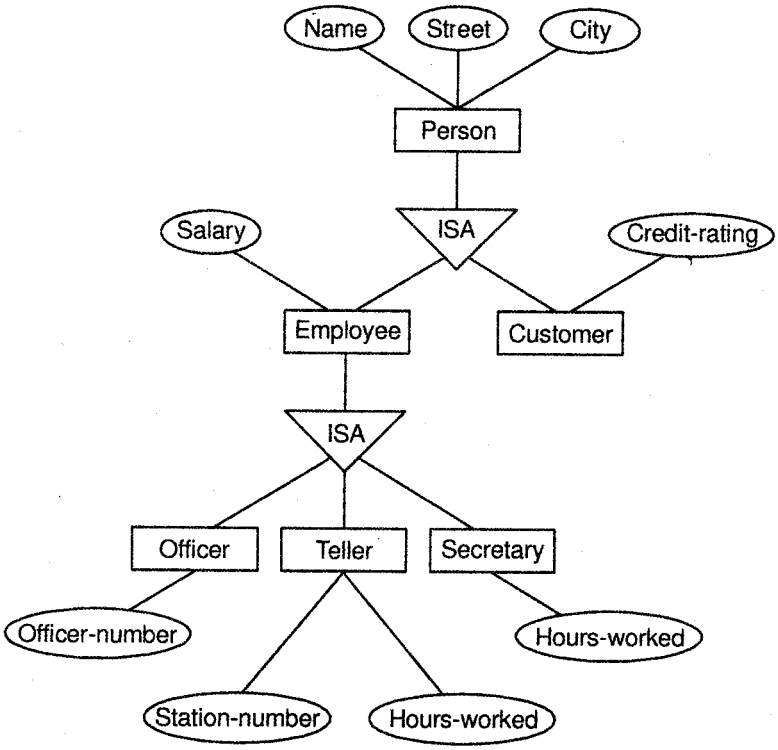


Figure 2.22: Specialization and generalization

16.3 Aggregation

Aggregation is an abstraction through which relationships are treated as higher level entities. Thus for example, we regard the relationship set works_on as a higher level entity set. Such entity set is treated in the same manner as in any other entity set. Then we create a binary relationship manages between works_on and manager to represent who manages what task. *Figure 2.23* shows a notation for aggregation commonly used to represent this situation.

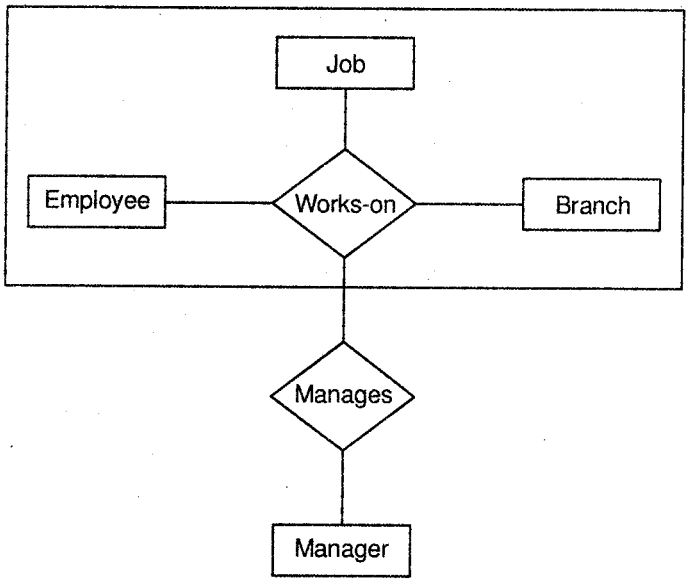


Figure 2.23: Aggregation

17. The Object-Oriented Model

1. The object-oriented model is based on a collection of objects, like the E-R model.
   - An object contains values stored in **instance variables** within the object.
   - Unlike the record-oriented models, these values are themselves objects.
   - Thus objects contain objects to an arbitrarily deep level of nesting.
   - An object also contains bodies of code that operate on the the object.
   - These bodies of code are called **methods**.
   - Objects that contain the same types of values and the same methods are grouped into **classes**.
   - A class may be viewed as a type definition for objects.
   - Analogy: the programming language concept of an abstract data type.
   - The only way in which one object can access the data of another object is by invoking the method of that other object.

- This is called **sending a message** to the object.
- Internal parts of the object, the instance variables and method code, are not visible externally.
- Result is two levels of data abstraction.

For example, consider an object representing a bank account.

- The object contains instance variables *number* and *balance*.
- The object contains a method *pay-interest* which adds interest to the balance.
- Under most data models, changing the interest rate entails changing code in application programs.
- In the object-oriented model, this only entails a change within the *pay-interest* method.

2. Unlike entities in the E-R model, each object has its own unique identity, independent of the values it contains:
   - Two objects containing the same values are distinct.
   - Distinction is created and maintained in physical level by assigning distinct object identifiers.

18. Case Study for E-R Diagrams

Consider the following banking enterprise with the following major banking characteristics:

1. The bank is organized into branches. Each branch is located into a particular city and is identified by a particular name. The bank monitors the asset of each branch.
2. Bank customers are identified by their customer identification number. Customer personal detail is stored in the bank. Customers may have account and take loans.
3. Bank stores the personal details of the employees and maintains the track of employees start date and length of service.
4. The bank offers two types of account – saving and checking account. Accounts can be held by more than one customer and customer can have more than one account in the bank. Each account has unique account number. The bank maintains all related account details.
5. A loan originates at a particular branch and can hold by one or more customers. A loan is identified by a unique loan number. For each loan, the bank keeps track of the loan amount and the loan payments.

Figure 2.24 shows the E-R diagram for the banking example.

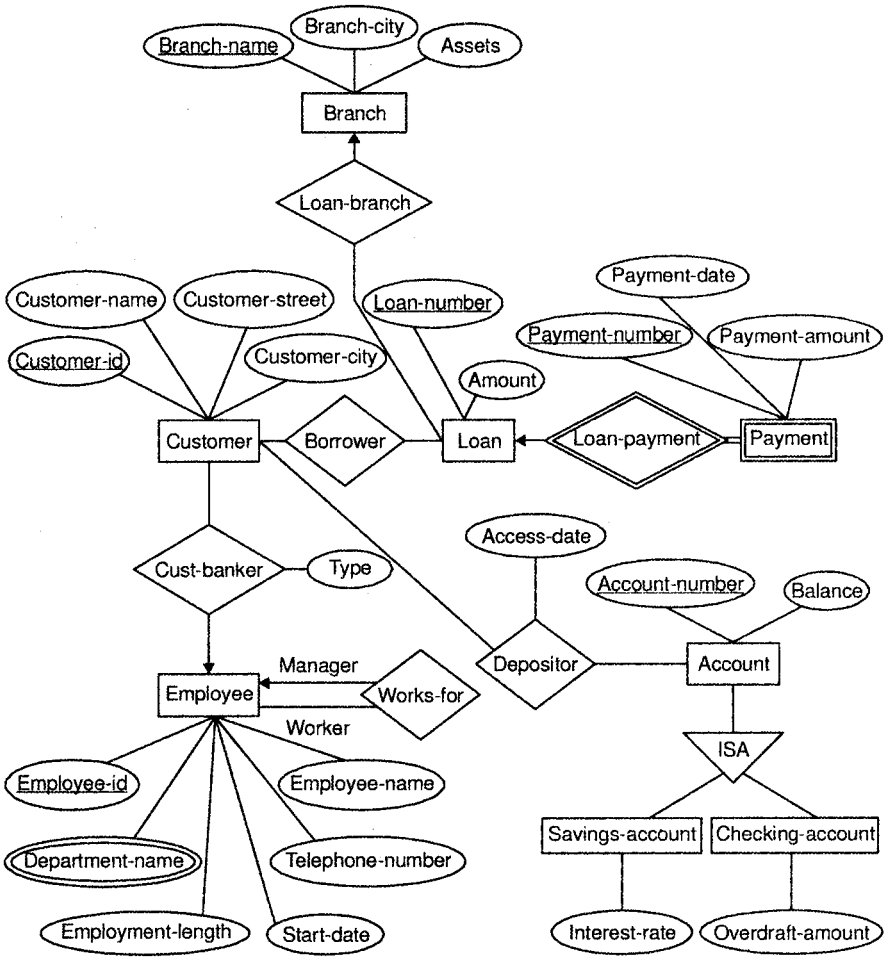


Figure 2.24: E-R diagram for the banking enterprise

Solved Case Studies

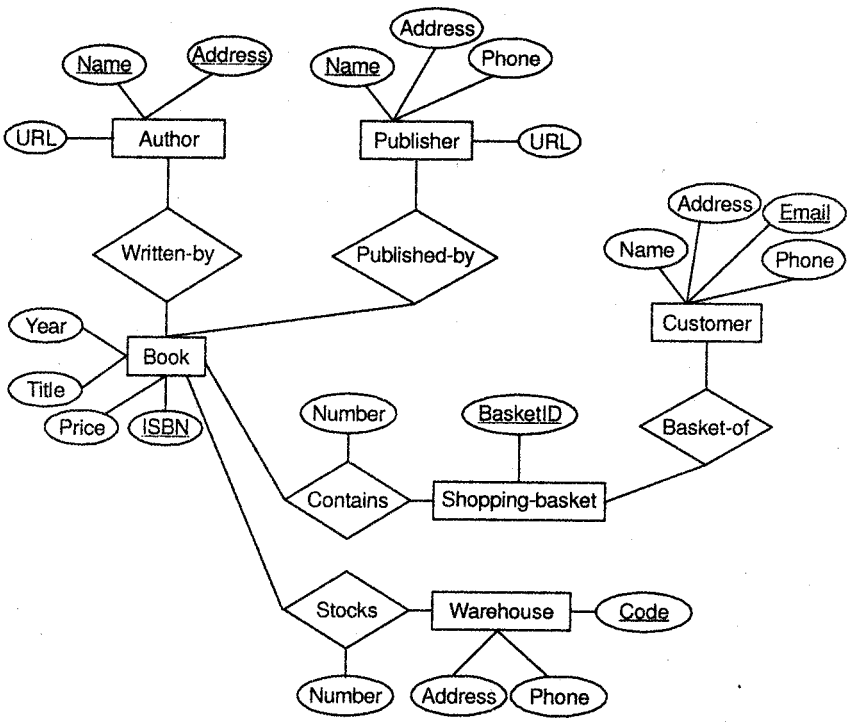
1. **A computer institute conducts lots of courses. For each course there are several batches throughout the day. Every batch has minimum and maximum capacity. The number of students admitted to a batch cannot exceed the maximum capacity and a batch cannot be started if the number of students admitted to that batch is less than the minimum capacity. Also for every batch there is a starting date and a student cannot be admitted to the batch after fifteen days of starting of the batch.**

For each course certain basic qualification is required for a student to get admission to the course. A student can select convenient batch from the batches available. Through_out the course time, the institute conducts four tests before the final examination. The final result of the student depends on the marks in these four tests and the marks in the final examination.

There are faculty members who teach these different courses. The faculty members have specialization in certain subjects and they can teach only those courses in which they have specialization. At a time he/she cannot teach more than three batches.

Draw an entity-relationship diagram.

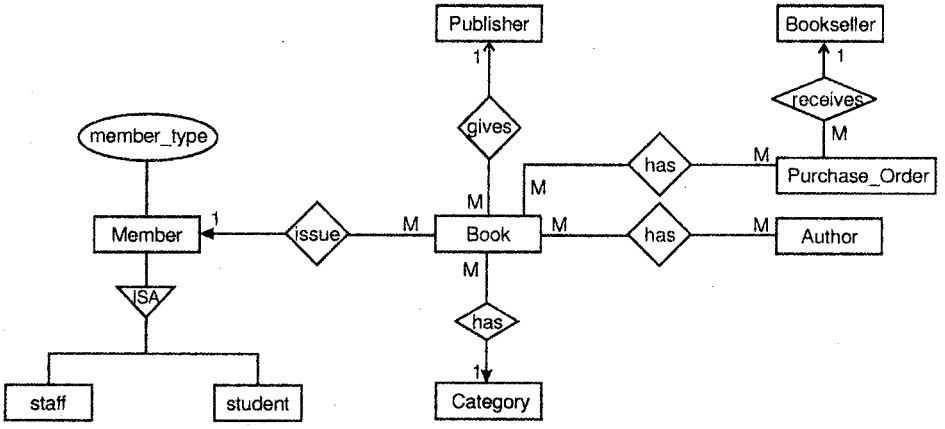
Solution



---

2. **Draw E-R diagram with key attributes for the Library: University library have many books. Books are written by different authors. Books are classified into different categories. Library purchases books from specific booksellers of different publishers. Members of Library are either staff-member or students.**

Solution



3. A company has several employees. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. A database should provide following details to the user.

Asked. Number of questions 2
Apr. 12, Oct.11 – *8M*

i. Identify all entities

ii. Identify all relations

iii. E-R Diagram

Solution

i. Identify all entities

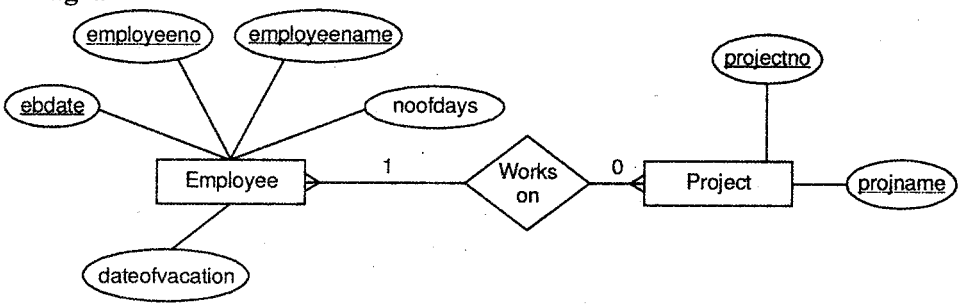
Employee

Project

ii. Identify all relations

Each project must have one or more employees.
Each employee can have 0 or more projects.
So there is many to many relationship between project and employee.
Employee(employeeno,empname,ebdate,dateofvacation, no of days)
Project(projectno,projname)
Employeeproject(employeeno,projectno)

iii. E-R Diagram



Asked. Number of questions 1
Apr. 2011 – 8M

4. A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments.

i. Identify all entities

ii. Identify all relations

iii. Draw E-R diagram

Solution

i. Identify all entities : Entities are as follows:

a. Department(dno,dname,noofemployee)

b. Employee(eno,ename,desg)

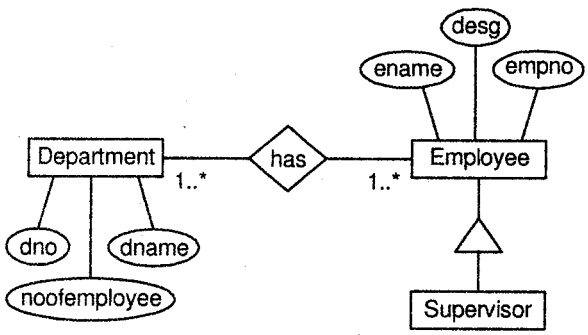
ii. Identify all relations

Department and Employee are related with many to many relationship.

So to maintain this relationship we have to create third table

Department_employee(dno,empno)

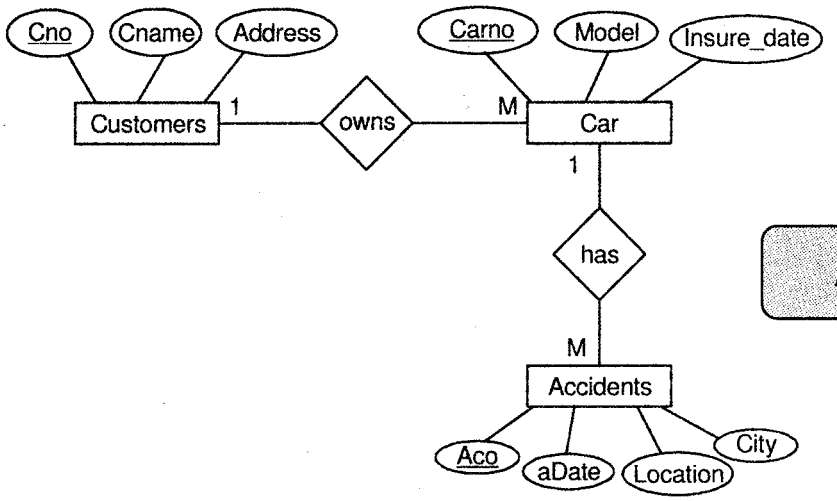
iii. E-R diagram



5. A car insurance company has a set of customers, each of whom owns one or more cars. Each car has associated with it zero to any number of record accidents.

i. Identify all entities **ii. Identify all relations**

iii. Draw E-R Diagram



Asked. Number of questions 1
Apr. 2010 – 8M

Solution

Entities are:
Customer (cno, cname, address, city, phone-no, licenceno);
Car (carno, model, insuredate, cno)
Accident (ano, carno, adate, location, city)
Relationship between entities are as follows

i. One customer has Many cars (One-to-many)
ii. One car has Many accidents (One-to-many)

---

6. 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 banks.
Database should provide the following details to user.
i. Identify all entities ii. Identify all relations iii. E-R diagram

Asked. Number of questions 1
Apr. 2010 – 8M

Solution

i. Identify all entities

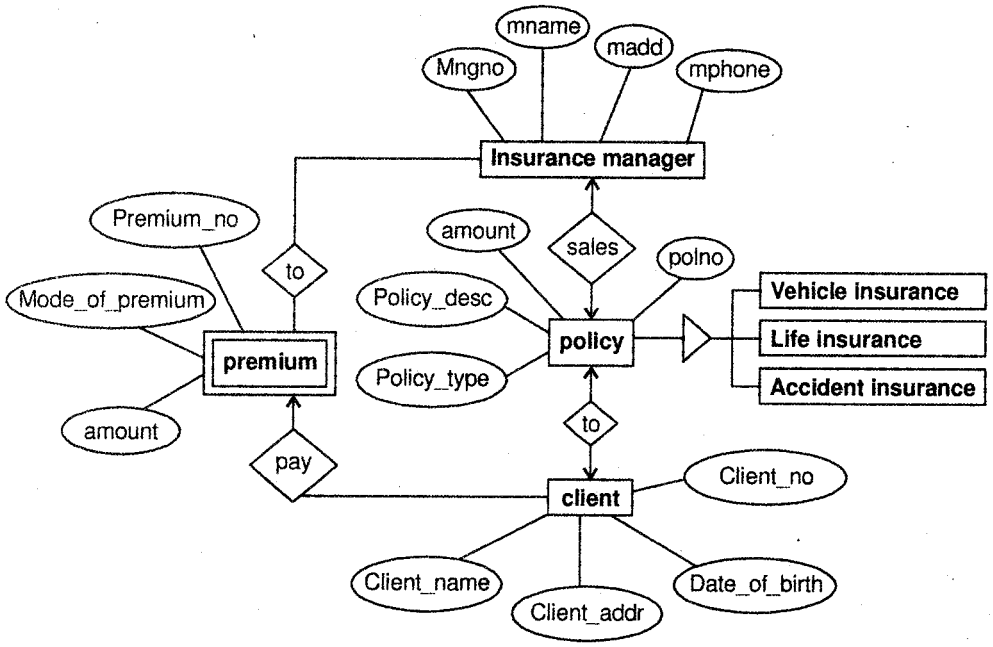
Entities are as follows:

a. Insurance Manager(mng_no,mname,madd,mphone)
b. Policies(pol_no,pol_desc,pol_type,amount)
c. Clients(client_no,client_name,client_addr,date_of_birth,client_oh);
d. Premiuns(premium_no, mode_of_premium,amount);

ii. Identify all relations

a. Client and policy are related with many to many relationships having attribute as maturity year.
b. Insurance manager and policies have many to many relationships.
c. Insurance manager and clients are related with many to many relationships.

iii. E-R diagram



Summary

- A DBMS is a complex software system consisting of a number of components. The DBMS provides users with a method of abstracting their data requirements and removes the drudgery of specifying the details of the storage and maintenance of data. DBMS propped up with the need to have more sophisticated method for storing large amount of data after encountering the various limitations of the pervious file – processing system.
- The various advantages of DBMS are centralized control, data independence, elimination of data redundancy, security enforcement etc. However there are also some disadvantages of DBMS that is problems associated with centralization, cost of software/hardware and migration, complexity of backup and recovery etc.
- A number of data representation models have been developed over the years. As in case of programming languages, one concludes that there is no one "best" choice for all applications. These models differ in their method of representing the association between entities and attributes.
- The entity – relationship data model, which is popular for high – level database design, provides a means of representing relationships between entities. An entity is an object that exists in the real world and is distinguishable from other objects. A relationship is an association among several entities.

PU Questions

1. What is data model? Explain relational model. [Oct.2012– 4M]
2. Write short note on extended features of ERD. [Oct.2012– 4M]
3. Describe the capabilities of DBMS. [Oct.2012– 4M]
4. Define: i. Relation ii. Cardinality [Oct.2012– 4M]
5. What are the drawbacks of file processing system? [Apr.2012– 4M]
6. What are the advantages of DBMS? [Apr.2012– 4M]
7. Explain Network Model. [Apr.2012– 4M]
8. Write short note on Generalization. [Apr.2012– 4M]
9. Explain Network Model with example. [Oct.11, Apr.11– 4M]
10. Differentiate between File processing system and DBMS. [Oct.2011– 4M]
11. Write short note on data independence. [Oct.2011– 4M]
12. List various Users of DBMS and specify their roles. [Apr.2011– 4M]
13. List Capabilities of Good DBMS. Explain any two of them. [Apr.2011– 4M]
14. Explain Overall Structure of Database Management System in brief. [Apr.2011– 4M]
15. What is an Attribute? Explain its types. [Apr.2011– 4M]
16. Differentiate between Specialization and Generalization with example. [Apr.2011– 4M]
17. Explain any four advantages of DBMS. [Oct.2010– 4M]
18. Explain system architecture in detail. [Oct.2010– 4M]
19. State functions of database users. [Oct.2010– 4M]
20. Write a short note on network model. [Oct.2010– 4M]
21. Explain weak entity with suitable example. [Oct.2010– 4M]
22. Enlist various users of DBMS and specify their roles. [Apr.2010– 4M]
23. Explain Dense Indexing with suitable example. [Apr.2010– 4M]
24. What are the limitations of File Processing System? [Apr.2010– 4M]
25. Enlist various users of DBMS and specify their roles. [Apr.2010– 4M]
26. What is Aggregation? Explain with example. [Apr.2010– 4M]

[Apr.10, 09– 4M] 27. Explain Hierarchical Model with example.

[Apr.2009– 4M] 28. Discuss advantages of DBMS.

[Apr.2009– 4M] 29. Enlist various users of DBMS and specify their roles.

[Apr.2009– 4M] 30. Explain Database Abstraction.

[Apr.2009– 4M] 31. Explain Data Independence.

[Oct.2012 – 8M] 32. Daulatnagr Electric Supply Company is a distributor of electricity in a small town having about 20,000 consumers. The consumers are divided into three categories. Agricultural, commercial, Domestic for whom different rates are charged. The bills are made according to the meter readings which are paid by the consumers.

i. Identify all entities.
ii. Identify all relations.
iii. Draw ERD

[Apr.12,Oct.11 – 8M] 33. A company has several employees. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. A database should provide following details to the user

i. Identify all entities ii. Identify all relations
iii. E-R Diagram

[Apr.2011 – 8M] 34. A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments.

i. Identify all entities ii. Identify all relations
iii. Draw E-R diagram

[Oct.2010 – 8M] 35. 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 banks. Database should provide the following details to user.

i. Identify all entities ii. Identify all relations
iii. E-R diagram

[Apr.2010 – 8M] 36. A car insurance company has a set of customers, each of whom owns one or more cars. Each car has associated with it zero to any number of record accidents.

i. Identify all entities ii. Identify all relations
iii. Draw E-R Diagram

Chapter 3

RELATIONAL MODEL

1. Introduction

The relational model for database is one of the most important models accepted after hierarchical and network model for its known simplicity. A relational database consists of collection of tables, each of which is assigned a unique name. A row in a table represents relationship among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of table and mathematical concept of relation, from which the relational data model takes its name.

2. Terms

This section describes the structural part of the relational model. This is achieved through a number of constructs which are illustrated in *Figure 3.1* and described below.

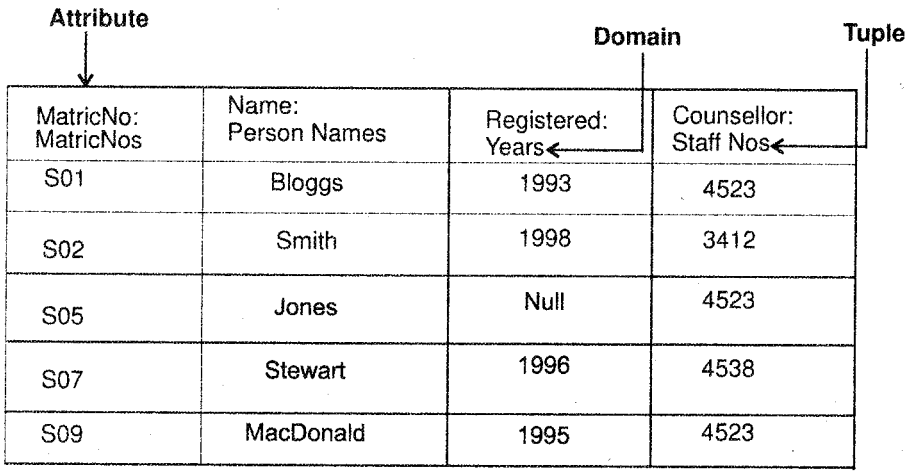


Figure 3.1

Relation: A relation comprises a set of tuples. *Figure 3.1* is a tabular representation of the Student relation containing five tuples.

Tuple: A tuple is a sequence of attributes i.e. a row in the relation table. There are five tuples shown in the Student relation in *figure 3.1* the one highlighted concerns Student identified by the MatricNo 's07'.

Attribute: An attribute is a named column in the relation table. The Student relation in *figure 3.1* contains four attributes - the MatricNo attribute is highlighted, other attributes are Name, Registered and Counsellor.

Domain: The domain construct is important as it identifies the type of an attribute. More formally the domain is a named set of values which have a common meaning.The domain of an attribute defines the set of values from which an attribute can draw

2.1 Properties of Relations

Relations have three important properties. Each relation has a name, a cardinality and a degree. These properties help us to further define and describe relations. The three properties introduced above are defined as follows:

1. **Name:** The first property is that a relation has a name which identifies it, for example the Student relation illustrated in *figure 3.1*.

2. **Cardinality:** The second property of a relation it its cardinality. This refers to the number of tuples in the relation. If we again take *figure 3.1* as our example, then the cardinality of the Student relation is 5.

3. **Degree:** The third and final property of a relation is its degree. The degree of a relation refers to the number of attributes in each tuple. Again, with reference to *figure 3.*1, the degree of the Student relation is 4, the attributes being MatricNo, Name, Registered and Counsello.

Figure 3.2 shows the *deposit* and *customer* tables for our banking *example*.

bname	account#	ename	balance
Camp	101	Anil	500
Bhagyanagar	213	Sunil	700
Vihar Kunj	102	Hemant	400
Vihar Kunj	301	Kavita	1300

ename	street	ecity
Anil	North	Pune
Sunil	East	Aurangabad
Hemant	West	Mumbai
Kavita	South	Mumbai
Suman	central	Nagpur

Figure 3.2: The deposit and customer relations

- It has four attributes.
- For each attribute there is a permitted set of values, called the **domain** of that attribute.
- *For example* the domain of *bname* is the set of all branch names.

1. Let D_1 denote the domain of *bname*, and D_2, D_3 and D_4 the remaining attributes' domains respectively.

   Then, any row of *deposit* consists of a four-tuple $(\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4)$ where

   $\upsilon_1 \in D_1, \upsilon_2 \in D_2, \upsilon_3 \in D_3, \upsilon_4 \in D_4$

   In general, *deposit* contains a **subset** of the set of all possible rows.

   That is, *deposit* is a subset of

   $D_1 \times D_2 \times D_3 \times D_4$, or abbreviated to $x_{i=1}^{4} D_i$

   In general, a table of n columns must be a subset of

   $x_{i=1}^{n} D_i$ (all possible rows)

2. Mathematicians define a relation to be a subset of a Cartesian product of a list of domains. You can see the correspondence with our tables.

   We will use the terms **relation** and **tuple** in place of **table** and **row** from now on.

3. Some more formalities:
   - Let the tuple variable t refer to a tuple of the relation r.
   - We say $t \in r$ to denote that the tuple t is in relation r.
   - Then t [*bname*] = t[1] = the value of t on the *bname* attribute.
   - So t[*bname*] = t[1] = "Downtown",
   - and t[*cname*] = t[3] = "Johnson".

Mathematicians define a relation to be a subset of a Cartesian product, of a list of domains. This definition corresponds almost exactly with our definition of table. The only difference is that we have assigned names to attributes, whereas mathematicians rely on numeric "names", using the integer 1 to denote the attribute whose domain appears first in the list of domains, 2 for the attribute whose domain appears second, and so on. Because tables are essentially relations, we shall use the mathematical terms relation and tuple in other words, a tuple variable is a variable whose domain is the set of all tuples.

▶ Database Scheme

When we talk about a database, we must differentiate between the database schema, which is the logical design of the database and the database instance, which is a snapshot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming language notion of a variable. The concept of a relation schema corresponds to the programming language.

It is convenient to give a name to a relation schema just as we give names to type definitions in programming languages. We adopt the convention of using lower case names for relations, and names beginning with an uppercase letter for relation schemes.

For example, the relation scheme for the *deposit* relation:

- *Deposit-scheme = (bname, account#, cname, balance)*

We may state that *deposit* is a relation on scheme *Deposit-scheme* by writing *deposit(Deposit-scheme).*

If we wish to specify domains, we can write:

(*bname*: string, *account#*: integer, *cname*: string, *balance*: integer).

Note that customers are identified by name. In the real world, this would not be allowed, as two or more customers might share the same name.

Figure 3.3 shows the E-R diagram for a banking enterprise.

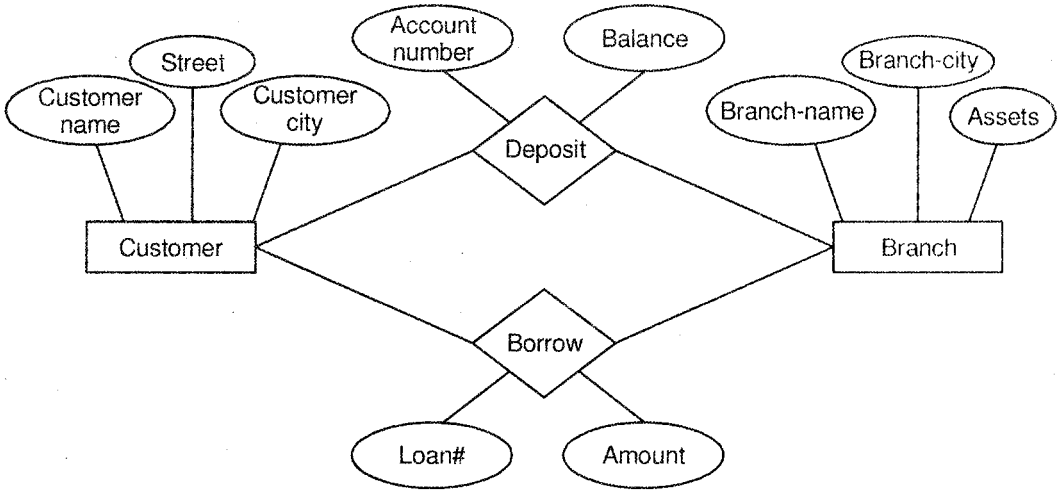


Figure 3.3: E-R diagram for the banking enterprise

The relation schemes for the banking example used throughout the text are:

- *Branch-scheme = (bname, assets, bcity)*
- *Customer-scheme = (cname, street, ccity)*
- *Deposit-scheme = (bname, account#, cname, balance)*
- *Borrow-scheme = (bname, loan#, cname, amount)*

Note: Some attributes appear in several relation schemes (e.g., *bname, cname*). This is legal, and provides a way of **relating** tuples of distinct relations.

1. Why not put all attributes in one relation?

   Suppose we use one large relation instead of *customer* and *deposit*:

   - *Account-scheme = (bname, account#, cname, balance, street, ccity)*
   - If a customer has several accounts, we must duplicate her or his address for each account.
   - If a customer has an account but no current address, we cannot build a tuple, as we have no values for the address.
   - We would have to use **null values** for these fields.

- Null values cause difficulties in the database.
- By using two separate relations, we can do this without using null values.

Queries made against the relational database, and the derived relvars in the database are expressed in a relational calculus or a relational algebra. In his original relational algebra, Dr. Codd introduced eight relational operators in two groups of four operators each. The first four operators were based on the traditional mathematical set operations:

- The union operator combines the tuples of two relations and removes all duplicate tuples from the result. The relational union operator is equivalent to the SQL UNION operator.
- The intersection operator produces the set of tuples that two relations share in common. Intersection is implemented in SQL in the form of the INTERSECT operator.
- The difference operator acts on two relations and produces the set of tuples from the first relation that do not exist in the second relation. Difference is implemented in SQL in the form of the EXCEPT or MINUS operator.

The cartesian product of two relations is a join that is not restricted by any criteria, resulting in every tuple of the first relation being matched with every tuple of the second relation. The cartesian product is implemented in SQL as the CROSS JOIN join operator.

The remaining operators proposed by Dr. Codd involve special operations specific to relational.

The selection, or restriction, operation retrieves tuples from a relation, limiting the results to only those that meet a specific criteria, i.e., a subset of terms of set theory. The SQL equivalent of selection is the SELECT query statement with a WHERE clause.

The projection operation is essentially a selection operation in which duplicate tuples are removed from the result. The SQL GROUP BY clause, or the DISTINCT keyword implemented by some SQL dialects, can be used to remove duplicates from a result set.

The join operation defined for relational databases is often referred to as a natural join. In this type of join, two relations are connected by their common attributes. SQL's approximation of a natural join is the INNER JOIN join operator.

The relational division operation is slightly more complex operation, which involves essentially using the tuples of one relation (the dividend) to partition a second relation (the divisor). The relational division operator is effectively the opposite of the cartesian product operator (hence the name).

Other operators have been introduced or proposed since Dr. Codd's introduction of the original eight including relational comparison operators and extensions that offer support for nesting and hierarchical data, among others.

3. Keys

Types of keys
i. Primary Key
ii. Unique Key
iii. Surrogate Key
iv. Foreign Key
v. Super Key
vi. Candidate Key

In relational database design, a unique key or primary key is a candidate key to uniquely identify each row in a table. A unique key or primary key comprises a single column or set of columns. No two distinct rows in a table can have the same value (or combination of values) in those columns. Depending on its design, a table may have arbitrarily many unique keys but at most one primary key.

A unique key must uniquely identify all possible rows that exist in a table and not only the currently existing rows. *Examples* of unique keys are Social Security numbers (associated with a specific person) or ISBNs (associated with a specific book). Telephone books and dictionaries cannot use names or words or Dewey Decimal system

Asked. Number of questions 2

Apr. 12, Oct.11– *4M*
What is a Key? Explain its types.

numbers as candidate keys because they do not uniquely identify telephone A primary key is a special case of unique keys. The major difference is that for unique keys the implicit NOT NULL constraint is not automatically enforced, while for primary keys it is. Thus, the values in unique key columns may or may not be NULL. Another difference is that primary keys must be defined using another syntax. numbers or words.

The relational model, as expressed through relational calculus and relational algebra, does not distinguish between primary keys and other kinds of keys. Primary keys were added to the SQL standard mainly as a convenience to the application programmer.

Unique keys as well as primary keys can be referenced by foreign keys.

3.1 Primary Keys

Primary keys are defined in the ANSI SQL Standard, through the PRIMARY KEY constraint. The syntax to add such a constraint to an existing table is defined in SQL: 2003 like this:

```
ALTER TABLE <table identifier>
ADD [CONSTRAINT <constraint identifier> ]
PRIMARY KEY (<column expression>
  {, <column expression>}... )
```

The primary key can also be specified directly during table creation. In the SQL Standard, primary keys may consist of one or multiple columns. Each column participating in the primary key is implicitly defined as NOT NULL.

Note that some DBMS require that primary key columns are explicitly marked as being NOT NULL.

```
CREATE TABLE table_name
(
      id_col   INT,
      col2     CHARACTER VARYING(20),
      ...
      CONSTRAINT tab_pk PRIMARY KEY(id_col),
      ...
);
```

If the primary key consists only of a single column, the column can be marked as such using the following syntax:

```
CREATE TABLE table_name
(
      id_col   INT   PRIMARY KEY,
      col2   CHARACTER VARYING(20),
      ...
)
```

3.2 Unique Keys

The definition of unique keys is syntactically very similar to primary keys.

```
ALTER TABLE <table identifier>
ADD [CONSTRAINT <constraint identifier>]
UNIQUE ( <column expression> {, <column expression>}... )
```

A column or a set of columns can be declared to be unique key. This constraint ensures that a value entered in the column must not be repeated. This constraint cannot be applied to the columns having LONG or LONG RAW type

Likewise, unique keys can be defined as part of the `Create table` SQL statement.

```
CREATE TABLE table_name
(    id_col    INT,
     col2      CHARACTER VARYING(20),
     key_col   SMALLINT,
       ...
     CONSTRAINT key_unique UNIQUE(key_col),
       ...
)
CREATE TABLE table_name
(  id_col   INT   PRIMARY KEY,
   col2     CHARACTER VARYING(20),
   ...
   key_col   SMALLINT UNIQUE,
   ...
)
```

3.3 Surrogate Keys

In some design situations the natural key that uniquely identifies a tuple in a relation is difficult to use for software development. For example, it may involve multiple columns or large text fields. A surrogate key can be used as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primary over the others.

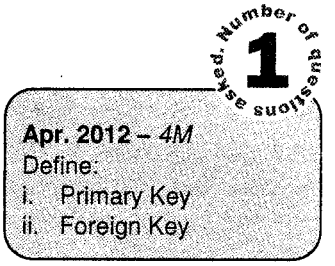
Since primary keys exist primarily as a convenience to the programmer, surrogate primary keys are often used—in many cases exclusively—in database application design.

Due to the popularity of surrogate primary keys, many developers and in some cases even theoreticians have come to regard surrogate primary keys as an inalienable part of the relational data model. This is largely due to a migration of principles from the Object-Oriented Programming model to the relational model, creating the hybrid object-relational model. In the ORM, these additional restrictions are placed on primary keys:

- Primary keys should be immutable, that is, not change until the record is destroyed.
- Primary keys should be anonymous integer or numeric identifiers.

However, neither of these restrictions are part of the relational model or any SQL standard. Due diligence should be applied when deciding on the immutability of primary key values during database and application design. Some database systems even imply that values in primary key columns cannot be changed using the UPDATE SQL statement.

3.4 Foreign Key



A foreign key is a field (or fields) that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, those only values that are supposed to appear in the database are permitted.

For example, say we have two tables, a CUSTOMER table that includes all customer data, and an ORDERS table that includes all customer orders. The constraint here is that all orders must be associated with a customer that is already in the CUSTOMER table.

In this case, we will place a foreign key on the ORDERS table and have it relate to the primary key of the CUSTOMER table. This way, we can ensure that all orders in the ORDERS table are related to a customer in the CUSTOMER table. In other words, the ORDERS table cannot contain information on a customer that is not in the CUSTOMER table.

The structure of these two tables will be as follows:

Table CUSTOMER

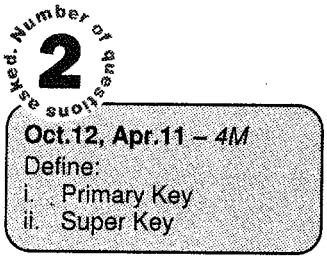
column_name	characteristic
SID	Primary Key
Last_Name	
First_Name	

Table ORDERS

Column_name	characteristic
Order_ID	Primary Key
Order_Date	
Customer_SID	Foreign Key
Amount	

In the above *example*, the Customer_SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.

3.5 Super Key



A superkey is defined in the relational model of database organization as a set of attributes of a relation variable (relvar) for which it holds that in all relations assigned to that variable there are no two distinct tuples (rows) that have the same values for the attributes in this set.

Equivalently a superkey can also be defined as a set of attributes of a relvar upon which all attributes of the relvar are functionally dependent.

Note that if attribute set K is a superkey of relvar R, then at all times it is the case that the projection of R over K has the same cardinality as R itself.

Informally, a superkey is a set of columns within a table whose values can be used to uniquely identify a row. A candidate key is a minimal set of columns necessary to identify a row, this is also called a minimal superkey. *For example*, given an employee table, consisting of the columns employeeID, name, job, and departmentID, we could use the employeeID in combination with any or all other columns of this table to uniquely identify a row in the table. *Examples* of superkeys in

this table would be {employeeID, Name}, {employeeID, Name, job}, and {employeeID, Name, job, departmentID}.

In a real database we don't need values for all of those columns to identify a row. We only need, as per our example, the set {employeeID}. This is a minimal superkey – that is, a minimal set of columns that can be used to identify a single row. So, employeeID is a candidate key.

Example

English Monarchs		
Monarch Name	Monarch Number	Royal House
Edward	II	Plantagenet
Edward	III	Plantagenet
Richard	II	Plantagenet
Henry	IV	Lancaster

In this *example*, the possible superkeys are:

- {Monarch Name, Monarch Number}
- {Monarch Name, Monarch Number, Royal House}

In the relational model, a candidate key of a relvar (relation variable) is a set of attributes of that relvar such that

1. at all times it holds in the relation assigned to that variable that there are no two distinct tuples with the same values for these attributes and
2. there is not a proper subset of this set of attributes for which (1) holds.

Since a superkey is defined as a set of attributes for which (1) holds, we can also define a candidate key as a minimal superkey, i.e., a superkey of which no proper subset is also a superkey.

3.6 Candidate Key

The importance of candidate keys is that they tell us how we can identify individual tuples in a relation. As such they are one of the most important types of database constraint that should be specified when designing a database schema. Since a relation is a set (no duplicate elements), it holds that every relation will have at least one candidate key (because the entire heading is always a superkey).

Since in some RDBMSs tables may also represent multisets (which strictly means these DBMSs are not relational), it is an important design rule to specify explicitly at least one candidate key for each

relation. For practical reasons, RDBMSs usually require that for each relation one of its candidate key is declared as the primary key, which means that it is considered as the preferred way to identify individual tuples. Foreign keys, for example, are usually required to reference such a primary key and not any of the other candidate keys.

The definition of candidate keys can be illustrated with the following (abstract) example. Consider a relation variable (relvar) R with attributes (A, B, C, D) that has only the following two legal values r1 and r2:

r1			
A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d1
a2	b1	c2	d1

r2			
A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d1
a1	b1	c2	d2

Here r2 differs from r1 only in the A and D values of the last tuple.

For r1 the following sets have the uniqueness property, i.e., there are no two tuples in the instance with the same values for the attributes in the set:

{A,B}, {A,C}, {B,C}, {A,B,C}, {A,B,D}, {A,C,D}, {B,C,D}, {A,B,C,D}

For r2 the uniqueness property holds for the following sets;

{B,D}, {C,D}, {B,C}, {A,B,C}, {A,B,D}, {A,C,D}, {B,C,D}, {A,B,C,D}

Since superkeys of a relvar are those sets of attributes that have the uniqueness property for all legal values of that relvar and because we assume that r1 and r2 are all the legal values that R can take, we can determine the set of superkeys of R by taking the intersection of the two lists:

{B,C}, {A,B,C}, {A,B,D}, {A,C,D}, {B,C,D}, {A,B,C,D}

Finally we need to select those sets for which there is no proper subset in the list, which are in this case:

{B,C}, {A,B,D}, {A,C,D}

These are indeed the candidate keys of relvar R.

We have to consider all the relations that might be assigned to a relvar to determine whether a certain set of attributes is a candidate key. For example, if we had considered only r1 then we would have concluded that {A,B} is a candidate key, which is incorrect. However, we might be able to conclude from such a relation that a certain set is not a candidate key, because that set does not have the uniqueness property (example {A,D} for r1). **Note** that the existence of a proper subset of a set that has the uniqueness property cannot in general be used as evidence that the superset is not a candidate key. In particular, note that in the case of an empty relation, every subset of the heading has the uniqueness property, including the empty set.

▶ Determining Candidate Keys

The previous *example* only illustrates the definition of candidate key and not how these are in practice determined. Since most relations have a large number or even infinitely many instances it would be impossible to determine all the sets of attributes with the uniqueness property for each instance. Instead it is easier to consider the sets of real-world entities that are represented by the relation and determine which attributes of the entities uniquely identify them. *For example,* a relation Employee(Name, Address, Dept) probably represents employees and these are likely to be uniquely identified by a combination of Name and Address which is therefore a superkey, and unless the same holds for only Name or only Address, then this combination is also a candidate key.

In order to determine correctly the candidate keys it is important to determine all superkeys, which is especially difficult if the relation represents a set of relationships rather than a set of entities. Therefore it is often useful to attempt to find any "forgotten" superkeys by also determining the functional dependencies. Consider for example the relation Marriage(Husband, Wife, Date) for which it will trivially hold that {Husband, Wife, Date} is a superkey. If we assume that a certain person can only marry once on a given date then this implies the functional dependencies{Husband, Date}→Wife and {Wife, Date}→Husband. From this then we can derive more superkeys by applying the following rule:

if S is a superkey and X→Y a functional dependency

then (S-Y)+X is also a superkey

where '-' is the set difference and '+' the set union. In this case this leads to the derivation of the superkeys {Husband, Date} and {Wife, Date}.

There are many types of integrity constraints applicable at various situations. They are as follows:

Types of integrity constraints
i. Domain constraint
ii. Referential constraint
iii. Entity integrity constraint
iv. Key constraint

i. **Domain Constraint:** For a given application an attribute is allowed to take a value from a set of permissible values. This set of allowable values for the attribute is called the domain of the attribute.

   Example: Employee ages: Possible ages of employees of a company must be a value between 18 and 60 years. Domain for the attribute age is all values between 18 to 60 years.

ii. **Referential Integrity Constraint:** The referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

   Example: Employee, Department tables with following attribute.

```
Employee (Empno, name, address, deptno)
Department (deptno, deptname)
```

   Deptno is foreign key in Employee table. So while entering tuple in employee relation, department value is compared to department tuples. If existing, insertion in employee is

allowed; if the value of dept no is not present in the department relation then system will not allow user to store the record.

iii. **Entity Integrity Constraint:** This is a specialization to the domain constraints for null values. It states that the primary key attribute(s) cannot have a null value in any tuple. As we know, the primary key value is used to identify individual tuples in a relation. If the null values are allowed it means that these tuples cannot be identified or distinguished from each other.

In SQL, when an attribute is defined as a primary key of a relation, integrity constraint is automatically applied.

iv. **Key Constraint:** It states that primary key value must be unique. It is not allowed to repeat primary key values.

4. Relational Algebra

In order to implement a DBMS, there must exist a set of rules which state how the database system will behave.

For instance, somewhere in the DBMS must be a set of statements which indicate that when someone inserts data into a row of a relation, it has the effect which the user expects.

One way to specify this is to use words to write an 'essay' as to how the DBMS will operate, but words tend to be imprecise and open to interpretation. Instead, relational databases are more usually defined using Relational Algebra.

Relational Algebra is:

- The formal description of how a relational database operates
- An interface to the data stored in the database itself
- The mathematics which underpin SQL operations

Operators in relational algebra are not necessarily the same as SQL operators, even if they have the same name. *For example*, the SELECT statement exists in SQL, and also exists in relational algebra.

These two uses of SELECT are not the same. The DBMS must take whatever SQL statements the user types in and translate them into relational algebra operations before applying them to the database.

4.1 Basic Operations

Basic Operations
Relational Algebra
i. Select Operation
ii. Project Operation
iii. Union Operation
iv. Set Difference Operation
v. Cartesian Product Operation

▶ Select Operation

The select operation selects the tuples (rows) that satisfy the given predicate (condition). The lower case Greek letter Sigma (σ) is used to represent the select operation.

The predicate appears as a subscript to σ and argument relation is given in parenthesis following σ. Predicates can be defined using the operators $=, \neq, \leq, <, >, \geq$ etc. and they may be connected by using the connectives ($\vee$) or ($\wedge$).

Examples

a. Find all tuples from player relation for which country is India.

Query: $\sigma_{\text{country = "India"}}$ (Player)

Result:

Player id	Team id	Country	Age	Runs	Wickets
101	101	India	25	10000	300
104	101	India	28	20000	200
106	101	India	22	15000	150
105	101	India	21	12000	400

b. Select all the tuples for which runs are greater than or equal to 15000.

Query: $\sigma_{\text{runs} \geq 15000}$ (Player)

Result:

Player id	Team id	Country	Age	Runs	Wickets
104	101	India	28	20000	200
106	101	India	72	15000	150

c. Select all the players whose runs are greater than 6000 and age is less than 25.

Query : $\sigma_{\text{runs} > 6000 \wedge \text{age} < 25}$ (Player)

Result:

Player id	Team id	Country	Age	Runs	Wickets
108	101	India	22	15000	150
109	103	England	24	6000	90

▶ Project Operation

Projection of a relation P(P–Schema) on the set of attributes Y $\in$ P–Schema is the projection of each tuple of the relation P on the set of attributes Y.

The projection operation is a unary operation and it returns its argument relation with certain attributes left out. It is denoted by a Greek letter pi(Π). The attributes, which appear in the result, are listed as a subscript to Π.

Examples

a. List all the countries in player relations.

Query: $\Pi_{country}$ (Player)

Result:

Country
India
Pakistan
England
Australia

b. List all the team ids and countries in table Player.

Query : $\Pi_{Team\_id, country}$ (Player)

Result :

Team id	Country
101	India
102	Pakistan
103	England
104	Australia

▶ Union Operation

Compatible relations: Two relations R and S are said to be compatible relations if they satisfy the following two conditions.

a. The relations R and S are of same entity, i.e., the number of attributes are same.

b. The domains of the i^{th} attribute of R and i^{th} attribute of S must be same for all i.

The **union** of R and S is the set theoretic union of R and S, if R and S are compatible relations.

It is denoted by ∪. The resultant relations P(P = R∪S) has tuples drawn from R and such that a tuple in P is either in R or S or in both of them.

R- Schema (id,name) R relation	
Id	**Name**
101	Raj
102	Rahul
103	Sachin
104	Anil
105	Prasad

S – Schema (id, name) S relation	
id	**Name**
101	Raj
104	Anil
106	Kapil
107	Sumit

Depositor relation	
Acc. No.	**Cust- name**
A 231	Rahul
A 432	Omkar
R 321	Sachin
S 231	Raj
T 239	Sumit

Borrower relation	
Loan no.	**Cust- name**
P – 3261	Sachin
Q – 6934	Raj
S – 4321	Ramesh
T – 6281	Anil

Examples

1. P = R ∪ S is given by the relation

Id	Name
101	Raj
102	Rahul
103	Sachin
104	Anil
105	Prasad
106	Kapil
107	Sumit

2. Find the names of customers having an account or loan.

Query: $\Pi_{\text{cust name}}$ (Depositor) ∪ $\Pi_{\text{cust name}}$ (Borrower)

Result:

Cust- name
Anil
Omkar
Rahul
Raj
Ramesh
Sachin
Sumit

▶ Set Difference Operation

The set difference operation removes common tuple from the first relation.
It is denoted by '–' sign. The expression R – S results in a relation containing those tuples in R but not in S. For set difference operation, relations must be compatible relations.

Examples

a. P = R – S is

Id	Name
106	Kapil
107	Sumit

b. Find all the customers having an account but not the loan.

Query : Π_{cust_name} (Depositor) – Π_{cust_name} (Borrower)

Result :

Cust_Name
Rahul
Omkar
Sumit

c. Find all the customers having a loan but not the account.

Query: Π_{cust_name} (Borrower) – Π_{cust_name} (Depositor)

Result :

Cust_Name
Ramesh
Anil

▶ Cartesian Product Operation

Cartesian product of two relations is the concatenation of tuples belonging to the two relations.

It is denoted by '×' sign. If R and S are two relations, R×S results in a new relation P, which contains all possible combination of tuples in R and S. For Cartesian product operation, compatible relations are not required. The schema of resultant relations is given by P – Schema = R – Schema || S – Schema

where || represents concatenation.

The degree / parity of the resultant relation is given by

|P – Schema| = |R – Schema| + |S – Schema|

The cardinality of the resultant relation is given by |P| = |R| * |S|

Examples

a. Employee–Schema = { Emp_id, Name }

Project–Schema = { Proj_name }

Employee	
Emp_id	**name**
101	Sachin
103	Rahul
104	Omkar
106	Sumit
107	Ashish

Project
Proj_Name
DBMS 1
DBMS 2

R = Employee × Project

R – Schema = {Emp_id, Name, Proj_name }

Emp– id	Name	Proj–name
101	Sachin	DBMS 1
101	Sachin	DBMS 2
103	Rahul	DBMS 1
103	Rahul	DBMS 2
104	Omkar	DBMS 1
104	Omkar	DBMS 2
106	Sumit	DBMS 1
106	Sumit	DBMS 2
107	Ashish	DBMS 1
107	Ashish	DBMS 2

If the attribute name is same in both argument relations, then that is distinguished by attaching the name of the relation from which the attribute originally came.

b. Customer schema = {cust _ id, name}

Employees Schema = {emp _id, name}

R = Customer × Employee

R – Schema = {cust _ id, customer.name, emp _ id, employee.name}.

Customer	
Cust_id	**Name**
101	Sachin
102	Rahul
103	Ramesh

Employee	
emp_id	**Name**
201	Omkar
202	Sumit
203	Ashish

Customer × Employee			
cust_id	**Customer-name**	**emp_id**	**Employee.name**
101	Sachin	201	Omkar
101	Sachin	202	Sumit
101	Sachin	203	Ashish
102	Rahul	201	Omkar
102	Rahul	202	Sumit
102	Rahul	203	Ashish
103	Ramesh	201	Omkar
103	Ramesh	202	Sumit
103	Ramesh	203	Ashish

4.2 Addition Operations

▶ Set Intersection Operation

Number of questions asked : 2

Apr. 2012 – *4M*
Explain Select and Intersection Operations of Relational Algebra with example.

Apr. 2011 – *4M*
Explain Union and Intersection Operations of relational algebra with example.

Intersection operation selects common tuples from the two relations. For set intersection operation, the two-argument relation must be compatible relation.

It is denoted by ($\cap$). If R and S are any two relations, $P = R \cap S$ has tuples drawn from R and S, such that each tuple in P is in R and S. Result of set intersection operation can also be obtained using set difference operation.

$R \cap S = R - (R - S)$

Examples

a.

R relation	
id	**Name**
101	Raj
102	Rahul
103	Sachin
104	Anil
105	Prasad

S relation	
id	**Name**
101	Raj
104	Anil
106	Kapil
107	Sumit

$P = R \cap S$

Id	Name
101	Raj
104	Anil

b. Find the names of customers having an account and loan.

Query: Π_{cust_name} (Depositor) $\cap$ Π_{cust_name} (Borrower)

Result :

Cust_name
Sachin
Raj

▶ Division Operation

Division operation is denoted by ÷ sign. It is useful in queries, which involve the phrase "for all objects having all the specified properties".

Let R (R – Schema) and S (S – Schema) be relations and let S – Schema $\subseteq$ R–Schema, i.e., any attribute of S–Schema is also in R–Schema. The relation R ÷ S is a relation on schema R–Schema – S–Schema, i.e., on the schema containing all the attributes of Schema R that are not in Schema S. A tuple t is in r ÷ s if and only if both the conditions hold.

a. t is in $\Pi_{R\text{–Schema} - S\_\text{Schema}}$ (R)

b. For every tuple t_S in S, there is a tuple t_r in R satisfying both of the following:
(i) $t_r[s] = t_s[s]$ (ii) $t[R - S] = t$

A	B
A_1	B_1
A_1	B_2
A_2	B_1
A_3	B_1
A_4	B_2
A_5	B_1
A_5	B_2

i.

Q

B
B_1
B_2

R = P ÷ Q

A
A_1
A_5

ii.

Q

B
B_1

R = P ÷Q

A
A_1
A_2
A_3
A_5

iii.

Q

B

R = P ÷Q

A
A_1
A_2
A_3
A_4
A_5

▶ Assignment Operation

Assignment operation is denoted by '←'

Relational variable ← Expression

The result of the expression to the right hand side of ← is assigned to relation variable on the left side of ← . The relation variable may be used in subsequent expressions

$R_1 \leftarrow \Pi_{nname}$ (Customer)

$R_2 \leftarrow \Pi_{nname}$ (Employee). $R = R_1 - R_2$

▶ JOIN Operator

JOIN is used to combine related tuples from two relations:

- In its simplest form the JOIN operator is just the cross product of the two relations.
- As the join becomes more complex, tuples are removed within the cross product to make the result of the join more meaningful.
- JOIN allows you to evaluate a join condition between the attributes of the relations on which the join is undertaken.

The notation used is

$R\ JOIN_{join\ condition}\ S$

Join Example

R

Col A	Col B
A	1
B	2
D	3
F	4
E	5

R JOIN R.Col A = S.S. Col A

A	1	A	1
D	3	D	3
E	5	E	4

S

Col A	SCol B
A	1
C	2
D	3
E	4

R JOIN R. Col B = S.Scol B

A	1	A	1
B	2	C	2
D	3	D	3
F	4	E	4

Figure 3.4: JOIN

Natural Join

Invariably the JOIN involves an equality test, and thus is often described as an equi-join. Such joins result in two attributes in the resulting relation having exactly the same value. A `natural join' will remove the duplicate attribute(s).

- In most systems a natural join will require that the attributes have the same name to identify the attribute(s) to be used in the join. This may require a renaming mechanism.
- If you do use natural joins make sure that the relations do not have two attributes with the same name by accident.

Outer Joins

Notice that much of the data is lost when applying a join to two relations. In some cases this lost data might hold useful information. An outer join retains the information that would have been lost from the tables, replacing missing data with nulls.

There are three forms of the outer join, depending on which data is to be kept.

i. **Left Outer Join**: The left outer join takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join. Keep data from the left-hand table.

ii. **Right Outer Join**: The right outer join is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join.

Keep data from the right-hand table.

iii. **Full Outer Join:** The full outer join does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.

Keep data from both tables.

Outer Join example 1

R

ColA	ColB
A	1
B	2
D	3
F	4
E	5

R LEFT OUTER JOIN R. ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-

S

SColA	SColB
A	1
C	2
D	3
E	4

R RIGHT OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
-		C	2

Outer Join example 2

R

ColA	ColB
A	1
B	2
D	3
F	4
E	5

R FULL OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-
-	-	C	2

S

SColA	SColB
A	1
C	2
D	3
E	4

Consider the following SQL to find which departments have had employees on the `Further Accounting' course.

```
SELECT DISTINCT dname
FROM  department, course, empcourse, employee
WHERE cname = 'Further Accounting'
AND course.courseno = empcourse.courseno
AND empcourse.empno = employee.empno
AND employee.depno = department.depno;
```

The equivalent relational algebra is

```
PROJECTdname (department JOINdepno = depno
(
  PROJECTdepno (employee JOINempno = empno
  (
    PROJECTempno (empcourse JOINcourseno = courseno
    (
      PROJECTcourseno (SELECTcname = 'Further Accounting'
      course)
    )
  )

)
```

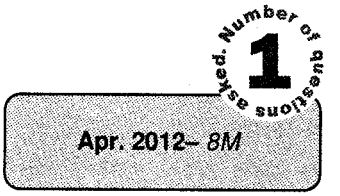
Solved Examples

1. Consider the following Relational Database:

Customer (cno, cname, city)

Quotation (qno, qdate, description, amt_quoted, cno)

Customer and Quotation are related with one–to–many relationship.



Write Relational Algebraic Expression for the following:

i. Display customer names having quotation for 'LCD'.

ii. List all the customers 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 Rs. 15,000.

Solution

i. Display customer names having quotation for 'LCD'.

$$\Pi_{cname}\ (\sigma_{description='LCD'}\ ^{(Customer \bowtie Quotation)})$$

ii. List all the customers bearing quotation dated '20-May-2010'.

$$\Pi_{cname}\ (\sigma_{qdate='20\text{-}May\text{-}2010'}\ ^{(Customer \bowtie Quotation)})$$

iii. List all the customers who live in 'M.P.' or 'U.P.'

$$\Pi_{cname}\ (\sigma_{city='M.P.'\ \vee\ 'U.P'.}\ ^{(Customer)})$$

iv. Display customers of amt_quoted as Rs. 15,000.

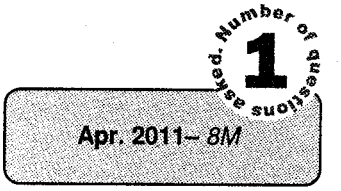
$$\Pi_{cname}\ (\sigma_{amt\_quoted\ >15000}\ ^{(Customer \bowtie Quotation)})$$

2. Consider the following Relational Database:

Customer (cno, cname, city)

Quotation (quot_no, q_date, description, amt_quoted, cno)

Customer and Quotation are related with one to many relationships.



Write Relational Algebraic Expression for the following:

i. List all the Customers who live in 'Mumbai' or 'Pune'.

ii. Display customer names having quotation for 'Desktop'.

iii. Display customers of amt_quoted as Rs. 10,000.

iv. List all the customers bearing quotation dated '1-1-10'.

Solution

i. $\Pi_{cname}\ (\sigma_{city='Mumbai'}\ \vee\ \sigma_{city='Mumbai'}\ ^{(customer)})$

ii. $\Pi_{cname}(\sigma_{description='Desktop'}{}^{(customer \bowtie quotation)})$

iii. $\Pi_{cname}(\sigma_{amt\_quoted=10000}{}^{(customer \bowtie quotation)})$

iv. $\Pi_{cname}(\sigma_{q\_date='1/1/10'}{}^{(customer \bowtie quotation)})$

Asked. Number of questions 1
Oct. 2010– 8M

3. Consider the following Relational Database:

Department(dept_no,dept_name,location);

Employee(e_no,ename,addr,salary,designation,dept_no);

Department and Employee are related with one to many relationship.

Construct queries into Relational Algebra:

i. List all employees who are working as 'Manager'.

ii. List all employees whose salary is greater than 10,000 and less than 25,000.

iii. List all employee details working in 'Accounts' of 'Pune' city.

iv. Display Department Name and Employee Name working in 'Sales' or 'Inventory'.

Solution

i. $\Pi_{ename}(\sigma_{designation='manager'})^{(employee \bowtie department)}$

ii. $\Pi_{ename}(\sigma_{salary>10,000 \wedge salary<25,000})^{(employee)}$

iii. $\Pi_{e\_no,ename,addr,salary,designation,dept\_no}(\sigma_{dept\_name='accounts' \wedge location='pune'})$ $(employee \bowtie department)$

iv. $\Pi_{dept\_name,e\_name}(\sigma_{dept\_name='sales' \vee dept\_name='inventory'})^{(employee \bowtie department)}$

Asked. Number of questions 1
Apr. 2010 – 8M

4. Consider Relational Database:

Customer (cust_no, cust_name, address, city)

Loan (loan_no, loan_amt, loan_date; cust_no)

Customer and Loan are related with one-many relationship.

Write relational algebraic expression for the following:

i. List loan details of customer name as 'Mr. Khurana'.

ii. Display customers with loan amount greater than 50,000.

iii. Display customer names who have taken loan on '10-Mar-2009' and city as 'Pune'.

iv. List names of customers who do not have loan at the bank.

Solution

i. $\Pi_{loan\_amt,\ loan\_date}\ (\sigma_{custname\ =\ "Mr.\ Khurana"}\ (customer \bowtie loan))$

ii. $\Pi_{custname}\ (\sigma_{loan\_amt\ >\ 50000}\ (customer \bowtie loan))$

iii. $\Pi_{custname}\ (\sigma_{loan\_date\ =\ "10\text{-}Mar\text{-}2009"\ \wedge\ city\ =\ "pune"}\ (customer \bowtie loan))$

iv. $\Pi_{cust\_no,\ custname}\ (customer) - \Pi_{cust\_no}\ (loan)$

Summary

Conceptually, a relation can be represented as a table; each column of the table represents an attribute of the relation and each row represents a tuple of the relation. Mathematically a relation is a correspondence between a numbers of sets and is a subset of Cartesian product of these sets. The sets are the domain of the attributes of the relation.

Duplicate tuples are not permitted in a relation. Each tuple is identified uniquely using a subset of attributes of the relation. Such a minimum subset is called a key (primary) of the relation. The unique identification property of the key is used to capture relationship between entities. Such a relationship is represented by a relation that contains a key for each entity involved in the relationship.

Relational algebra is a procedural manipulation language. It specifies the operations and order in which they are to be performed on tuples of relations. The result of this operation is also a relation. The algebraic relation operations are union, difference, Cartesian product, intersection, projection, selection, join and division.

Key Terms

- **Attribute:** A character of an entity or object. An attribute has a name and a data type.
- **Attribute Domain:** Used to organize and describe an attribute's set of possible values
- **Candidate Key:** A minimal superkey, that is, one that does not contain a subset of attributes that is itself a superkey.
- **Composite(bridge) Entity:** An entity designed to transform an M: N relationship into two 1:M relationships. The composite entity's primary key comprises at least the primary keys of the entities that it connects.
- **Composite Key:** Multiple-attribute keys. May be further subdivided.
- **Domain:** Used to organize and describe an attributes set of possible values.
- **Entity:** Something about which you want to store data; typically a person, place, thing, concept or event.
- **Entity Integrity:** The absence of null "values" in a primary key. Guarantees that each entity will have a unique identity.
- **Entity Set:** A grouping of related entities.
- **EquiJOIN:** A join operator that links tables based on an equality condition that compares specified columns of the tables.

- **Foreign Key:** An attribute (or combination of attributes) in one table whose values must match the primary key in another table or whose values must be null.
- **Index:** A pointing device that does for a database table what a book index does for a book.
- **Index Key:** The index (file) is composed of a reference value- the index key- and a set of pointers.
- **Join Column(s):** Columns with common values and attributes. JOIN is one of eight relational algebra functions and allows information from two or more tables to be combined. Join is the real power behind the relational database, allowing the use of independent tables linked by common attributes.
- **Key:** An entity identifier based on the concept of functional dependence. May be classified as Super key, Candidate key, Primary key, Secondary key, and Foreign key.
- **Key Attribute:** Create the entity's primary key.
- **Linking Table:** A table that implements a composite entity.
- **Natural JOIN:** A relational function that links tables by selecting only the rows with common values in their common attribute(s).
- **NULL:** The absence of an attribute value. Note: a null is not a blank.

PU Questions

[Oct.2012 – 4M]	1.	Define: i. Primary Key ii. Super Key
[Apr.12, Oct.11 – 4M]	2.	What is a Key? Explain its types.
[Apr.12, 11 – 4M]	3.	Define: i. Primary Key ii. Foreign Key
[Apr.2012 – 4M]	4.	Explain Select and Intersection Operations of Relational Algebra with example.
[Oct.2009 – 4M]	5.	Explain Union and Intersection Operations of relational algebra with example.
[Oct.2009 – 4M]	6.	Define the following: i. Super Key ii. Relation
[Oct.2009 – 4M]	7.	What are the select and intersection operations in relational algebra? Discuss with suitable example.
[Oct.2010 – 4M]	8.	Explain Project and Union Operation of Relational Algebra with example.
[Oct.2010 – 4M]	9.	Explain Referential Integrity with suitable example.
[Apr.2011 – 4M]	10.	Define: i. Foreign Key ii. Domain
[Apr.2010 – 4M]	11.	What are Select and Project Operations in Relational Algebra? Discuss with the help of example.

12. Consider relational database **[Oct.2012 – 8M]**
Machine(mno,mname,mtype,mcost)
Part(pno,pname,pdesc,mno)
Machine and part are related with one-many relationship.
Write relational algebraic expression for the following:
i. List all machines having coast > 5,000
ii. Display the names of all machines having parts 'wheel'.
iii. List all the parts of 'Television' machine.
iv. Display the machine names whose coast is between 50,000 to 70,000

13. Consider the following Relational Database: **[Apr.2012 – 8M]**
Customer (cno, cname, city)
Quotation (qno, qdate, description, amt_quoted, cno)
Customer and Quotation are related with one–to–many relationship.
Write Relational Algebraic Expression for the following:
i. Display customer names having quotation for 'LCD'.
ii. List all the customers 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 Rs. 15,000.

14. Consider the following relational database. **[Oct.2011 – 8M]**
Student(roll_no, name, city,marks, c_no)
Course(c_no,cname,fees)
Construct queries into relational algebra:
i. List student details enrollled for 'BBA' course.
ii. List the course having fees < 20,000.
iii. Display all students living in either 'Nasik' or 'Pune' city.
iv. Display course detail for student 'Gaurav Sharma'

15. Consider the following Relational Database: **[Apr.2011 – 8M]**
Customer (cno, cname, city)
Quotation (quot_no, q_date, description, amt_quoted, cno)
Customer and Quotation are related with one to many relationships.
Write Relational Algebraic Expression for the following:
i. List all the Customers who live in 'Mumbai' or 'Pune'.
ii. Display customer names having quotation for 'Desktop'.
iii. Display customers of amt_quoted as Rs. 10,000.
iv. List all the customers bearing quotation dated '1-1-10'. **[Apr.2011 – 8M]**

16. Consider the following Relational Database:
Customer (cno, cname, city)
Quotation (qno, qdate, description, amt_quoted, cno)

Customer and Quotation are related with one–to–many relationship. Write Relational Algebraic Expression for the following:

i. Display customer names having quotation for 'LCD'.
ii. List all the customers 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 Rs. 15,000.

[Oct.2010 – *8M*] 17. Consider the following Relational Database:
Department(dept_no,dept_name,location);
Employee(e_no,ename,addr,salary,designation,dept_no);
Department and Employee are related with one to many relationship.Construct queries into Relational Algebra:

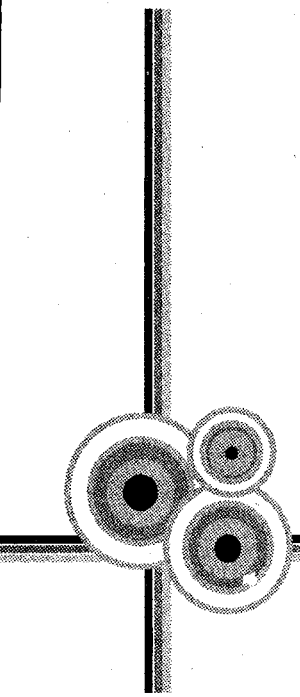
i. List all employees who are working as 'Manager'.
ii. List all employees whose salary is greater than 10,000 and less than 25,000.
iii. List all employee details working in 'Accounts' of 'Pune' city.
iv. Display Department Name and Employee Name working in 'Sales' or 'Inventory'.

[Apr.2010 – *8M*] 18. Consider Relational Database:
Customer (cust_no, cust_name, address, city)
Loan (loan_no, loan_amt, loan_date; cust_no)
Customer and Loan are related with one-many relationship.
Write relational algebraic expression for the following:

i. List loan details of customer name as 'Mr. Khurana'.
ii. Display customers with loan amount greater than 50,000.
iii. Display customer names who have taken loan on '10-Mar-2009' and city as 'Pune'.
iv. List names of customers who do not have loan at the bank.

[Oct.2009 – *8M*] 19. Consider following Relational Database:
Doctor(dno, dname, address, dcity)
Hospital(hno, hname, street, hcity)
Dochosp(dno, hno, date)
Construct queries into relatcional algebra

i. Find hospital names to which 'Dr. Mehata' has visited.
ii. Find out all the doctors who have visited hospitals in the same city.
iii. List all the doctors who visited 'Krishna' on '1-1-02'.
iv. List names of hospital to which 'Dr. Aman' has visited on '5-11-02'.



Chapter 4

SQL(STRUCTURED QUERY LANGUAGE)

1. A Brief History of Databases

Modern databases emerged in the 1960s thanks to research at IBM, among other companies. The research mainly centered around office automation, in particular automating data storage and indexing tasks that previously required a great deal of manual labor. Computing power and storage had become much cheaper, making the use of computers for data indexing and storage a viable solution.

A pioneer in the database field was Charles W. Bachman, who received the Turing Award in 1973 for pioneering work in database technology. In 1970, an IBM researcher named Ted Codd published the first article on relational databases.

Although IBM was a leader in database research, Honeywell Information Systems, Inc., released a commercial product in 1976 based on the same principles as the IBM information system, but it was designed and implemented separately from IBM's work.

In the early 1980s, the first database systems built upon the SQL standard appeared from companies such as Oracle, with Oracle Version 2, and later SQL/DS from IBM, as well as a host of other systems from other companies.

Now that you have a brief idea of where databases came from, you can turn to the more practical task of what databases are and why and when to use them. SQL is a standard computer language for accessing and manipulating databases.

2. Structured Query Language (SQL)

Structured Query Language (SQL) originated with the System R project in 1974 at IBM's San Jose Research Center.

The purpose of this project was to validate the feasibility of the relational model and to implement a DBMS based on this model. The results of this project are well documented in the database literature. In addition to contributing to the concept of query compilation and optimization and concurrency control mechanisms, the most salient result of this research project was the development of SQL.

The System R project, concluded in 1979, was followed by the release of a number of commercial relational DBMS products from IBM.

The first of these was SQL/DS for IBM's mid-range computer. Subsequently, DB2 was released for IBM's mainframe systems.

SQL (the original version was called SEQUEL and a predecessor of SEQUEL was named SQUARE) was the data definition and manipulation language for System R.

SQL has emerged as the standard query language for relational DBMSs, and most of the commercial relational database management system use SQL or a variant of SQL.

The first questions to ask are what SQL is and how do you use it with databases? SQL has three main roles:

- Creating a database and defining its structure
- Querying the database to obtain the data necessary to answer questions
- Controlling database security

Defining database structure includes creating new database tables and fields, setting up rules for data entry, and so on, which is expressed by a SQL sublanguage called Data Control Language (DCL). The next section discusses querying the database.

Finally, DCL deals with database security. Generally, database security is something that database administrators handle.

Creating SQL every time you want to change the database structure or security sounds like hard work, and it is! Most modern database systems allow you to execute changes via a user-friendly interface without a single line of SQL.

What is SQL?

- SQL stands for **S**tructured **Q**uery **L**anguage
- SQL allows you to access a database
- SQL is an ANSI standard computer language
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn

3. SQL is a Standard

SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems.

SQL statements are used to retrieve and update data in a database.

SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

Unfortunately, there are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

3.1 SQL Data Definition Language (DDL)

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables.

Apr. 2011 – *4M*
List various DDL commands. Explain any one with example.

The most important DDL statements in SQL are:

- **CREATE TABLE** - creates a new database table
- **ALTER TABLE** - alters (changes) a database table
- **DROP TABLE** - deletes a database table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

▶ SQL Create Database, Table, and Index

Create a Database

To create a database:

```
CREATE  DATABAS database_name
```

Example

```
SQL> CREATE DATABASE Student
```

Create a Table

To create a table in a database:

```
CREATE TABLE table_name
(
  column_name1 data_type,
  column_name2 data_type,
  .......
)
```

Example

This example demonstrates how you can create a table named "Person", with four columns. The column names will be "LastName", "FirstName", "Address", and "Age":

```
CREATE TABLE Person
(
  LastName varchar2(10),
  FirstName varchar,
  Address varchar2(20),
  Age int;
)
```

Create Index

Indices are created in an existing table to locate rows more quickly and efficiently. It is possible to create an index on one or more columns of a table, and each index is given a name. The users cannot see the indexes; they are just used to speed up queries.

Note: Updating a table containing indexes takes more time than updating a table without, this is because the indexes also need an update. So, it is a good idea to create indexes only on columns that are often used for a search.

A Unique Index

Creates a unique index on a table. A unique index means that two rows cannot have the same index value.

```
CREATE UNIQUE INDEX index_name
ON table_name(column_name);
```

The "column_name" specifies the column you want indexed.

A Simple Index

Creates a simple index on a table. When the UNIQUE keyword is omitted, duplicate values are allowed.

```
CREATE INDEX index_name
ON
table_name(column_name);
```

The "column_name" specifies the column you want to be indexed.

Example

This *example* creates a simple index, named "PersonIndex", on the LastName field of the Person table:

```
CREATE INDEX PersonIndex
ON Person (LastName);
```

If you want to index the values in a column in descending order, you can add the reserved word DESC after the column name:

```
CREATE INDEX PersonIndex
ON Person (LastName DESC);
```

If you want to index more than one column you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName);
```

▶ Sql Alter Table

The ALTER TABLE statement is used to add or drop columns in an existing table.

i.
```
ALTER TABLE table_name
ADD column_name datatype;
```

ii.
```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Note: Some database systems don't allow the dropping of a column in a database table (DROP COLUMN column_name).

Person

LastName	FirstName	Address
Waghmare	Vikas	Varje

Example

To add a column named "City" in the "Person" table:

```
ALTER TABLE Person ADD City
varchar2(30);
```

Result

LastName	FirstName	Address	City
Waghmare	Vikas	Varje	

Example

To drop the "Address" column in the "Person" table:

```
ALTER TABLE Person DROP COLUMN Address
```

Result

LastName	FirstName	City
Waghmare	Vikas	

▶ SQL Drop Index, Table and Database

Drop Index

You can delete an existing index in a table with the DROP INDEX statement.

Syntax for Microsoft SQLJet (and Microsoft Access):

```
DROP INDEX index_name ON
table_name
```

Syntax for MS SQL Server:

```
DROP INDEX
table_name.index_name
```

Syntax for IBM DB2 and Oracle:

```
DROP INDEX index_name
```

Syntax for MySQL:

```
ALTER TABLE table_name DROP INDEX
index_name
```

Delete a Table or Database

To delete a table (the table structure, attributes, and indexes will also be deleted):

```
DROP TABLE table_name
```

To delete a database

```
DROP DATABASE database_name
```

Truncate a Table

What if we only want to get rid of the data inside a table, and not the table itself? Use the TRUNCATE TABLE command (deletes only the data inside the table):

```
TRUNCATE TABLE table_name
```

3.2 SQL Data Manipulation Language (DML)

SQL (Structured Query Language) has syntax for executing queries. But the SQL language also includes syntax to update, insert, and delete records.

These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- **INSERT INTO**: inserts new data into a database table
- **SELECT**: extracts data from a database table
- **UPDATE**: updates data in a database table
- **DELETE**: deletes data from a database table

Number of questions asked: 1

Oct. 2011 – *4M*
List various DML commands. Explain any one with example.

▶ SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new rows into a table.

Syntax

```
INSERT INTO table_name VALUES (value1,value2,....)
```

You can also specify the columns for which you want to insert data:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Example: Insert a New Row

"Persons" table:

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune

SQL statement:

```
INSERT INTO Persons
VALUES ('Gaikwad', 'Vikas', 'Wagholi', Pune')
```

Will give this result:

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune
Gaikwad	Vikas	Wagholi	Pune

Insert Data in Specified Columns

"Persons" table:

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune
Gaikwad	Vikas	Wagholi	Pune

And This SQL statement:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Kulkarni', 'Katraj')
```

Will give this result:

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune
Gaikwad	Vikas	Wagholi	Pune
Kulkarni		Katraj	

▶ The SQL SELECT Statement

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

Syntax

```
SELECT column_name(s) FROM table_name
```

Note: SQL statements are not case sensitive. SELECT is the same as select.

Example

To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

```
SELECT LastName,FirstName FROM Persons
```

The database table "Persons":

LastName	FirstName	Address	City
Rathod	Raj	Katraj	Pune
Jadhav	Shree	CIDCO	Aurangabad
Sen	Vijay	Anand Nagar	Beed

Result

LastName	FirstName
Rathod	Raj
Jadhav	Shree
Sen	Vijay

Select All Columns

To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

```
SELECT * FROM Persons
```

Result

LastName	FirstName	Address	City
Rathod	Raj	Katraj	Pune
Jadhav	Shree	CIDCO	Aurangabad
Sen	Vijay	Anand Nagar	Beed

Condition specification (WHERE clause)

To select Persons from the "Persons" table where persons city is Pune, use a WHERE clause after SELECT statement, like this:

```
SELECT * FROM
TableName WHERE
Condition
```

Example

SELECT * FROM Person WHERE City= 'Pune'

Result

LastName	FirstName	Address	City
Rathod	Raj	Katraj	Pune

The SELECT DISTINCT Statement

The DISTINCT keyword is used to return only distinct (different) values.

The SELECT statement returns information from table columns. But what if we only want to select distinct elements?

With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement:

Syntax

```
SELECT DISTINCT column_name(s) FROM table_name
```

Example

To select ALL values from the column named "Company" we use a SELECT statement like this:

```
SELECT Company FROM Orders
```

"Orders" table

Company	OrderNumber
S1	3412
IBM	2312
Texas	4678
Texas	6798

Result

Company
S1
IBM
Texas
Texas

Note that "Texas" is listed twice in the result-set.

To select only DIFFERENT values from the column named "Company" we use a SELECT DISTINCT statement like this:

```
SELECT DISTINCT Company FROM Orders
```

Result

Company
S1
IBM
Texas

Now "Texas" is listed only once in the result-set.

The SELECT INTO Statement

The SELECT INTO statement is most often used to create backup copies of tables or for archiving records.

Syntax

```
SELECT column_name(s) INTO newtable [IN externaldatabase]
FROM source
```

Example

The following *example* makes a backup copy of the "Persons" table:

```
SELECT * INTO Persons_backup FROM Persons
```

The IN clause can be used to copy tables into another database:

```
SELECT Persons.* INTO Persons IN 'Backup.mdb' FROM Persons
```

If you only want to copy a few fields, you can do so by listing them after the SELECT statement:

```
SELECT LastName,FirstName INTO Persons_backup FROM Persons
```

You can also add a WHERE clause. The following *example* creates a "Persons_backup" table with two columns (FirstName and LastName) by extracting the persons who lives in " Washington " from the "Persons" table:

```
SELECT LastName,Firstname INTO Persons_backup FROM Persons WHERE
city='Washington'
```

Selecting data from more than one table is also possible. The following *example* creates a new table "Empl_Ord_backup" that contains data from the two tables Employees and Orders:

```
SELECT Employees.Name,Orders.Product
INTO Empl_Ord_backup
FROM Employees
INNER JOIN Orders
ON
Employees.Employee_ID=Orders.Employee_ID
```

▶ ORDER BY

Sort the rows
The ORDER BY clause is used to sort the rows.

Orders

Company	OrderNumber
Satyam	3412
Armstrong	5678
WNS	6798
WNS	2312

Example

To display the company names in alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

Result

Company	OrderNumber
Armstrong	5678
Satyam	3412
WNS	6798
WNS	2312

Example

To display the company names in alphabetical order AND the OrderNumber in numerical order:

```
SELECT Company, OrderNumber FROM Orders ORDER BY Company,
OrderNumber
```

Result

Company	OrderNumber
Armstrong	5678
Satyam	3412
WNS	2312
WNS	6798

Example

To display the company names in reverse alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC
```

Result

Company	OrderNumber
WNS	6798
WNS	2312
Satyam	3412
Armstrong	5678

Example

To display the company names in reverse alphabetical order AND the OrderNumber in numerical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC, OrderNumber ASC
```

Result

Company	OrderNumber
WNS	2312
WNS	6798
Satyam	3412
Armstrong	5678

Notice that there are two equal company names (WNS) in the result above. The only time you will see the second column in ASC order would be when there are duplicated values in the first sort column, or a handful of nulls.

▶ JOIN Statement

SQL does not have a direct representation of the join operation. However, the type of join can be specified by an appropriate predicate in the where clause of a select statement, wherein the relations to be joined are specified in the from clause. The join is performed by using the appropriate tuples of the participating relations, followed by selection and projection. Consider the following SQL statement. The relation name precedes the attribute name, the two being separated by a period. This method of qualifying is used to distinguish identical attribute names.

```
Select T1.a11,t1.a12…..,T1.a1n, T2.a11,t2.a12…..,T2.a1n
From T1,T2
Where T1.a1j=T2.a2k
```

This statement is evaluated by performing a Cartesian product of the tables T1, T2, and tuples satisfying the where clause are selected.

Example

Retrive the order numbers, client names and their order dates from the client_master and sales_order tables.

The order date should be displayed in 'DD/MM/YY' format and sorted in ascending order.

Table name: sales_order

order_no	client_no	order_date
100	C115	12-Apr-96
101	C114	23-Jun-96
102	C111	01_Aug-96

Table name: client_master

client_no	name	bal_due
C111	Vijay	500
C112	Sachin	100
C113	Ramesh	0
C114	Prasad	600
C115	Kiran	50

SELECT order_no, name, to_char(order_date, "DD/MM/YY") "Order Date" FROM sales_order, client_master WHERE client_master.client_no = sales_order.client_no ORDER BY to_char(order_date, "DD/MM/YY");

Result

order_no	client_no	order_date
102	C111	12-Apr-96
101	C114	23-Jun-96
100	C115	01_Aug-96

▶ Queries within Queries

SQL allows queries within queries, or subqueries, which are SELECT statements inside SELECT statements. This might sound a bit odd, but subqueries can actually be very useful. The downside, however, is that they can consume a lot of processing, disk, and memory resources. A subquery's syntax is just the same as a normal SELECT query's syntax. As with a normal SELECT statement, a subquery can contain joins, WHERE clauses, HAVING clauses, and GROUP BY clauses. Specifically, this chapter shows you how to use subqueries with SELECT statements, either returning results inside a column list or helping filter results when used inside a WHERE or HAVING clause; to update and delete data by using subqueries with UPDATE and DELETE statements; and with operators such as EXISTS, ANY, SOME, and ALL, which are introduced later in this chapter.

Subqueries are particularly powerful when coupled with SQL operators such as IN, ANY, SOME, and ALL, which are covered shortly. Before starting, you should note that versions of MySQL prior to version 4.1 do not fully support subqueries and many of the *examples* in this chapter won't work on early versions of MySQL.

A nested query is a query with another query embedded in it.

Embedded query=sub query

A sub query can in turn have sub queries. Nested queries are convenient for computing results that depend on some intermediate results that need to be computed. In SQL a sub query can appear in the FROM or Where or HAVING clause.

Example

Customers

sid	cname	level	type	age
2336	Cho	Beginner	snowboard	18
2334	Luke	Inter	snowboard	25
1887	Ice	Advanced	ski	20
2339	Paul	Beginner	ski	33

Activities

sid	slope-id	day
2336	S3	01/05/03
2336	S1	01/06/03
2336	S1	01/07/03
1887	S2	01/07/03
1887	S1	01/07/03
2334	S2	01/05/03

Slopes

slope-id	name	color
S1	Mountain Run	blue
S2	Olympic Lady	black
S3	Magic Carpet	green
S4	KT-22	black

Find the names of customers who did not go on the slope on 01/07/03

SELECT cname
FROM Customers c
WHERE c.sid NOT IN

```
(SELECT.sid FROM Activities a
WHERE a.day="01/07/03")
```

▶ GROUP BY...

GROUP BY... was added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it was impossible to find the sum for each individual group of column values.

The ***syntax*** for the GROUP BY function is:

```
SELECT column,SUM(column) FROM table GROUP BY column
```

GROUP BY Example

"Sales" Table:

Company	Amount
T System	5500
IBM	4500
T System	7100

SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Returns this result:

Company	SUM(Amount)
T System	17100
IBM	17100
T System	17100

The above code is invalid because the column returned is not part of an aggregate. A GROUP BY clause will solve this problem:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

Returns this result

Company	SUM(Amount)
T System	12600
IBM	4500

▶ HAVING...

HAVING... was added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING... it would be impossible to test for result conditions. The syntax for the HAVING function is:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

This "Sales" Table:

Company	Amount
T System	5500
IBM	4500
T System	7100

This SQL:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

Returns this result

Company	SUM(Amount)
T System	12600

▶ BETWEEN ... AND

The BETWEEN ... AND operator selects a range of data between two values. These values can be numbers, text, or dates.

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

Original Table (used in the *examples)*

LastName	FirstName	Address	City
Chavan	Harish	Hadapsar	Pune
Datt	Nitin	Tathwade	Pune
Sen	Prasad	Anand Nagar	Aurangabad
Singh	Sachin	Katraj	Pune

Example 1

To display the persons alphabetically between (and including) "Harish" and exclusive "Prasad", use the following SQL:

```
SELECT * FROM Persons WHERE LastName
   BETWEEN 'Harish' AND 'Prasad'
```

Result

LastName	FirstName	Address	City
Chavan	Harish	Hadapsar	Pune
Datt	Nitin	Tathwade	Pune

IMPORTANT! The BETWEEN...AND operator is treated differently in different databases. With some databases a person with the LastName of "Harish" or "Prasad" will not be listed (BETWEEN..AND only selects fields that are between and excluding the test values).

With some databases a person with the last name of "Harish " or " Prasad " will be listed (BETWEEN..AND selects fields that are between and including the test values). With other databases a person with the last name of "Harish " will be listed, but " Prasad" will not be listed

(BETWEEN..AND selects fields between the test values, including the first test value and excluding the last test value). Therefore: Check how your database treats the BETWEEN....AND operator!

▶ AND & OR

AND and OR join two or more conditions in a WHERE clause.

The AND operator displays a row if ALL conditions listed are true. The OR operator displays a row if ANY of the conditions listed are true.

Original Table (used in the examples)

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Example

Use AND to display each person with the first name equal to "Tove", and the last name equal to "Svendson":

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

Result

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Example

Use OR to display each person with the first name equal to "Tove", or the last name equal to "Svendson":

```
SELECT * FROM Persons
WHERE firstname='Tove'
OR lastname='Svendson'
```

Result

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Example

You can also combine AND and OR (use parentheses to form complex expressions):

```
SELECT * FROM Persons WHERE
(FirstName='Tove'OR FirstName='Stephen')
        AND LastName='Svendson'
```

Result

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

▶ Pattern Matching

The LIKE predicate allows for a comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters. Two wildcard characters that are available are:

For character data types: The Percent sign (%) matches any string.

The Underscore (_) matches any single character.

Example

1. SELECT * FROM client_master WHERE name LIKE 'Sa%';

   Retrieves all information about client whose names begin with the letters 'Sa' from client_master.

2. SELECT * FROM client_master WHERE name LIKE 'Sa_'

   Retrieves all information about client whose names beginning with letter 'Sa' and having length three characters.

3. IN and NOT IN predicates:

   The arithmetic operator (=) compares a single value to another single value. In case a value needs to be compared to list of values then the IN predicate is used. One can check a single value against multiple values by using the IN predicate.

   Retrieves the name from table client_master where the name is either Vijay, Sachin, Ramesh, Prasad, Kiran.

   SELECT name FROM client_master WHERE name IN ('Vijay','Sachin','Ramesh','Prasad');

   The NOT IN predicate is the opposite of the IN predicate. This will select all the rows where values do not match all of the values in the list.

   SELECT name FROM client_master WHERE name NOT IN ('Vijay','Sachin','Ramesh','Prasad');

3.3 Arithmetic and Aggregate operators

Oct. 2012 – *4M*
Explain SQL aggregate functions with example.

Apr. 2012 – *4M*
List various Aggregate Functions. Explain any one with example.

SQL provides a set of built in functions. The operand of each of these functions is column of an existing relation. NULL values are ignored except (COUNT *). The function is described below.

▶ Aggregation

Aggregation is one way of saying a summary of data. *For example*, aggregating data might involve finding the average age of film club members or counting how many members live in a particular state. What you've learned so far allows you to answer questions pertaining to which film categories each member likes or what Sachin's favorite film category is.

However, by the end of this chapter, using a combination of groups and aggregation will enable you to answer questions such as how many members like thrillers. The difference between grouping and aggregation is that grouping finds out information about a particular record, whereas aggregation summarizes more than one record.

Compute summary results over a table. E.g.,

1. Find the average/min/max score of all students who took CS180.
2. Find the total number of snowboarders.
3. Find total salary of employees in Sales department.

▶ Counting Results

You can use the COUNT () function to count the number of records in the results. It's used in the SELECT statement along with the column list. Inside the brackets, insert the name of the column you want counted. The value returned in the results set is the number of non-NULL values in that column.

Alternatively, you can insert an asterisk (*), in which case all columns for all records in the results set are counted regardless of whether the value is NULL or not. The COUNT () function can also accept expressions, *for example* COUNT(MemberId + CategoryId).

Execute the following SQL:

```
SELECT COUNT(*) FROM MemberDetails;
```

You get the answer 14, which is how many records the SQL returns, which, given the lack of a WHERE clause, represents all the records in the MemberDetails table. However, if you execute the following SQL, you get the answer 13:

```
SELECT COUNT(Street) FROM MemberDetails;
```

Why 13 and not 14? After all, it's the same SQL with no WHERE clause, so surely it has the same number of records regardless of which column is counted.

The difference is that when a column name is specified in the COUNT () function's arguments, only the columns where the value is not NULL are counted.

As you can see from the following table, which is the result of SELECT MemberId, Street, City, State FROM MemberDetails, one of the records in the Street, City, and State columns contains a NULL value:

Member Details

MemberID	Street	City	State
1	Saraf	Pune	Maharashtra
2	Hadapsar	Aurangabad	Maharashtra
3	Pune Nagar	Chiplun	Maharashtra
4	Katraj	Goa	Maharashtra
5	Nagar raod	Beed	Maharashtra
6	Ganesh	Amhemadnagar	Maharashtra
7	Saraf	Sangali	Maharashtra
8	Hadapsar	Kolhapur	Maharashtra
9	Pune Nagar	Pune	Maharashtra
10	Pune Nagar	Wasmat	Maharashtra
11	Katraj	Purandar	Maharashtra
12	Nagar raod	Yewatmal	Maharashtra
13	Ganesh	Chandrapur	Maharashtra
14	Saraf	Amrawati	Maharashtra
15	NULL	NULL	NULL

▶ Adding Results

The SUM() function adds up all the values for the expression passed to it as an argument, either a column name or the result of a calculation. The basic syntax is as follows

```
SUM(expression_to_be_added_together)
```

For example, the following code adds up all the values in the **MemberID** column for all records in the Category table:

```
SELECT SUM(MemberID) FROM MemberDetails;
```

The result of the statement is 120. Given that the **MemberID** column is simply a primary key column, the result is not that meaningful. SUM() can also contain expressions and calculations:

▶ MAX() and MIN() in Results

The MAX() and MIN() aggregate functions return the lowest and highest values found in a results set, respectively.

Basically, it's the same as if the results were ordered and the first and last results were picked from the list. Unlike the SUM() and AVG() functions, you can use MAX() and MIN() with data types other than numerical data types.

For example, you can use MAX() and MIN() to find the earliest or latest date or time in a date or time field. The following SQL finds the youngest and oldest members in the MemberDetails table:

```
SELECT MAX(MemberID, MIN(MemberID) FROM MemberDetails;
```

It provides the following results:

MAX(MemberID) MIN(MemberID)

1 15

In addition to a date or time field, you can also choose a character field. Again, the MIN() and MAX() are the values for the specified column for the first and last records in an ordered results set.

DUAL Table

DUAL is a small Oracle worktable, which consists of only one row and one column, and contains the value x in that column. Besides arithmetic calculations, it also supports date retrieval and it's formatting.

Example

```
SQL> SELECT 2*5 FROM dual;
```

Result

2*5 = 10

Example

```
SQL> SELECT sysdate FROM dual;
```

Result

```
SYSDATE
-----------
04-MAY-2008
```

▶ Numeric Functions

Syntax: **ABS(n)**

Returns the absolute value of n.

Example

```
SQL> SELECT ABS(-15) "Absolute" FROM dual
```

Result

```
Absolute
----------
15
```

Syntax: **POWER(m,n)**

Returns m raised to nth power.

Example

```
SQL>SELECT POWER(3,2) "RAISED" FROM dual
```

Result

```
RAISED
----------
9
```

Syntax: **ROUND(n,m)**

Returns n rounded to m places right of the decimal point.

Example

```
SQL> SELECT ROUND(15.19,1) "ROUND" FROM dual
```

Result

```
ROUND
----------
15.2
```

Syntax: **SQRT(n)**

Returns square root of 'n' if n<0, NULL, SQRT returns a real result.

Example

```
SQL> SELECT SQRT(25)"SQUARE ROOT" FROM dual
```

Result

```
SQUARE ROOT
--------------------
5
```

▶ String Functions

i. **LOWER(char):** Returns char, with all letters in lower case.

Syntax: `LOWER(char)`

Example

```
SQL>SELECT LOWER('RAKESH DESHMUKH') "LOWER" FROM dual
```

Result

```
LOWER
----------
rakesh deshmukh
```

ii. **INITCAP(char):** Returns string with the first letter in upper case.

Syntax: `INITCAP(char)`

Example

```
SQL>SELECT INITCAP('PRAKASH') "TITLE CASE" FROM dual
```

Result

```
TITLE CASE
----------------
Prakash
```

iii. **UPPER(char):** Returns char, with all letters forced to uppercase.

Syntax: `UPPER(char)`

Example

```
SQL> SELECT UPPER('mango') FROM dual
```

Result

```
UPPER ('mango')
----------------------
MANGO
```

iv. **SUBSTR(char, m[,n]):** Returns a portion of char, beginning at character m exceeding upto n characters. If n is omitted, result is returned up to the end char. The first position of char is 1

Syntax: `SUBSTR(char, m[,n])`

Example

```
SQL> SELECT SUBSTR('SELECT',3,4) "SUBSTRING" FROM dual
```

Result

```
SUBSTRING
----------------
 LECT
```

v. **LENGTH(char):** Returns the length of char

Syntax: `LENGTH(char)`

Example

```
SQL> SELECT LENGTH('BEGINNING')"LENGTH" FROM dual
```

Result

```
LENGTH
------------
9
```

vi. **LTRIM (char [,set]):** Removes characters from the left of char with initial characters removed upto the first character no in set.

Syntax: `LTRIM (char [,set])`

Example

```
SQL> SELECT LTRIM('NISHA','N') "LEFT TRIM" FROM dual
```

Result

```
LEFT TRIM
---------------
ISHA
```

vii. **RTRIM (char [,set]):** Removes characters from the right of char with initial characters removed upto the last character no in set.

Syntax: `RTRIM (char [,set])`

Example

```
SQL> SELECT RTRIM ('NISHA','A') "RIGHT TRIM" FROM dual
```

Result

```
RIGHT TRIM
---------------
NISH
```

viii. **LPAD (char1,n,[char2]):** Returns char1, left padded to length n with sequence of character in char2.

Syntax: `LPAD (char1,n,[char2])`

Example

```
SQL> SELECT LPAD ('Hello',10, '*') "LPAD" FROM dual
```

Result

```
LPAD
-------
**** Hello
```

ix. **RPAD(char1,n,[char2]):** Returns char1, right padded to length n with sequence of character in char2.

Syntax: `RPAD(char1,n,[char2])`

Example

```
SQL> SELECT RPAD('Hello ',10,'*') "RPAD" FROM dual
```

Result

```
RPAD
-------
Hello ****
```

▶ Date Functions

i. **TO_DATE (char,[,fmt]):** Converts a character field to a date field.

Syntax: `TO_DATE(char,[,fmt])`

Example

```
SQL> INSERT INTO sales_order(order_no,order_date)
VALUES('087650',TO_DATE('30-SEP-8510:55A.M.', 'DD-MM-YY HH:MM A.M.');
```

ii. **MONTHS_BETWEEN(D1,D2)**

Syntax: `MONTHS_BETWEEN(D1,D2)`

Returns number of months between D_1 and D_2.

Example

```
SQL>SELECT MONTHS_BETWEEN('02-FEB-92','02-JAN-92') "MONTHS" FROM dual
```

Result

```
MONTHS
------------
1
```

▶ Examining Objects Created by a User

Finding out the tables created by user

To determine which table the user has access to the syntax is

```
SELECT * FROM TAB;
```

The objects name and type are displayed. The object types, i.e., the TABTYPE column is the table TAB will be table, since this is the only object created so far.

```
  TNAME   TABTYPE
----------  --------------
  CLIENT TABLE
  STUDENTTABLE
```

Finding out column details of a table created

```
DESCRIBE tablename;
```

This command displays the column names, the data types and the special attribute connected to the table.

Describe Student

NAME	TYPE
NAME	CHAR
ROLLNO	NUMBER

3.4 SQL UPDATE Statement

The UPDATE statement is used to modify the data in a table.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Person

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune
Kale	Anil	Hadapsar	

Update one Column in a Row.

We want to add a first name to the person with a last name of "Kale":

```
UPDATE Person SET FirstName = 'Anil'
WHERE LastName = 'Kale'
```

Result

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune
Kale	Anil	Hadapsar	

Update several columns in a Row.

We want to change the address and add the name of the city:

```
UPDATE Person
SET Address='Swarget', City = 'Pune'
WHERE LastName = 'Kale'
```

Result

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune
Kale	Anil	Swarget	Pune

3.5 SQL DELETE Statement

The DELETE statement is used to delete rows in a table.

Syntax

```
DELETE FROM table_name
    WHERE column_name = some_value
```

Person

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune
Kale	Anil	Swarget	Pune

Delete a Row

"Anil Kale" is going to be deleted:

```
DELETE FROM Person WHERE LastName = 'Kale'
```

Result

LastName	FirstName	Address	City
Joshi	Sachin	Kothrud	Pune

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
 DELETE FROM table_name
          or
DELETE * FROM table_name
```

4. View

In SQL, a VIEW is a virtual table based on the result-set of a SELECT statement. A view contains rows and columns, just like a real table.

The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from a single table.

Note: The database design and structure will NOT be affected by the functions, where, or join statements in a view.

Syntax

```
CREATE VIEW view_name AS
 SELECT column_name(s)
 FROM table_name
 WHERE condition
```

Note: The database does not store the view data. The database engine recreates the data, using the view's SELECT statement, every time a user queries a view.

Example

Consider the client_master table referred above had more additional fields like Address1, Address2, City, PinCode, State, Etc. And we are suppose to create view on the client_master table for the administration department. The view would be created as below.

CREATE VIEW vw_clientadmin AS SELECT name, address1, address2, city, pincode, state FROM client_master. Once a view has been created it can be queried exactly like base table.

Syntax

```
SELECT columnname, columnname from viewname;
```

Example

```
SELECT name, city FROM vw_clientadmin WHERE city IN ('Pune','Aurangabad')
```

PU Questions

[Oct.2012– 4M]	1.	Explain SQL aggregate functions with example.
[Apr.2012– 4M]	2.	List various Aggregate Functions. Explain any one with example.
[Oct.2011– 4M]	3.	List various DML commands. Explain any one with example.
[Apr.2011– 4M]	4.	List various DDL commands. Explain any one with example.
[Oct.2010– 4M]	5.	Explain any two aggregate functions of SQL with suitable example.
[Apr.2010– 4M]	6.	Explain group by clause with example.

Chapter 5

RELATIONAL DATABASE DESIGN

1. Introduction

For database administrator, planning physical structure of database is critical before building the database. Similarly for application developer careful planning and creation of the database objects is crucial before actually starting the development of the application.

Designing and normalizing tables

The theory of normalization is the key element of database design while creating application schema. Normalization is the process of decomposing and arranging the attributes in the schema, which results in a set of tables, with varying structure. Purpose of normalization is to make the tables as simple as possible.

Normalization directly affects the functionality and performance of an application. *for example,* CUSTOMER (ID, NAME, and ADDRESS).

If data is stored in this format it is not possible to sort the data on the last name of the customer. Instead in the above structure if CUSTOMER (ID, NAME, LNAME, ADDRESS) structure is used we can obtain the required results.

Number of questions asked. 4

Oct. 2012 – *4M*
Explain normalization with example.

Apr.12, 11, Oct.11 – *4M*
Write a short note on Normalization.

The main focus in normalization process is on reducing the redundancy of information within the database.

Another focus is on determining the relationships that exist between different attributes in the schema. This identification of relationships allows basis for determining the primary keys and referential integrity rules in a database.

Data integrity: All of the data in the database are consistent, and satisfy all integrity constraints.

Data redundancy: If data in the database can be found in two different locations (direct redundancy) or if data can be calculated from other data items (**indirect** redundancy) then the data is said to contain redundancy.

Data should only be stored once and avoid storing data that can be calculated from other data already held in the database. During the process of normalization redundancy must be removed, but not at the expense of breaking data integrity rules.

If redundancy exists in the database then problems can arise when the database is in normal operation:

- When data is inserted the data must be duplicated correctly in all places where there is redundancy.

  For instance, if two tables exist for in a database, and both tables contain the employee name, then creating a new employee entry requires that both tables be updated with the employee name.

- When data is modified in the database, if the data being changed has redundancy, then all versions of the redundant data must be updated simultaneously.

  So, in the employee example, a change to the employee name must happen in both tables simultaneously.

  The removal of redundancy helps to prevent insertion, deletion, and update errors, since the data is only available in one attribute of one table in the database.

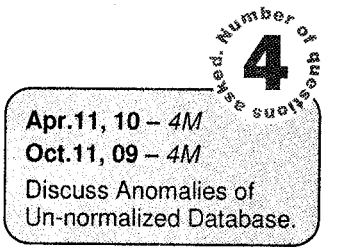
  The data in the database can be considered to be in one of a number of 'normal forms'. Basically the normal form of the data indicates how much redundancy is in that data. The normal forms have a strict ordering:

  1. 1st Normal Form
  2. 2nd Normal Form
  3. 3rd Normal Form
  4. BCNF

There are other normal forms, such as 4th and 5th normal forms. They are rarely utilized in system design and are not considered further here.

To be in a particular form requires that the data meets the criteria to also be in all normal forms before that form.

Thus to be in 2nd normal form the data must meet the criteria for both 2nd normal form and 1st normal form. The higher the form the more redundancy has been eliminated.



▶ Normalization Avoids

- **Duplication of Data:** The same data is listed in multiple lines of the database.
- **Insert Anomaly:** A record about an entity cannot be inserted into the table without first inserting information about another entity .

  For example: If system is having M.C.M 's 5 subject marks and if there is addition of 6th subject then adding valid data to nonexisting subject is Insertion anamoly.
- **Delete Anomaly:** A record cannot be deleted without deleting a record about a related entity.

  For example: If student registered in a given course discontinues the course, the information as to which professor is offering the course will be lost if this is the only relation in the database showing the association between a faculty member and the course he or she teaches.
- **Update Anomaly:** Cannot update information without changing information in many places. To update customer information, it must be updated for each sales order the customer has placed.

Normalization is a three stage process – After the first stage, the data is said to be in first normal form, after the second, it is in second normal form, after the third, it is in third normal form

▶ Before Normalization one should take the following care -

1. Begin with a list of all of the fields that must appear in the database. Think of this as one big table.
2. Do not include computed fields.
3. One place to begin getting this information is from a printed document used by the system.
4. Additional attributes besides those for the entities described on the document can be added to the database.

2. Integrity Constraints

Two types of Integrity Constraints
i. Entity integrity
ii. Referential integrity

An integrity constraint is a rule that restricts the values that may be present in the database. The relational data model includes constraints that are used to verify the validity of the data as well as adding meaningful structure to it:

i. **Entity integrity**: The rows (or tuples) in a relation represent entities, and each one must be uniquely identified.

Hence we have the primary key that must have a unique non-null value for each row.

ii. **Referential integrity:** This constraint involves the foreign keys. Foreign keys tie the relations together, so it is vitally important that the links are correct.

Every foreign key must either be null or its value must be the actual value of a key in another relation.

3. Normal Form

Three types of form
i. 1NF
ii. 2NF
iii. 3NF
iv. BCNF

Normalization theory is built around the concept of normal forms. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints.

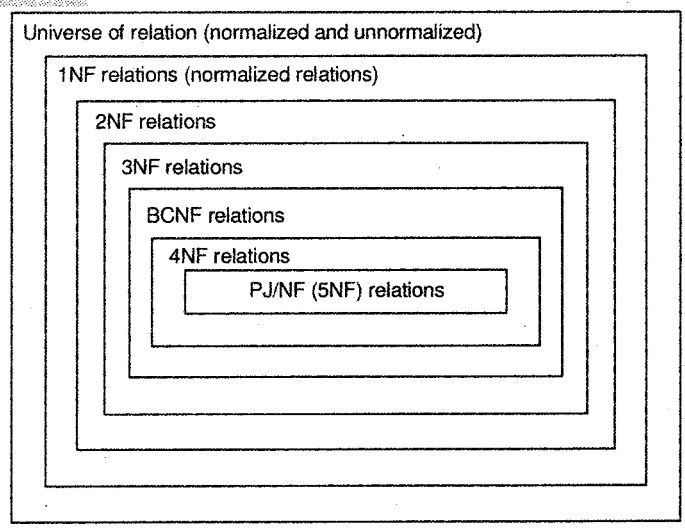


Figure 5.1: Normal forms

Numerous normal forms have been defined as shown in *figure 5.1.* Codd originally defined first, second and third normal form (1NF, 2NF and 3NF).

Figure 5.2 shows Supplier-and-Parts database which will be used to explain the concept of normalization and show how relations are reduced to a particular normal form.

Before we begin with the concept of normal form let us see the concept of functional dependency.

4. Functional Dependency

Sometimes the starting point for understanding data is given in the form of relations and functional dependencies. This would be the case where the starting point in the process was a detailed specification of the problem. We already know what relations are. Functional dependencies are rules stating that given a certain set of attributes (the determinant) determines a second set of attributes.

The definition of a functional dependency looks like A->B. In this case B is a single attribute but it can be as many attributes as required (for instance, X->J, K, L, M).

In the functional dependency, the determinant (the left hand side of the -> sign) can determine the set of attributes on the right hand side of the -> sign. This basically means that A selects a particular value for B, and that A is unique.

In the second *example*, X is unique and selects a particular set of values for J,K,L, and M. It can also be said that B is functionally dependent on A. In addition, a particular value of A always gives you a particular value for B, but not vice-versa.

Consider this example:

```
R(matric_no, firstname, surname, tutor_number, tutor_name)
                        tutor_number -> tutor_name
```

Here there is a relation R, and a functional dependency that indicates that:

- instances of tutor_number are unique in the data
- from the data, given a tutor_number, it is always possible to work out the tutor_name.
- As an *example* tutor number 1 may be "Mr Smith", but tutor number 10 may also be "Mr Smith". Given a tutor number of 1, this is ALWAYS "Mr Smith". However, given the name "Mr Smith" it is not possible to work out if we are talking about tutor 1 or tutor 10.

There is actually a second functional dependency for this relation, which can be worked out from the relation itself. As the relation has a primary key, then given this attribute you can determine all the other attributes in R. This is an implied functional dependency and is not normally listed in the list of functional dependents.

Thus functional dependency FD can be defined as, "Given a relation R, attribute Y of R is functionally dependent on attribute X of R if and only if each X- value in R has associated with it precisely one Y value in R (at any one time)"

In the suppliers and parts database, *for example,* attributes SNAME, STATUS & CITY of relation S are each functionally depend on attribute S#.

In symbol we have,

S.S# → S.NAME

S.S# → S.STATUS

S.S# → S.CITY

Which can be also represented S.S# → S.(SNAME, STATUS, CITY). Here we can agree upon that the combination (SNAME, STATUS, CITY) are the composite attribute of relation S.

A functional dependence is a special form of integrity constraint. When we say, for example, that the relation S satisfy the FD S.S# → S.CITY we mean that every legal extension (tabulation) of that relation satisfies that constraints; in other words, we are saying something about the intension of the relation.

It is always convenient to represent the FD's in a given set of relations by means of a functional dependency diagram *figure 5.3* shows the functional dependencies in relation S,P,SP.

We also introduce the concept of full functional dependence.

"Attribute Y is fully functionally dependent on attribute X if it is functional dependent on X and functionally dependent on any proper subset of X.

For example, in the relation S, the attribute CITY is functionally dependent on the composite attribute (S#, STATUS); however, it is not fully functionally dependent on S# alone.

Now, once we have understood the concept of functional dependency the next start is to understand first, second and third normal forms.

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

P#	PNAME	COLOR	CITY
P1	Nut	Red	London
P2	Bolt	Green	Paris
P3	Screw	Blue	Rome
P4	Screw	Red	London
P5	Cam	Blue	Paris
P6	Cog	Red	London

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Figure 5.2: Supplier- Parts database: Relational view

5. First Normal Form

A relation R is the first normal form (1NF) if and only if all the underlying domains contain atomic values only.

A relation that is in first normal form has a structure that is undesirable for a number of responses. To illustrate the point, let us suppose that information concerning supplier and shipments, rather than being split into two separate relations (S and SP) is lumped together into a single relation FIRST (S#, STATUS, CITY, P#, QTY). *Figure 5.3* is the functional dependency diagram for the relation first tabulated in *figure 5.4*.

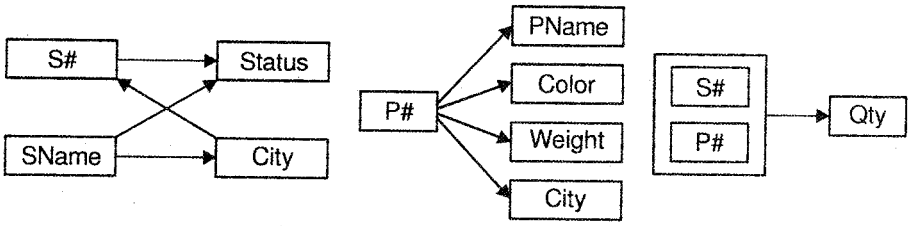


Figure 5.3: Functional dependencies in relation S, P and SP

FIRST

S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

Figure 5.4: Sample tabulation of FIRST

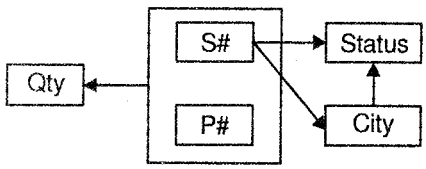


Figure 5.5: Functional dependencies in the relation FIRST

To any database there are three basic operations of insertion, deletion and updation. Concentrating on the association between suppliers and cities that is functional dependency of CITY on S#, problems occur with each of three basic operations.

▶ Inserting

We cannot enter the fact that a particular supplier is located in a particular city until that supplier supplies at least one part. In *figure 5.7* the tabulation doesn't show that supplier S5 is located in Athens. The reason is that, until S5 supplies some part, we have no appropriate primary key value. Here we observe the Integrity rule which states "No component of a primary key value may be null". In relation FIRST, primary key values consist of a supplier number and a part number.

▶ Deleting

If we delete the only first tuple for a particular supplier, we destroy not only the shipment connecting that supplier to some part but also the information that the supplier is located in a particular city. For example, if we delete the FIRST tuple with S# values S3 and P# value P2, we loose the information that S3 is locate in pairs.

▶ Updating

The city value for a given supplier appears in FIRST many times, in general. This redundancy cause's update problem. For example, if supplier S1 moves from London to Amsterdam, we are faced with either the problem of searching the FIRST relation to find every tuple connecting S1 and London or the possibility of productivity an inconsistent result. The solution to the above problem is to replace the relation FIRST by the two relations SECOND (S#, STATUS, CITY) and SP (S#, P#, QTY).

Figure 5.6 shows the functional dependencies in the relations SECOND and SP and *figure 5.7* shows the sample tabulation of SECOND and SP which has over come the above problems.

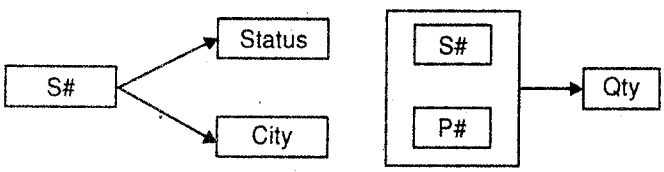


Figure 5.6

S#	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Figure 5.7

6. Second Normal Form

A relation R is in second Normal form (2NF) if and only if it is 1NF and every non key attribute is fully dependent on the primary key. By fully dependency we mean that if any attribute from key attribute is removed, the dependency is not preserved.

Non Key: An attribute is non key if it doesn't participate in the primary key.

Relations SECOND and SP are both 2NF

[The primary keys are S# and the combination CS#, P# respectively]

Relation FIRST is not 2NF. A relation that is in first normal form and not in second can always be reduced to an equivalent collection of 2NF relations.

The reduction of FIRST to the pair (SECOND,SP) is an example of a non loss decomposition since the original relation can always be recovered by taking the natural join of projections (SECOND, SP).

The SECOND/SP structure still causes problems however. Relation SP is satisfactory (SP is in third normal form)

Relation SECOND, on the other hand, still suffers from a lack of mutual independence among the non key attributes.

The dependency diagram for SECOND is still "more complex" than 3NF diagram.

To be specific, the dependency of STATUS on S#, even if it is functional is transitive, i.e., via CITY. Each S# value determines a CITY value, and this in turn determines the STATUS value. This transitivity leads, once again to difficulties over Update operations.

Though in 2NF there still exists some problems with the three basic operation.

▶ Inserting

We cannot enter the fact that a particular city has a particular status value.

For example, we cannot state any supplier in Rome must have a status of 50 until we have some supplier located in that city.

▶ Deleting

If we delete the only SECOND tuple for particular city, we destroy not only the information for the supplier concerned but also the information that city has that particular status value.

For example, if we delete the SECOND tuple from S5, we lose the information that the status for Athens in 30.

▶ Updating

The status value for a given city appears in SECOND manage, in general. Thus if we need to change the status value for London from 20 to 30, we are faced with either the problem of searching the SECOND relation to find every tuple for London or the possibility of producing an inconsistent result.

Again the solution to the problems is to replace the original relation (SECOND) by two projections, in the case (SC, CS#, CITY) and CS (CITY, STATUS).

7. Third Normal Form

Third Normal Form is applied when the relations are in 2NF and there is any non–key attribute that depends transitively on the primary key. The attributes are said to be transitively dependent if PK $\rightarrow$ A and A $\rightarrow$ B. An attribute A is dependent on attribute PK (Primary Key) and B is dependent on A implies that B is transitively dependent on PK PK $\rightarrow$ B.

EmpNo.	EmpName	AreaMarket	Vehicle	Vehicle_Allowance
121	Manoj	Taluka	Sumo	Diesel Bill – 80/-
122	Neelesh	City	Scooter	Petrol Bill – 70/-
123	Vikram	South region	Public	Ticket Amount – 100/-

Here `EmpNo.→ (EmpName, AreaMarket, Vehicle, Vehicle_allowance)`

But, `EmpNo.→ Vehicle And Vehicle → Vehicle_Allowance`

So `EmpNo.→ Vehicle_Allowance`

That means Vehicle-Allowance transitively depends on Emp–No. To transform the relations to 3NF, the transitive dependencies if any are removed by decomposing the relations and referring them by a Primary Key attribute of new relation. The 3NF will be

```
Emp. (Emp-No., Emp-Name, Area-Market, Vehicle-No.)
Vehicle (Vehicle-No., Vehicle-Name, Allowance)
```

8. Boyce-Codd Normal Form (BCNF)

Boyce-Codd Normal Form states mathematically that: A relation R is said to be in BCNF if whenever $X \rightarrow A$ holds in R, and A is not in X, then X is a candidate key for R.BCNF covers very specific situations where 3NF misses inter-dependencies between non-key (but candidate key) attributes.

Typically, any relation that is in 3NF is also in BCNF. However, a 3NF relation won't be in BCNF if (a) there are multiple candidate keys, (b) the keys are composed of multiple attributes, and (c) there are common attributes between the keys.

Example 1: Consider the following relation

Sports Club

Member	Sport	Coach
Anil Rao	Soccer	Kurian
Vivek Das	Soccer	Kurian
Raju Pai	Hockey	Rao
Umesh Vaz	Soccer	Kurian
Rajesh Iyer	Hockey	Rane
⋮	⋮	⋮

The candidates for primary key are:

- Member + Sport
- Member + Coach

because either of these uniquely identifies the row.

The attribute coach is not a candidate for Primary key even though the attribute sport is fully – functionally dependent on it, since a Coach is associated with only one sport (Assumption a Coach is an 'expert' in one sport, though he can coach other sports also).

Since the relation has an attribute (Coach) on which some other attribute is fully-functionally dependent, but is not a candidate for the primary key, the above relation is NOT FULLY NORMALISED. *The above relation will result in the following problems.*

- If Mr. Rajesh Iyer withdraws from the membership, information about coach Rane is also lost.
- If Coach Kurian's name is misspelt and has to be corrected to Kurien, the update needs to be done in all the rows.

A better implementation of the situation can be made through the following relations.

a. MEMBER_SPORT

MEMBER	SPORT
Anil Rao	Soccer
Vivek Das	Soccer
⋮	⋮

b. SPORTS_COACH

SPORT	COACH
Soccer	Kurian
Hockey	Rao
⋮	⋮

By this process we have reduced the 3NF to BCNF by ensuring that they are in 3NF for every Candidate key.

Basically, a humorous way to remember BCNF is that all functional dependencies are: "The key, the whole key, and nothing but the key, so help me Codd."

9. Examples

In this section various examples of normalization has been illustrated that will give better understanding to the concept of normalization.

1. Before Normalization – Example

See Sales Order from below:

Sales Order

Fiction Company
202 N. Main
Mahattan, KS 66502

Customer Number:	1001	Sales Order Number	405
Customer Name:	ABC Company	Sales Order Date	2/1/2000
Customer Address:	100 Points	Clerk Number:	210
	Manhattan, KS 66502	Clerk Name:	Martin Lawrence

Item Ordered	Description	Quantity	Unit Price	Total
800	widgit small	40	60.00	2400.00
801	tingimalligger	20	20.22	400.00
805	thingibob	10	100.00	1000.00
	Order Total			3,800.00

Fields in the original data table will be as follows:

SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName, ItemNo, Description, Qty, UnitPrice

Think of this as the baseline – one large table

Normalization: First Normal Form

- Separate Repeating Groups into New Tables.
- Repeating Groups Fields that may be repeated several times for one document/entity.
- Create a new table containing the repeating data.

The primary key of the new table (repeating group) is always a composite key; usually document number and a field uniquely describing the repeating line, like an item number.

First Normal Form Example

The new table is as follows:

```
SalesOrderNo, ItemNo, Description, Qty, UnitPrice
```

The repeating fields will be removed from the original data table, leaving the following.

SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName

These two tables are a database in first normal form.

What if we did not Normalize the Database to First Normal Form?

Repetition of Data – SO Header data repeated for every line in sales order.

Normalization: Second Normal Form

- Remove Partial Dependencies.
- *Functional Dependency:* The value of one attribute in a table is determined entirely by the value of another.
- *Partial Dependency:* A type of functional dependency where an attribute is functionally dependent on only part of the primary key (primary key must be a composite key).
- Create separate table with the functionally dependent data and the part of the key on which it depends. Tables created at this step will usually contain descriptions of resources.

Second Normal Form Example

The new table will contain the following fields:

```
ItemNo, Description
```

All of these fields except the primary key will be removed from the original table. The primary key will be left in the original table to allow linking of data:

```
SalesOrderNo, ItemNo, Qty, UnitPrice
```

Never treat price as dependent on item. Price may be different for different sales orders (discounts, special customers, etc.)

Along with the unchanged table below, these tables make up a database in second normal form:

SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName

What if we did not Normalize the Database to Second Normal Form?

- Repetition of Data: Description would appear every time we had an order for the item.
- Delete Anomalies: All information about inventory items is stored in the SalesOrderDetail table. Delete a sales order, delete the item.
- Insert Anomalies: To insert an inventory item, must insert sales order.
- Update Anomalies: To change the description, must change it on every SO.

Normalization: Third Normal Form

- Remove transitive dependencies.
- *Transitive Dependency:* A type of functional dependency where an attribute is functionally dependent on an attribute other than the primary key.
  Thus its value is only indirectly determined by the primary key.

- Create a separate table containing the attribute and the fields that are functionally dependent on it. Tables created at this step will usually contain descriptions of either resources or agents. Keep a copy of the key attribute in the original file.

Third Normal Form Example

The new tables would be:

```
CustomerNo, CustomerName, CustomerAdd
ClerkNo, ClerkName
```

All of these fields except the primary key will be removed from the original table. The primary key will be left in the original table to allow linking of data as follows:

```
SalesOrderNo, Date, CustomerNo, ClerkNo
```

Together with the unchanged tables below, these tables make up the database in third normal form.

```
ItemNo, Description
SalesOrderNo, ItemNo, Qty, UnitPrice
```

What if we did not Normalize the Database to Third Normal Form?

- Repetition of Data: Detail for Cust/Clerk would appear on every SO
- Delete Anomalies: Delete a sales order, delete the customer/clerk
- Insert Anomalies: To insert a customer/clerk, must insert sales order.
- Update Anomalies: To change the name/address, etc, must change it on every SO.

Completed Tables in Third Normal Form

```
Customers: CustomerNo, CustomerName, CustomerAdd
Clerks: ClerkNo, ClerkName
Inventory Items:  ItemNo, Description
Sales Orders:  SalesOrderNo, Date, CustomerNo, ClerkNo
SalesOrderDetail:  SalesOrderNo, ItemNo, Qty, UnitPrice
```

---

2. Assumption: A customer can have multiple orders but an order can be for only 1 product. CustName and OrderNo preassigned as keys.

0NF

```
CUSTOMER ORDER(CustName, OrderNo, ProdNo, ProdDesc, Qty, CustAddress,
DateOrdered)
```

1NF - remove multivalued dependencies

```
CUSTOMER (CustName, CustAddress) ORDER (CustName, OrderNo, ProdNo,
ProdDesc, Qty, DateOrdered)
```

2NF - remove partial dependencies

```
CUSTOMER (CustName, CustAddress)
CUSTOMER ORDER (CustName, OrderNo)
ORDER (OrderNo, ProdNo, ProdDesc, Qty, DateOrdered)
```

3NF - remove transitive dependencies

```
CUSTOMER (CustName, CustAddress)
CUSTOMER ORDER (CustName, OrderNo)
ORDER (OrderNo, ProdNo, Qty, DateOrdered)
PRODUCT (ProdNo, ProdDesc)
```

BCNF - resolve intrakey dependencies

```
CUSTOMER(CustName, CustAddress)CUSTOMER ORDER (CustName, OrderNo) -
CustName becomes just a foreign key ORDER
(OrderNo, ProdNo, Qty, DateOrdered)PRODUCT (ProdNo, ProdDesc)
```

---

3. Consider the following unnormalized table

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	201-01	211-02	214-01

First Normal Form: No Repeating Groups

Tables should have only two dimensions. Since one student has several classes, these classes should be listed in a separate table. Fields Class1, Class2, & Class3 in the above record are indications of design trouble.

Spreadsheets often use the third dimension, but tables should not.

Another way to look at this problem: with a one-to-many relationship, do not put the one side and the many side in the same table. Instead, create another table in first normal form by eliminating the repeating group (Class#), as shown below:

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	201-01
4123	Smith	216	211-02
4123	Smith	216	214-01

Second Normal Form: Eliminate Redundant Data

Note the multiple Class# values for each Student# value in the above table. Class# is not functionally dependent on Student# (primary key), so this relationship is not in second normal form.

The following two tables demonstrate second normal form:

Students:

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registration

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01

Third Normal Form: Eliminate Data Not Dependent On Key

In the last example, Adv-Room (the advisor's office number) is functionally dependent on the Advisor attribute.

The solution is to move that attribute from the Students table to the Faculty table, as shown below:

Students:

Student#	Advisor
1022	Jones
4123	Smith

Faculty

Name	Room	Dept
Jones	412	42
Smith	216	42

Solved Examples

1. **Consider the following entities and their relationship**
   Doctor (dno, dname, city)
   Patient (opdno, pat_name, addr, disease)
   The relation between Patient and Doctor is many-to-many.

   Asked. Number of questions 1
   Apr. 2012 – *8M*

   Create a RDB in 3NF and solve any five of the following:

   a. **Insert a row in doctor table.**
   b. **Display names of doctors living in 'Gujarat' city.**
   c. **Add 'discharge_date' column to patient table.**
   d. **Find the names of patients who are treated by 'Dr. Mishra'.**
   e. **Count the number of patients suffering from 'Malaria'.**
   f. **Display total no. of patients treated by each doctor.**

Solution

RDB in 3NF will be as follows:

Doctor (dno, dname, city)

Patient (opdno, pat_name, addr, disease)

Docpat(dno,opdno)

a. Insert a row in doctor table.

```
Insert into doctor values (101,'Mr. Yewole', 'Pune');
```

b. Display names of doctors living in 'Gujarat' city.

```
Select dname
From doctor
Where city = 'Gujarat';
```

c. Add 'discharge_date' column to patient table.

```
Alter table patient add discharge_date date;
```

d. Find the names of patients who are treated by 'Dr. Mishra'.

```
Select pat_name
From doctor,patient,docpat
Where doctor.dno=docpat.dno
and patient.opdno=docpat.opdno
and dname= 'Dr. Mishra';
```

e. Count the number of patients suffering from 'Malaria'.

```
Select count(*) as total
From patient
Where disease='Malaria';
```

f. Display total no. of patients treated by each doctor.

```
Select  dname,count(docpat.opdno)
From  patient,doctor,docpat
Where doctor.dno=docpat.dno and
patient.opdno=docpat.opdno
Group by (dname);
```

Asked. Number of questions 1
Apr. 2011 – *4M*

2. Consider the following entities and their relationship:
Doctor (dno, dname, city)
Patient (opdno, pat_name, addr, disease)
The relation between Patient and Doctor is many_to_many.

Create a RDB in 3NF and solve any five of the following:

a. Insert a row in Doctor Table.

b. Find names of patients who are treated by 'Dr. Deshpande'.

c. Display names of doctors who live in 'Pune' city.

d. Count number of patients suffering from 'Cancer'.

e. Add 'Discharge date' Column to Patient Table.

f. Display total no. of patients treated by each doctor.

Solution

To create RDB in 3 NF third table is created having primary key of doctor ant patient table. So following is the RDB in 3 NF

Doctor (dno, dname, city)

Patient (opdno, pat-name, addr, disease)

Docpat (dno,opdno)

a. Insert a row in Doctor Table.

```
Insert into doctor (dno, dname, city) values (101,'Mr.Kale', 'Pune');
```

b. Find names of patients who are treated by 'Dr. Deshpande'.

```
Select pat_name
From patient,doctor,docpat
Where docpat.opdno=patient.opdno and
      docpat.dno=doctor.dno  and
      dname='Dr. Deshpande';
```

c. Display names of doctors who live in 'Pune' city.

```
Select dname
From doctor
Where city='Pune';
```

d. Count number of patients suffering from 'Cancer'

```
Select count(*) as total_ patients
From patient
Where disease='cancer';
```

e. Add 'Discharge date' Column to Patient Table.

```
Alter table patient
Add (discharge_date date not null);
```

f. Display total no. of patients treated by each doctor.

```
Select dname,count(pno)
From doctor, patient,docpat
Where patient.opdno=docpat.opdno   and doctor.dno=docpat.dno
Group by(dname);
```

Number of questions asked. 1
Oct. 2010 – 8M

3. Consider the following entities and their relationships:
Book(bno,bname,publication,price)
Author(ano,name,addr)
Book and author are related with many to many relationship.
Create RDB in 3NF and solve any five of following:

a. Insert a row in Author table.
b. Count total number of books written by 'Balguruswamy'.
c. Add 'publish_year'(p_year)column to Book table, use alter table command.
d. Update book_name from 'DBMS' to 'RDBMS' published by 'Vision Publication'.
e. Display books published by 'Nirali Publication' and written by 'Mr. Raina'.
f. Display book information whose price is greater than Rs. 300.

Solution

a.
```
Insert into author values(10,'Balguruswamy','California');
```

b.
```
Select count(bookauthor.bno)
From book,author,bookauthor
Where book.bno=bookauthor.bno and author.ano=bookauthor.ano
and aname= 'Balguruswamy';
```

c.
```
Alter table book add p_year(int);
```

d.
```
Update book set bname='RDBMS' where bname='DBMS' and
publication='Vision Publication';
```

e.
```
Select book.bno, bname from book,author,bookauthor where
book.bno=bokauthor.bno and author.ano=bookauthor.ano and
publication='Nirali Publication' and aname='Mr.Raina';
```

f.
```
Select *from book where price>300;
```

Number of questions asked. 1
Apr. 2010 – 8M

4. Consider the following entities and their relationship:
Employee (emp_id, emp_name, desg, salary)
Project (proj_id, proj_name)
Employee and Project are related with many to many relationship with a descriptive attribute 'hrs_worked'.

Create a RDB in 3NF and solve the following queries by using SQL: (Any 5)

a. Insert a row in project table.
b. Display employee details working as 'Project Leader'.
c. Display no. of employees working on 'JAVA' Project.

d. Add 'completion_date' column to project table. Use alter command.
e. Display total number of hours worked by each employee.
f. Display projects on which 'Mr. Amit S.' has worked.

Solution

The relation is many to many with attribute hrs-worked.

∴ Relational table will be:

emp_proj (emp_id, proj_id, worked_hrs)

i.
```
Insert into project values (101, "C++");
```

ii.
```
Select * from employee where employee.desg = 'project leader';
```

iii.
```
Select count (emp_proj.emp_id)
From employee, project, emp_proj Where employee.emp_id =
emp_proj.emp_id and project.proj_id = emp_proj.proj_id  and
project.proj_name = "JAVA";
```

iv.
```
Alter table project add.comp_date date;
```

v.
```
Select sum(emp_proj.hrs_worked), employee.empname From emp_proj,
employee Where emp_proj. emp_id = employee.emp_id
Group by emp_proj.emp_id;
```

vi.
```
Select project.proj_name from employee, project, emp_proj Where
employee.emp_id = emp_proj.emp_id and project.proj_id =
emp_proj.proj_id and employee.empname  = "Mr. Amit S.";
```

---

5. Consider the following entities and their relationship:

Game(gno, gname, no_of_player, coachname)

player(pno, pname)

Game and Player related with many to many relationship.

Asked: Number of questions 1 — Oct. 2009 – *8M*

Create RDB in 3NF and solve any five of the following:

a. Insert a row in game table.
b. List total number of players playing 'Cricket'.
c. Display players having coach as 'Mr. Sharma'.
d. Add 'birthdate' column to player table. Use alter table command.
e. List all games played by 'Rahul'.
f. Count total number of players where coach name is 'Mr. Chappell'.

Solution

RDB 3NF

Game(gno,gname,no_of_player,coachname)

Player(pno,pname)

Game_player(gno,pno)

a. Insert a row in game table.

```
Insert into game values (g1,9"cricket", 11,"Gaikwad");
```

b. List total number of players playing 'Cricket'.

```
Select distinct(count(*)) from player;
```

c. Display players having coach as 'Mr. Sharma'.

```
Select p.pname from player p, game_player gp, game g
Where gp.pno=p.pno and gp.gno=g.gno and g.coachname="Mr. Sharma";
```

d. Add 'birthdate' column to player table. Use alter table command.

```
Alter table player add birthdate date;
```

e. List all games played by 'Rahul'.

```
Select g.gno,g.gname,p.pname from game g, player p, game_player gp
where g.gno=gp.gno and p.pno=gp.pno and p.pname="Rahul";
```

f. Count total number of players where coach name is 'Mr. Chappell'.

```
Select count(no_of_player) from player where coachname="Mr.
Chappell";
```

Asked. Number of questions 1

Apr. 2009 – *8M*

6. 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 by using SQL: (any five)

a. Insert a row in game table.

b. List total number of players playing "Cricket".

c. Add birthdate column to player table. Use alter table command.

d. Count total no. of players whose coach is "Mr. Desouza".

Solution

3NF

game(gno, gname, no_of_player, coachname)

player(pno, pname)

gp(pno,gno)

a. Insert a row in game table.

```
Insert into game (gno, gname, no_of_player, coachname)
values ('g1','cricket', 15, 'Anshuman');
```

b. List total number of players playing "Cricket".

```
Select count(no_of_player) from game;
```

c. Add birthdate column to player table. Use alter table command.

```
Alter player add column bdate date;
```

d. Count total no. of players whose coach name is "Mr. Desouza".

```
Select count(no_of_player) from game where coachname= "Mr. Desouza"
```

Summary

- Normalization is the useful aid in the design process of a database. Normalization theory is built around the concept of normal forms. A relation is said to be in a particular normal form.
- If it satisfies a certain specified set of constraints. The constraints can be used as guide for the reduction process. There are four projection of the original 1NF relation to eliminate every non fall functional dependencies.
- This will produce the collection of 2NF relations. Take three projections of these 2NF relations to eliminate any transitive dependencies. This will produce a collection of 3NF relation.
- The general objective of the reduction process is to reduce redundancy, and hence to avoid certain problems over update operations. Normalization guidelines ever only guidelines; sometimes these are good reasons for not normalizing.
- Also numerous, NF have defined other than 1NF, 2NF & 3NF is BCNF, 4NF, PJ/NF also known as 5NF although there is infinite form to state the number of normal forms, 5NF is actually "final" normal form in a special case, i.e., the ultimate normal form with respect to projection and join.

PU Questions

[Oct.2012– 4M] 1. Explain normalization with example.

[Oct.2012– 8M] 2. Consider the following entities and their relationship:
Student(sno,sname,city,class)
Subject(sub_no, sub_name)
Student and subject are related with many to many relationship with a descriptive attribute 'Marks'.
Create a RDB in 3NF and solve the following queries by using SQL:
i. Insert a row in student table.
ii. Display student no. with highest marks.
iii. Add 'PRN' column to student table,
iv. Display details of student form 'Pune' city having marks >70.
v. Change subject name from 'DBMS' to RDBMS'.
vi. Display the total number of student from 'F.Y.B.C.A.'

[Apr.12,11, Oct. 11– 4M] 3. Write a short note on Normalization.

[Apr.2012– 8M] 4. Consider the following entities and their relationship
Doctor (dno, dname, city)
Patient (opdno, pat_name, addr, disease)
The relation between Patient and Doctor is many-to-many.
Create a RDB in 3NF and solve any five of the following:
a. Insert a row in doctor table.
b. Display names of doctors living in 'Gujarat' city.
c. Add 'discharge_date' column to patient table.
d. Find the names of patients who are treated by 'Dr. Mishra'.
e. Count the number of patients suffering from 'Malaria'.
f. Display total no. of patients treated by each doctor.

[Oct.2011– 8M] 5. Consider the following entities and their relationship
Doctor (dno, dname, city)
Patient (opdno, pat_name, addr, disease)
Doctor and Patient related with many to many relationship.
Create RDB in 3NF and solve any five of the following:
a. Insert a row in patients table.
b. Display names of patients suffering from 'Daibeties' Or 'BP'.

c. Count the no. of patients, treated by Dr. 'Warma'.
d. Add 'addmit _date' column to patients table.
e. Display total number of patient treated by each doctor.
f. Change patient address from 'Pune' to 'Mumbai'.

6. Discuss Anomalies of Un-normalized Database.

7. Consider the following entities and their relationship: [Apr.11,10, Oct. 11,09 – *4M*]
Doctor (dno, dname, city)
Patient (opdno, pat_name, addr, disease) [Apr.2011 – *4M*]
The relation between Patient and Doctor is many_to_many.
Create a RDB in 3NF and solve any five of the following:
a. Insert a row in Doctor Table.
b. Find names of patients who are treated by 'Dr. Deshpande'.
c. Display names of doctors who live in 'Pune' city.
d. Count number of patients suffering from 'Cancer'.
e. Add 'Discharge date' Column to Patient Table.
f. Display total no. of patients treated by each doctor.

8. Define normalization. Explain advantages of normalization with example. [Oct.2010 – *4M*]

9. Consider the following entities and their relationships:
Book(bno,bname,publication,price) [Oct.2010 – *8M*]
Author(ano,name,addr)
Book and author are related with many to many relationship.
Create RDB in 3NF and solve any five of following:
a. Insert a row in Author table.
b. Count total number of books written by 'Balguruswamy'.
c. Add 'publish_year'(p_year)column to Book table, use alter table command.
d. Update book_name from 'DBMS' to 'RDBMS' published by 'Vision Publication'.
e. Display books published by 'Nirali Publication' and written by 'Mr. Raina'.
f. Display book information whose price is greater than Rs. 300.

10. Explain Normalisation with example. [Apr.2010 – *4M*]

11. Consider the following entities and their relationship: [Apr.2010 – *4M*]
Employee (emp_id, emp_name, desg, salary)
Project (proj_id, proj_name)
Employee and Project are related with many to many relationship with a descriptive attribute 'hrs_worked'.

Create a RDB in 3NF and solve the following queries by using SQL: (Any 5)

a. Insert a row in project table.
b. Display employee details working as 'Project Leader'.
c. Display no. of employees working on 'JAVA' Project.
d. Add 'completion_date' column to project table. Use alter command.
e. Display total number of hours worked by each employee.
f. Display projects on which 'Mr. Amit S.' has worked.

[Oct.2009 – 4M]

[Oct.2009 – 8M] 12. Explain Normalization with example.

13. Consider the following entities and their relationship:
Game(gno, gname, no_of_player, coachname)
player(pno, pname)
Game and Player related with many to many relationship.
Create RDB in 3NF and solve any five of the following:

a. Insert a row in game table.
b. List total number of players playing 'Cricket'.
c. Display players having coach as 'Mr. Sharma'.
d. Add 'birthdate' column to player table. Use alter table command.
e. List all games played by 'Rahul'.
f. Count total number of players where coach name is 'Mr. Chappell'.

[Apr.2009 – 4M] 14. Explain Referential Integrity with example.

[Apr.2009 – 4M] 15. Explain Normalization with normal forms.

[Apr.2009 – 4M] 16. Explain Anomalies of Unnormalized Database.

[Apr.2009 – 8M] 17. 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 by using SQL: (any five)

a. Insert a row in game table.
b. List total number of players playing "Cricket".
c. Display all players having coach as "Mr. Sharma".
d. Add birthdate column to player table. Use alter table command.
e. List all games played by Rahul.
f. Count total no. of players whose coach is "Mr. Desouza".

NOTE

Suggestive Readings:

1. Database Management System Tutorial – Tutorials point (2020). Available at: Database Management System Tutorial (Accessed: 26 December 2020).
2. What is DBMS? Application, Types, Example, Advantages (2020). Available at: What is DBMS (Database Management System)? Application, Types & Example (Accessed: 26 December 2020)
3. Database (2013). Available at: Database - Wikipedia (Accessed: 26 December 2020).
4. What is a Database Management System (DBMS)? - Definition from Techopedia (2020).Available at: What is a Database Management System (DBMS)? - Definition from Techopedia (Accessed: 26 December 2020).
5. C. J. Date, An Introduction to Database Systems, Pearson Education, Inc., 8th Edition, 2006.
6. A. Silberschatz, H. F. Korth and S. Sudarshan, Database System Concepts, Tata McGraw-Hill, 6th Edition, 2011.
7. R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, Pearson Education, Inc., 4th Edition, 2004.
8. R. Ramakrishnan and J. Gehrke, Database Management Systems, McGraw-Hil, 3rd Edition, 2007.
9. H. Garcia-Molina, J. D. Ullman and J. Widom, Database Systems: The Complete Book, Pearson Education, Inc., 2nd Edition, 2009.
10. G. Harrison and S. Feuerstein, MySQL stored procedure programming. O'Reilly Media, Inc., 2006.