**Overview of Database Management**

1. What is database?

A database is a logically coherent collection of data with some inherent meaning, and which is designed, built and populated with data for a specific purpose. The key idea behind the database concept is separating the data from the application program (data independence)

2.What is DBMS?

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

3. What is a Database System?

The database and DBMS software together is called as Database system.

4. What was the need for the Database approach?

The database approach was developed and adopted because of the problems associated with data being stored within application programs or within file based systems. The problem with data being stored within application programs was that it was hard to access from other places and there were limits to what could be done with the data

**Database and a File Based System**

A file based system is a collection of application programs that perform services for the users wishing to access information. Each program within a file based system defines and manages its own data in the files. Because of this, **there are limits as to how that data can be used or transported or linked/related.**

**File based systems were developed as better alternatives to paper based filing systems.** By having files stored on computers, the **data could be accessed more efficiently.** It was common practice for larger companies to have **each of its departments looking after its own data.**

The problems that arise with this type of file based system are listed below:

- Data separation and isolation(each file is separate OS object, and hence independent of other files which contain other important data, and hence no concept of relating/linking and/or comparing data in one file with the data in the other file)

- Data dependence(access of data in a particular file is possible only thru those application programs that access that particular file)

- Data duplication(data could be heavily duplicated within the same or across different data files)

- Incompatible data (data in different files could be in different formats)

- Lack of flexibility in organizing/sorting and querying the data(adhoc data analysis and adhoc computations, especially calculations involving data spread in different data files was very difficult)

- Increased number of different application programs(huge number of application programs required to access data from single/multiple data files in different ways)

Some advantages of database systems are outlined below:

- Sharing of the centralized data

- Consistency of data(if data redundancy is avoided)

- Integrity of data(thru constraints)

- Security of data

- Data independence(logical and physical)

- Allows for more analysis of the same amount of data

- Improved data access(since data is centralized) and system performance(several performance enhancement tools/techniques available)

- Increased concurrency

- Improved data backups and recovery

Some potential disadvantages of database systems are the **cost of implementing them, the amount of effort needed to transfer data into the database from a current system, and also the impact on the whole company if the database fails (even if only for a relatively short period).**

**Database Language(SQL) categorization :**

Applications interact with the database thru a special language called SQL(STRUCTURED QUEY LANGUAGE) which includes :

DDL(Create, Alter, Drop)

DML(Insert, Update, Delete)

DQL(Select)

TCL(Commit, Rollback, Savepoint)

DCL(Grant, Revoke)

**INTERFACES**

Following are two important types of Database Interface :

Interactive : SQL commands can be used interactively from a terminal with tools like SQL Plus

Embedded : SQL can be embedded in another host language program(C,C++,Cobol) which interacts with the database.

We also have User-Interface, which are typically Graphical User Interfaces(GUIs) which make interaction with the Database much easier(Example :Toad, PLSQL Developer, etc)

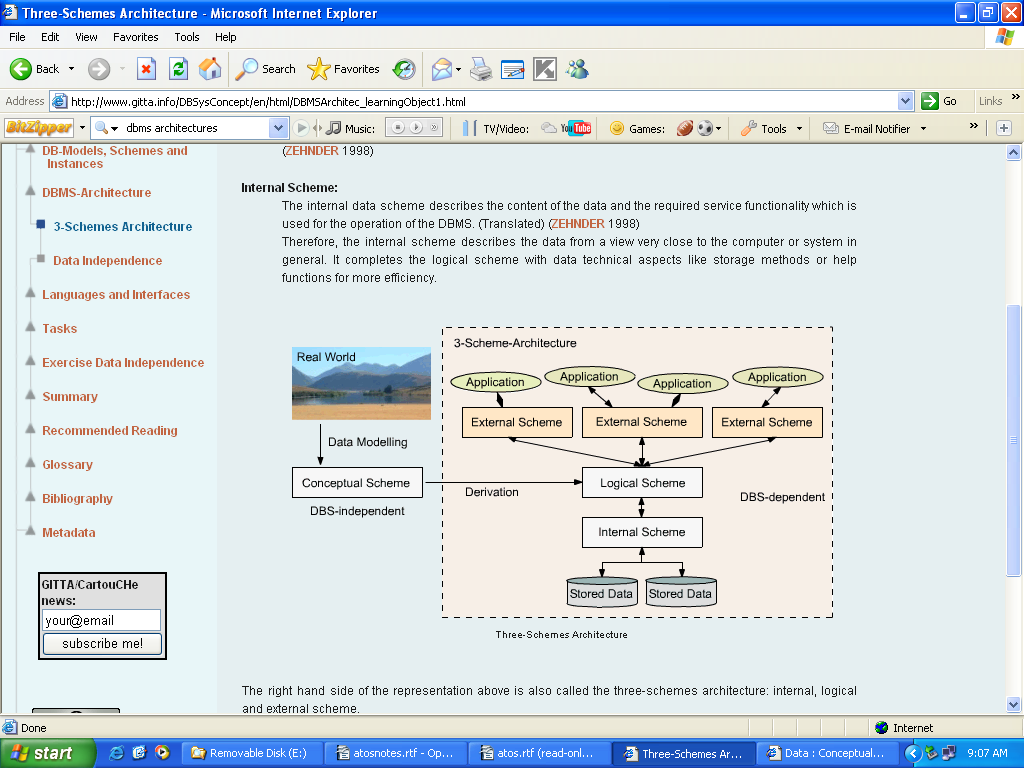
**DBMS architecture**

3-Scheme Architecture:

The **internal scheme** describes the physical grouping of the data and the use of the storage space

The **logical(conceptual) scheme** describes the basic construction of the data structure.

The **external scheme** of a specific application, generally, only highlights that part of the logical scheme which is relevant for its application.



The overall design of the database is called database schema. Schemas are not changed frequently. In general, database systems support one internal schema, one conceptual schema and several external schemas.

**External / View level:** Many users of the database system are not concerned with all the information in the database. Instead, they need to access only a part of the database. The external level simplifies the end user’ s interaction with the system. The system may provide many views for the same database.

**Conceptual / Logical level:** The conceptual level describes what data are stored in the database in terms of objects, and what relationships exist among those objects. This level is used by the *Database Administrators* (DBA) and Database Users, who in turn decide what information must be kept in the database.

**Internal / Physical level:** The internal level describes the data storage and access methods. Database Administrator(s) may be aware of certain details of the physical organization of the data.

**Mapping**

Mapping means that the implementation of the External, Logical(Conceptual) and the Internal(Physical) schemas/schemes should be done in such a way, using the tools/objects of the DBMS, that changes in the Internal/Physical structure should not at all or by very minor efforts affect the External, Logical(Conceptual) layer. Achieving this independence between the Internal/Physical and External and Conceptual) layer is called Data Independence.

**Data independence**

What is Data Independence?

Data independence means that the application is independent of the storage structure and access strategy of data.

Two types of Data Independence:

**Physical Data Independence**: access to the database, via application programs, must remain logically consistent whenever changes to the storage representation or access methods to the data, are changed.

Example, if a user’s data was earlier on a single physical file and now is spread across two or more such physical files, this physical or internal representation should not make any difference to how the user accesses his data via the SQL/PLSQL

Example, if an index is built and destroyed by the DBA on a table, any user should still retrieve the same data from that table, although a bit slowly.

**Logical Data Independence**: Application programs must be independent of changes made to the base tables, with minor changes in the corresponding queries or use of views(makes changes in the query of the view)

TAB1 FRAG1 FRAG2

**A B C D A B A C D**

1 A C E 1 A 1 C E

4 A C F 4 A 4 C F

6 B D G 6 B 6 D G

2 B D H 2 B 2 D H

It should be possible to split a table vertically into more than one fragment as long as such splitting preserves all the original data(is non loss) and maintain the primary key in each and every segment

FRAG1 FRAG2 TAB1

**A B A C D A B C D**

1 A 1 C E 1 A C E

4 A 4 C D 4 A C D

6 B 6 D G 6 B D G

2 B 2 D H 2 B D H

It should be possible to combine base tables into one by way of a non-loss join

**Database Schema** is the overall design/description of the database, and is not frequently subject to changes in a running system.

**Database Instance** is the state of the data in the database at a given moment of time, and is always subject to a lot of changes in an online/running system

**Analytic and Operational Databases**

Analytic databases used for OLAP(On Line Analytical Processing) are primarily static, read-only databases which store archived, historical data used for analysis. For example, a bank might store customer transaction records over the last ten years in an analytic database and use that database to analyze banking strategies or trends to give better service to it’s customers.

Operational databases used for OLTP(On Line Transaction Processing), are used to manage more dynamic bits of data or current or live data. Operational databases allow you to modify data (add, change or delete data). These types of databases are usually used to track real-time information. For example, a Bank might have an operational database used to track day-to-day/daily transaction of their customers, new customer entries, existing customer-detail changes, etc

**Database Security Threats and Countermeasures**

Databases need to have level of security in order to protect the database against both malicious and accidental threats. A threat is any type of situation that will adversely affect the database system. Some factors that drive the need for security are as follows:

- Theft and fraud(Theft of Hardware/Equipment)

- Confidentiality/Privacy(Only Authorised/Authenticated Users access the hardware and/or the software)

- Integrity(Data is always in accordance with Business rules)

- Database availability

Threats to database security can come from many sources. People are a substantial source of database threats. Different types of people can pose different threats. Users can gain unauthorised access through the use of another person's account. Some users may act as hackers and/or create viruses to adversely affect the performance of the system. Programmers can also pose similar threats. The Database Administrator can also cause problems by not imposing an adequate security policy.

Some other threats related to the hardware of the system are as follows:

- Equipment failure

- Deliberate equipment damage (e.g. bombs)

- Accidental / unforeseen equipment damage (e.g. fire, flood)

- Power failure

- Equipment theft

Threats can exist over the communication networks that an organisation uses. Techniques such as wire tapping, cable disruption (cutting / disconnecting), and electronic interference can all be used to disrupt services or reveal private information.

**Countermeasures**  
Some countermeasures that can be employed are outlined below:

- Access Controls (can be Discretionary or Mandatory, possible attempts to get yourself authenticated on the hardware/software when you are not supposed to do so)

- Authentication (determining whether a user is who they claim to be)   
- Authorization (granting legitimate access rights)  
- Backup

- Journaling (maintaining a log file - enables easy recovery of changes)

- Encryption (encoding data using an encryption algorithm)

- RAID (Redundant Array of Independent Disks - protects against data loss due to disk failure)

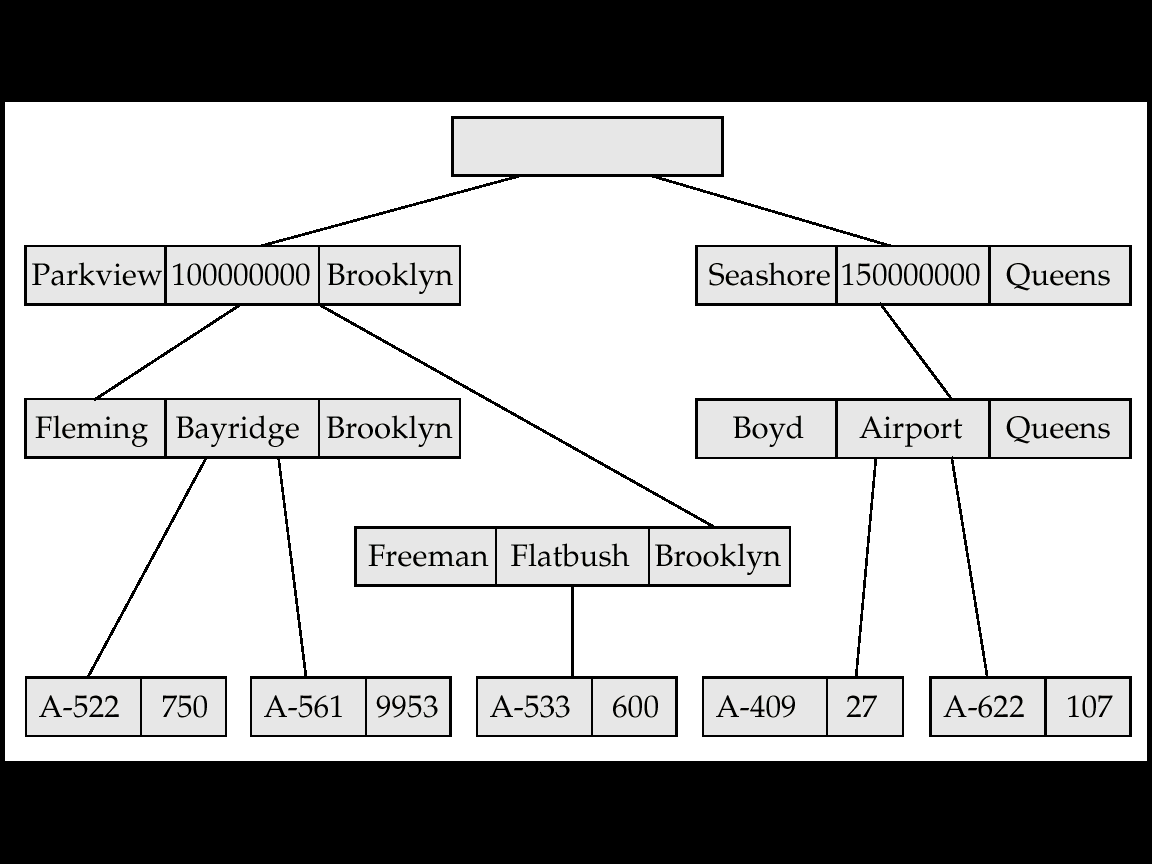
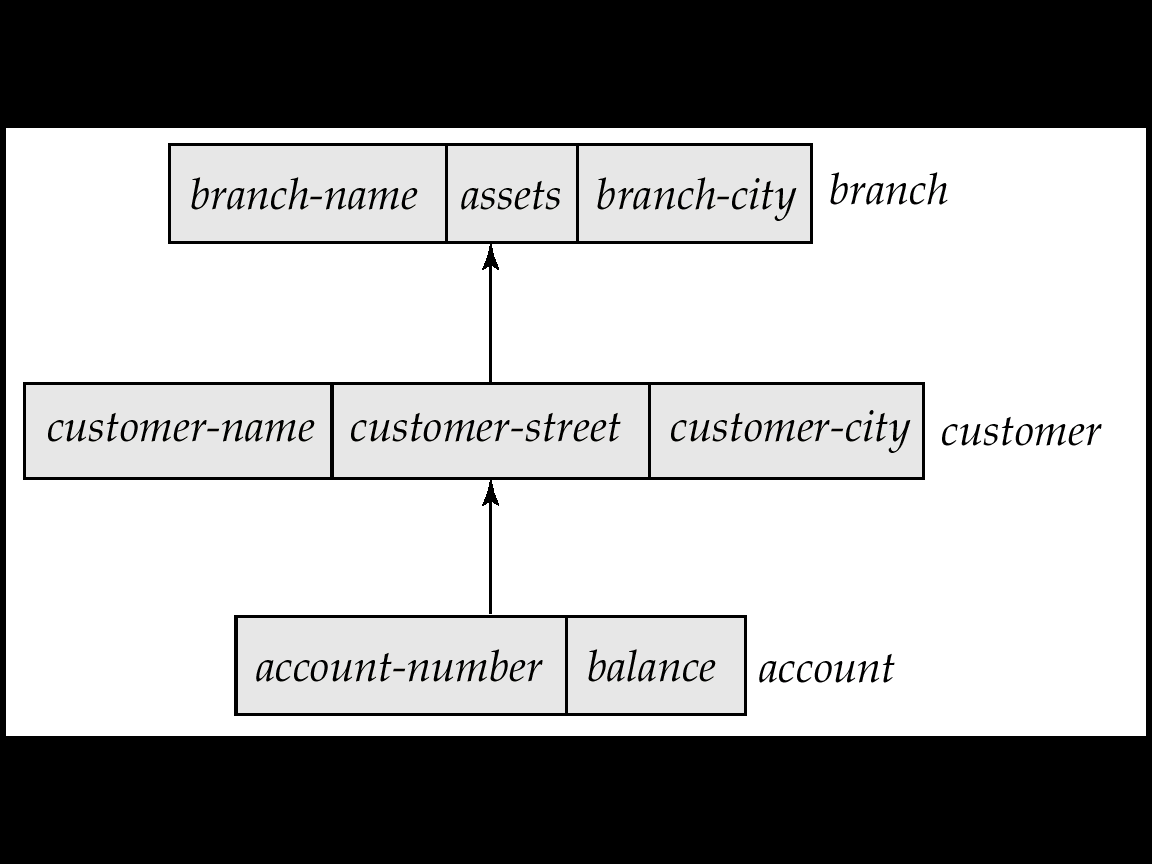
- Polyinstantiation (data objects that appear to have different rights/values to users with different access rights / clearance)

- Views (virtual relations which can limit the data viewable by certain users)

**Record-Based Logical Models**

## Hierarchical Model

* A Hierarchical database consists of a collection of records that are connected to one another through links
* A record is a collection of fields, and each field contains only one data value
* Differs from the Network Model in that the records are organized as collections of trees rather than as arbitrary graphs
* A link is an association between two records



Hierarchical Database Model defines hierarchically-arranged data. You may visualize this type of relationship as an upside down tree of data. In this tree, a single table acts as the "root" of the database from which other tables "branch" out.

Relationships in such a system are thought of in terms of children and parents such that a child may only have one parent but a parent can have multiple children. Parents and children are tied together by links called "pointers" (perhaps physical addresses inside the file system). A parent will have a list of pointers, one for each of it’s children.

This child/parent rule assures that data is systematically accessible. To get to a low-level table, you start at the root and work your way down through the tree until you reach your target. The problem with this system is that the user must know how the tree is structured in order to find the required data.

The hierarchical model however, is much more efficient than the flat-file model as there is not as much need for redundant data. If a change in the data is necessary, the change might only need to be processed once. The flatfile model would store an excessive amount of redundant data. If we implemented this in a hierarchical database model, we would get much less redundant data.

However, the hierarchical database model has some serious problems. You cannot add a record to a child table until it has already been incorporated into the parent table. This might be troublesome if, for example, you wanted to add a customer who had not yet assigned any branch.

The hierarchical database model still creates repetition of data within the database. There could be redundancy because customers having accounts in more than one branch would have redundant information.

Redundancy would occur because hierarchical databases handle one-to-many relationships well but do not handle many-to-many relationships well. This is because a child may only have one parent. However, in many cases you will want to have the child be related to more than one parent.

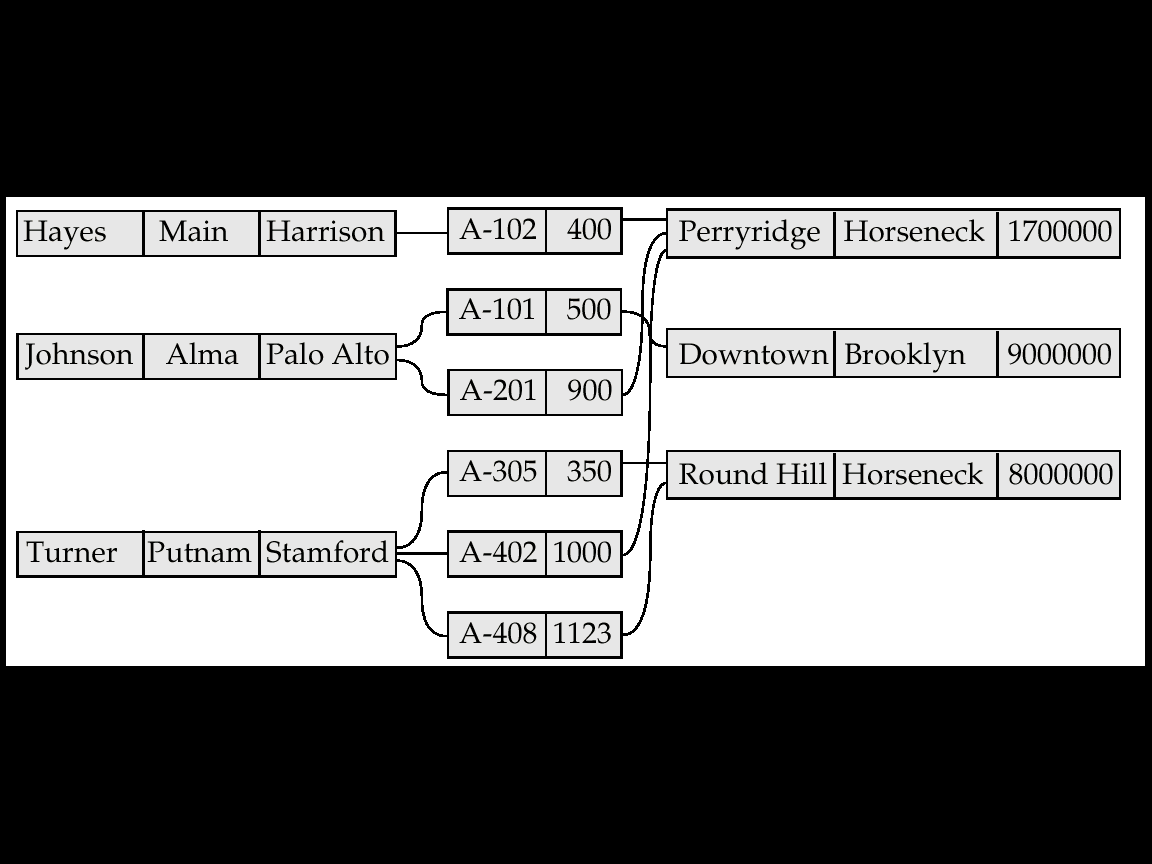
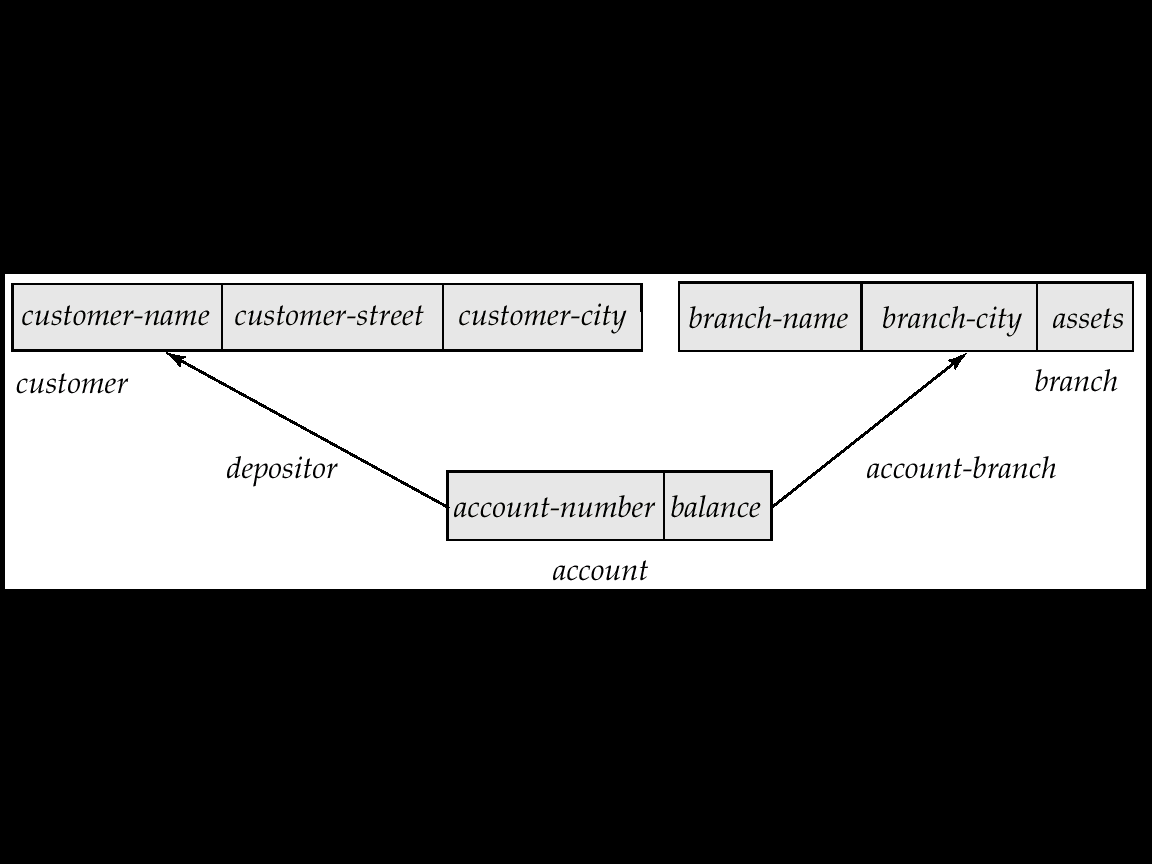
Faced with these serious problems, the network model was evolved.

## Network Model

Network Database model was designed to solve some of the more serious problems with the Hierarchical Database Model. The network model is very similar to the hierarchical model actually. In fact, the hierarchical model is a subset of the network model. However, instead of using a single-parent tree hierarchy, the network model uses set theory to provide a tree-like hierarchy with the exception that child tables were allowed to have more than one parent. This allowed the network model to support many-to-many relationships.

Visually, a Network Database looks like a hierarchical Database in that you can see it as a type of tree. However, in the case of a Network Database, the look is more like several trees which share branches. Thus, children can have multiple parents and parents can have multiple children.

Though it was a dramatic improvement over the hierarchical model, the network model had it’s own problems of implementation. Network model was difficult to implement and maintain. Most implementations of the network model were used by computer programmers rather than real users. What was needed was a simple model which could be used by real end users to solve real problems.



Relational Model

* A Database consists of multiple Relations
* Information about an enterprise is broken up into parts, with each relation storing one part of the information
* Each part is termed as Table and rows of a table are termed as Tuples and columns as Attributes

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

A relational database is a **collection of 2-dimensional tables which consist of rows and columns. Tables are known as "relations", columns are known as "attributes" and rows (or records) are known as "tuples".**

Tables / Relations are a **logical structure therefore they are an abstract concept and they do not represent how the data is stored on the physical computer system. Each column / attribute in a relation represents an attribute of an entity. A single row / tuple contains all the information (in the form of attributes) about a single entity.**

**The "cardinality" of a relation is the number of row / tuples it has. The "degree" of a relation refers to the number of columns / attributes in that relation. The order of records and columns is irrelevant. Relations/Tables and columns should always be uniquely named and therefore uniquely identifiable.** No duplicate rows should occur in a single relation.

**Each cell of a relation contains a single value or element which is atomic. This means that arrays or lists, for example, would not be stored under a single attribute. (Though in OORDBMSs, this is implemented, example, collections in Oracle)**. Multi-valued attributes are possible though but this involves a technique of referring to another relation which holds these multiple values.

**Object-Oriented Relational Model**

* Built on top of relational data models by including object orientation and constructs to deal with added data types
* Allow attributes of tuples to have complex types, including non-atomic values such as nested relations
* Provide a convenient migration path for users of RDBMS who wish to use OO features
* It encapsulates methods with data structures like columns(object columns) and tables(object tables)
* Compatibility with existing relational languages such as SQL

**KEYS concept in RDBMS/Relational Model**

A Key is a single attribute or combination of two or more attributes of an entity set that is used to identify one or more instances of the set.

PRIMARY KEY:-A primary key is a field that uniquely identifies each record in a table. As it uniquely identify each entity, it cannot contain null value and duplicate value.

**Super Key**

Any combination of attributes that can uniquely identify an entity is called a Super Key. The default Super Key is combination of all the columns of a table. A table has many Super Keys

**Candidate Key**

A minimal Super Key is called candidate Key. By minimal we mean if you drop any component from the key then it no more remains unique. A table may have multiple candidate keys

Example 1:

======

Imagine a table with the fields <Name>, <Age>, <SSN> and <Phone Extension>.

This table has many possible superkeys.

Three of these are

<SSN>

<SSN,Phone Extension>

<SSN, Name>.

Of those listed, only <SSN> is a candidate key, as the others contain information not necessary to uniquely identify records.

Example 2:

======

Given table:

EMPLOYEES{employee\_id, firstname, surname, sal}

Possible superkeys are:

{employee\_id}

{employee\_id, firstname}

...

(employee\_id, firstname, surname, sal}

Only the the minimal superkey - {employee\_id} - will be considered as a candidate key.

ALTERNATE KEY:-A candidate key that is not the primary key is called an Alternate key.

COMPOSITE KEY:- Creating a primary key are jointly from more than one attribute is known as composite key.

FOREIGN KEY:- Foreign key is a primary key of master table, which is reference in the current table, so it is known as foreign key in the current table. A foreign key is one or more columns whose value must exist in the primary key of another table or same table

**Domain/Data Constraints**

A domain defines the possible values of an attribute. Domain/Value **Constraints**(Check, Unique, Not Null) govern these values. In a database system, the domain integrity is defined by:

The datatype and the length

The NULL value acceptance

The allowable values, through techniques like check/unique/not null constraints

The default value

For example, if you define the attribute of Age, of an Employee entity, is an integer, the value of every instance of that attribute must always be numeric and an integer. If you also define that this attribute must always be positive, then a negative value is forbidden. The value of this attribute being compulsory indicates that the attribute cannot be NULL. All of these characteristics form the domain/Value constraints of this attribute.

**Integrity Constraints**

Entity Integrity Constraint: States that “Primary key cannot have NULL value”

Referential Integrity Constraint: States that “Foreign Key can be either a NULL value or should be a value from the Primary Key to which it points.

**Relational Algebra**

Relational Algebra is a procedural language which specifies how to get the required information and how to build a relation(set) from one or more other relations(sets). It specifies operations to be performed on existing relations(sets). The basic operations are the traditional set operations : UNION, DIFFERENCE, INTERSECTION and CARTESIAN PRODUCT. The first three operations, except Cartesian product, require that the operands(sets or relations or tables) be compatible(structurally same number, order and data-type of attributes) for the operation.

Consider the following two Operands(sets or relations or tables) which are Union compatible.

P Q

ID NAME ID NAME

101 Jones 103 Smith

103 Smith 104 Loly

104 Loly 106 Byron

107 Even 110 Drew

110 Drew

112 Smith

UNION operation

Hence **R=P U(Union) Q**

ID Name

101 Jones

103 Smith

104 Loly

107 Even

110 Drew

112 Smith

106 Byron

DIFFERENCE operation

Hence  **R=P-Q**

ID Name

101 Jones

107 Even

112 Smith

**R=Q-P**

ID Name

106 Byron

INTERSECTION operation

**R=P Q**

ID Name

103 Smith

104 Loly

110 Drew

CARTESIAN PRODUCT

P Q

ID NAME DEPTNAME

101 Jones Accounts

103 Smith Medical

104 Loly IT

107 Even

**R=P X Q**

ID NAME DEPTNAME

101 Jones Accounts

101 Jones Medical

101 Jones IT

103 Smith Accounts

103 Smith Medical

103 Smith IT

104 Loly Accounts

104 Loly Medical

104 Loly IT

107 Even Accounts

107 Even Medical

107 Even IT

**Additional Relational Algebraic Operations**

Besides the above 4 basic set operations, PROJECTION, SELECTION, JOIN and DIVISION.

Projection(Л)

The projection of a relation(table or set) is defined as a projection of all its tuples(rows) over some set of attributes, i.e., yields a vertical subset of the relation. The projection operation is used to either reduce the number of attributes in the relation under consideration or to reorder its attributes

PERSONAL Projection

ID NAME AGE ID AGE

1 A 23 1 23

2 B 56 2 56

3 C 34 3 34

4 D 87 4 87

5 E 25 5 27

The projection operation can also be used to reduce the cardinality of the resultant relation, that is, due to deletion of duplicate rows, if any.

Selection(σ)

The selection operation yields a horizontal subset of a given relation, that is, it selects only some tuples(rows) of a given relation under consideration. To have a tuple(row) included in the resultant relation, the specified selection conditions or predicates must be satisfied.

PERSONAL Selection(Age<=50)

ID NAME AGE ID NAME AGE

1 A 23 1 A 23

2 B 56 3 C 34

3 C 34 5 E 25

4 D 87

5 E 25

Join(∞)

It allows combining of two or more relations to form a single new relation. The tuples(rows) from the operand relations that participate in the join operation and contribute to the resultant relation are related. Thus, the join operation allows the processing of relationships existing between the operand relations.

DEPT EMP

Deptno Dname Empno Ename Age Deptno

10 Accounts 1 A 23 20

20 Medical 2 B 43 20

30 IT 3 C 65 30

4 D 42 10

5 E 45 30

6 F 62 10

Considering the above two operand relations(tables), suppose we want to respond to the query “Get the Empno of all employees whose department name is Accounts”.

This requires first computing the Cartesian product of the DEPT and EMP relations. Let us name this Cartesian product as TEMP. This is followed by selecting those tuples(rows) of TEMP where the attribute Dname has the value “Accounts” and the value of the attribute Deptno of DEPT relation is equal to the value of the attribute Deptno of EMP relation. The required result is obtained by projecting these tuples on the attribute Empno. The operations are shown below :

Cartesian Product : TEMP=(DEPT X EMP)

Projection :ЛEmpno(σ Dname=’Accounts’ ^ DEPT.deptno=EMP.deptno(TEMP))

Division(+)

Consider the following relation(table) P:

P

A B

a1 b1

a1 b2

a2 b1

a3 b1

a4 b2

a5 b1

a5 b2

Scenario 1

Now, if the relation Q is

Q

B

b1

b2

Hence, R= P ÷ Q

A

a1

a5

Scenario 2

Now, if the relation Q is

Q

B

b1

Hence, R= P ÷ Q

A

a1

a2

a3

a5

Scenario 3

Now, if the relation Q is

Q

B

b1

b2

b3

Hence, R= P ÷ Q

A

Empty

Scenario 4

Now, if the relation Q is

Q

B

Empty

Hence, R= P ÷ Q

A

a1

a2

a3

a4

a5

In scenario 1, the result of dividing P by Q is the relation R and it has two tuples. For each tuple in R, its product with the tuples of Q must be in P. In our scenario, (a1,b1), (a1,b2),(a5,b1) and (a5,b2) must all be tuples in P.

In scenario 2, the Cartesian product of Q and R gives a resulting relation which is again a subset of P.

In scenario 3, since there are no tuples in P with a value b3 for the attribute B(that is selectionB=b3(P)=0), we have an empty relation R, which has a cardinality of zero

In scenario 4, the relation Q is empty. It is unusual to allow division by an empty relation. The resultant relation is projection of P on the attributes in P-Q.

Consider Q as set of properties. Each tuple in P represents an object with some given property. The resultant relation R is a set of entities/tuples that posses all of the properties specified in Q. The two entities a1 and a5 posses all the properties, b1 and b2. The other entities/tuples in P, a2, a3 and a4 only posses one, not both, of the properties.

**The Division operation is useful when a query involves the phrase “for all the objects/tuples having all the specified properties”**

**Relational calculus** simply provides a definition of a relation in terms of one or more other relations. In other words, it says what data is required, but not how it is actually retrieved. Hence its is non-procedural, as it is a query system wherein queries are expressed as variables and formulas on these variables.

**Relation Operators**

In a **relational operator** is a construct that tests some kind of relation between two entities . These include numerical equality (e.g., 5=5) and inequality (e.g., 4≥3). In programming languages that include a distinct boolean type in their type system, like java, these operators return true or false, depending on whether the conditional relationship between the two entities holds or not. In other languages such as C, relational operators return the integers 0 or 1.

Relational Operator Symbol

EQ Equal to =

NE Not equal to <> or # or !=

GT Greater than >

GE Greater than or equal to >=

LT Less than <

LE Less than or equal to <

An expression created using a relational operator forms what is known as a *relational expression* or a *condition*.

**DATABASE STORAGE HIERARCHY**

**1) DATABASE**

A collection of data files integrated and organised into a simple comprehensive file system, which is arranged to minimize duplication of data and to provide convenient access to information within that system to satisfy a wide variety of user needs.

**2) DATA FILES**

Datafile is the file which actually physically keeps the data in secondary storage device. It represents the physical representation of the database.

**3) DATA OBJECTS(Tables)**

Objects such as tables, procedures, functions, etc, especially TABLES logically keep the data stored together. These objects may me spread across one or more physical data files. Tables or Relations are Entity Sets.

**4) RECORDS**

A collection of related items of data that are treated as a unit. It is the information about an entity.

eg:- An employee record would be collection of all fields of one employee.  
Record is sometimes referred as tuple.

**5) FIELD**

Individual element of data is called Field.  
eg:- Bank cheque consist of following field cheque no, date, payee, numeric amt, signature, bank,etc.

Field is sometimes referred as Data item Or column or attribute.

**Introducing Relational Databases**

**Relational Model Paradigm**

1) In a DBMS, relation between two files is created and maintained programmatically. Hence the relation is temporary.

In a RDBMS, relation between the two tables can be created at the time of table creation or later on, and hence it is permanent

2) In a DBMS, we have the concept of Fields, Records and Database Files. In a RDBMS, we have the concept of Column(attribute), Row(Tuple or Entity) and Table(Relation or Entity Class/Set or Applet)

3) IN a DBMS, Network traffic is high, since most of the processing is done at the client or requestors side.

In a RDBMS, network traffic is low, since most of the processing is done at the Server side

4) DBMS does not support the Client-Server architecture. It is a point-to-point communication between two machines

Most of the RDBMSs support the Client-Server Architecture

Example MS Access is a RDBMS, which does not support the Client-Server Architecture.

5) In a DBMS, we have file –level locking.

In a RDBMS, we have table, as well as Row-level locking

6) In a DBMS, there is no support for Distributed-Database Concept

In a RDBMS, there is support for a Distributed-Database Concept

7) In a DBMS. There is no Security of Data.

In a RDBMS, we have multiple levels of Data-Security like i)Logging in Security ii) Command Level Security iii)Object Level Security

8) In a RDBMS, we have the concept of Data-Dictionary(or Meta-Data or Data about Data) implemented in the form of System Tables and Views.

In a DBMS, there is no Data-Dictionary concept.

**Database Design / Application Lifecycle**

Before a database design is set into motion, planning is an essential stage. This is typically the responsibility of management within the organisation. Good planning will enable the design and implementation to be successfully completed with the desired results. A general definition of the desired system should also be drawn up at this stage. This would include information such as what the database application should be able to do, to what areas it will be applied, and who will be using it.

Three main factors that should be analysed in the planning are  
- What work will need to be done  
- The resources needed to complete the work  
- How much it will all cost

Requirements Analysis and Collection  
Information can be gathered using various techniques. Some of these are listed below:

- Interviewing individuals

- Observation

- Examining documents

- Using questionnaires

- Using expert knowledge and experience from other design work

**Interviewing many individuals can be time consuming but can result in quality feedback which can be used in the system design. Using questionnaires is a quicker and easier way to gain feedback from relevant people. It is hard to ensure that the answers people give are accurate though. Looking at the current documents and paperwork in circulation can be useful in determining Database Design**

The database design stage will result in a database model that will hopefully support the organization’s goals. The two design approaches that can be used are Bottom Up or Top Down. **The Top Down approach starts with a very general overview of the system and details are added in an iterative manner. Bottom Up design involves getting all the details first then constructing the overall design from the smaller more detailed parts.**

**Basic RDBMS TERMS**

**1) RELATION**

A relationship is an association among several entities  
eg:- A cusst\_Acct relationship associates a customer with each account that she or he has.

Also, a table or an entity set is sometimes called a relation.

In both cases, a relation is a rectangular output consisting of row(tuples or records) and columns(attributes of fields)

**2) REDUNDANCY**

If same piece of information is stored in database for number of times the database is said to be redundant. We should check our database should not be redundant as it wastes disk space, reduced efficiency of database, require more processing time, and their are chances of inconsistency due to it in our database.  
eg:-If we have to tables emp\_details (contains details of employee) and Payroll(contains Payment details to employee), than if we include details of employee in payroll table, than it is said to be redundancy as same piece of information is repeated.

**3) INCONSISTENCY**

Inconsistency is various copies of the same data which is not consistent. Inconsistency occurs due to redundancy, so redundancy should be reduced.

eg:- If we have details of employee stored in emp\_details and payroll table than while updating information we should check that both tables are updated or not, if we update the address of one employee in emp\_details and same details is not updated in payroll table, than database is said to be in inconsistent state.

**4) PROPAGATING UPDATES**

Propagating updates ensures that changes made to records/data of one relation or tables, are automatically made to other tables or relations. This process is known as Propagating updates. Where the term "Updates" is used to cover all the operations of insertion, deletion and modification.   
We can avoid inconsistency by using propagating update technique.

**6)USERS**

There are broadly three different types of database system users.

**End Users**

They are users who interact with the system by invoking one of the application programs. Thus, they are persons who use the information generated by a computer based system. Retrieval is the most common function for this class of user. Generally application user, data entry operators and report generators are come under this category.

**Application Programmers**

They are who prepare or code the application. Application programs operates on the data in all the usual ways: retrieving information, inserting new information, deleting or changing existing information.

**Database Administrator**

These are the highly skilled users responsible for the overall performance and maintenance of the database. They design the database schema, create the database, create users, grant and revoke rights from users and manage the database. Their responsibility includes recovery of database in case of failure and tuning the database for optimum efficiency.

**Roles**

A Role is a collection of rights which can be allocated to one or more users of the Database. It helps in minimizing the administrative tasks/maintenance of the DBAs.

# Entity Relationship Model and Diagram

# ER model(Chen and Crow’s Foot models) forms the basis of an ER diagram

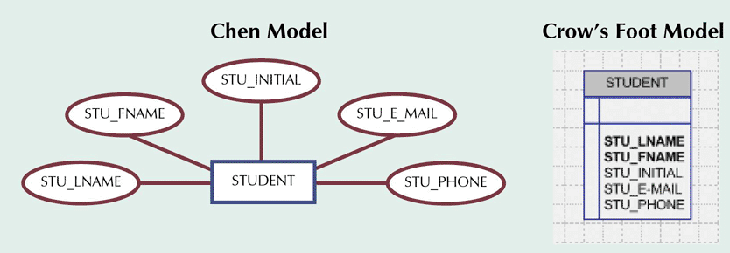
# ERD represents the conceptual database as viewed by end user

ERDs depict the ER model’s three main components:

Entities

Attributes

Relationships



Entities and the Relationships between them are shown using following conventions(Chen Model) :

-An **entity** is shown as a **rectangle**.

-A **diamond** represents the **relationships** among a number of entities, which are connected to the diamond by **lines**.

-The **attributes**, shown as **ovals**, are connected to the entities or relationships by lines.

-Diamonds, rectangles and ovals are **labeled**. The type of relationship existing between the entities is represented by giving the **cardinality** of the relationship on the line joining the relationship to the entity.

Attr1

N

M

Attr1

Attr1

RELATIONSHIP

ENTITY2

ENTITY1

Attr2

Attr2

Attr2

**Entity**

An entity is an object that is of interest to an organization. Entities or Objects of similar types are characterised by some **common set of properties or attributes**. Such similar entities or objects form an **entity set or entity type**. Two objects or entities are distinguishable and this fact is represented in the entity set by giving them a **Unique identifier(s)**.

# Attributes

# They are characteristics of entities. *Domain* is a set of possible values for a particular attribute. Key or Unique Identifier or Primary key is underlined or boldfaced in the ER diagram. Attributes can be of following types :

### Simple

# Cannot be subdivided such as Age, sex, etc

# Composite

# Can be subdivided such as Address consists of street, city, state, zip

# Single-valued

# Has only a single value such as Social security number

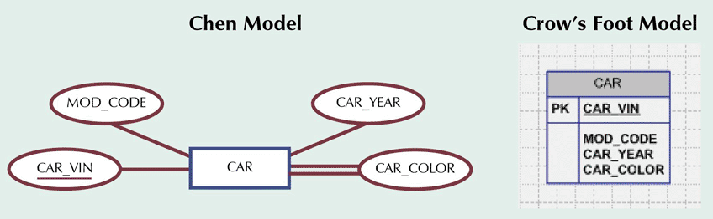
# Multi-valued

# Can have many values. For example a person may have several college degrees

# Derived

# Can be calculated from other information like age can be derived from D.O.B., total salary can be derived from Basic, DA, TA, etc

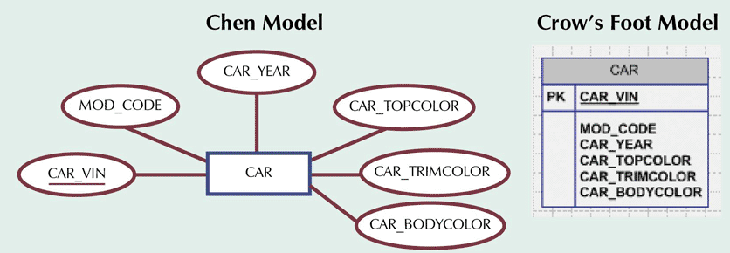
**Multivalued Attributes**

****

# Resolving Multivalued Attribute Problems

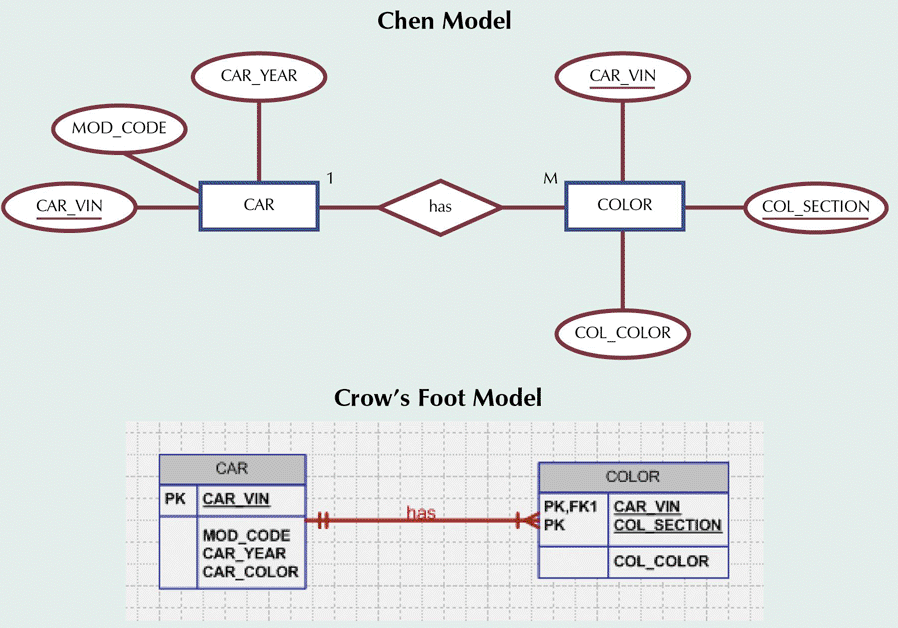
Creating new attributes

# Within original entity, create several new attributes, one for each of the original multivalued attribute’s components. But this can lead to major structural problems in the table



# Creating a new Entity Set

# Create a new entity composed of original multivalued attribute’s components



# Relationships

# They are associations between entities which are established by Business Rules. The Connected entities are termed as participants. The Connectivity describes relationship classification(1:1, 1:M, M:N). Cardinality is the number of entity occurrences associated with one occurrence of related entity.

# Relationship Strength

# Existence Dependent

# Entity's existence depends on existence of another related entitity.

# Child is dependent on employee. A child cannot exist without a corresponding parent in the employee entity set.

# Existence-independent

# Entities can exist apart from related entities

# Employee has Children. Existence of employee does not depend on existence of children as an employee may or may not have children

Thus, the entity Employee is Child-independent, but the entity child is Employee-dependent

Existence Dependency/Existence Independency has got nothing to do with the Relationship type(one-to-one, one-to-many, or many-to-many). Example, a table contains multiple columns, The relationship is one-to-many. But existence of a table requires at least one column, similarly, a column cannot exist without a table. Hence both, table is existence dependent on column, and column is also existence dependent on table. Similarly, personal and salary details in two different tables, relationship is one-to-one, but both are existence dependent on the other.

**Strong and Weak Entities** :

Objects are represented by their **attributes**. As objects are **inter-distinguishable**, a subset of these attributes forms a **Primary Key** or a **key that Uniquely identifies that object** also known as **an instance of an Entity Set**. **An entity which has its own unique identifier, and is not built from another entities’ identifier is called STRONG/DETERMINANT entity.**

For example in any organization, the Entity set **EMPLOYEE will be a Strong Entity**, since each employee has a **unique Employee number**. No two instances of the Entity Employee can have the same value for the attribute Employee number.

A weak entity is called so since t**he Primary Key or the unique identifier of the such an entity includes the Primary key of the Strong entity to which such a weak entity is related.** Thus, the weak entity, **has primary key that is partially or totally derived from parent entity.**

Consider the following example :

An organization maintains it’s Employee details and also the details of family members(or dependents) of each employees.

Employee\_id

Dependent\_name

Relation\_to\_employees

1

N

EMPLOYEE

DEPENDENTS

Here, **Dependents is a Weak Entity Set, since currently it does not have any attribute which uniquely identifies each instance of it**. Instances of a Weak Entity Set associated with the same instance of the Strong Entity Set must be distinguishable from each other by a subset of attributes of the weak entity set(the subset may be the entire weak entity, that is combination of Dependent\_name and Relation\_to\_employee). **Such a subset of attributes of the weak entity set is called Discriminator of the weak entity set**.

But to identify the relationship between the employees and his dependents we must add the employee\_id attribute into the Depandents table, and then make the combination of Employee\_id, Dependente\_name, Relation\_to\_employee as the composite Unique identifier. Thus the entity Dependents would be then a Weak entity as its’ unique identifier **is partially or totally derived from parent entity.**

**Identifying Relationship**

The Primary Key of the weak entity set is thus formed by using the Primary Key of the strong entity set to which it is related, alongwith the discriminator of the weak entity set. Thus, the attribute employee\_id has to be added to the weak entity Dependents. This special type of relationship that applies when we have weak entity is called **“Identifying Relationship”, and is indicated like a diamond within a diamond**

Here by adding some attribute like PAN number, Social Security number or such other attribute, the weak entity may be converted to a Strong Entity set.

Assume, there are two employees : Ram Sharma(employee\_id=10) and Gopal Sharma(employee\_id=50). Ram Sharma has two dependents : Sita Sharma(wife), Rahul Sharma(son). Gopal Sharma has two dependents : Madhurre Sharma(wife), Rahul Sharma(son). Here, the two instances of the weak entity(Rahul Sharma, son) associated with two different instances of the strong Entity Set Employee are **non-distinguishable** from each other. They are nonetheless distinct because they are associated with different instances of the strong entity set employee. **The Primary Key of the weak entity set is thus formed by using the Primary Key of the strong entity set to which it is related, alongwith the discriminator of the weak entity set. Thus, the attribute employee\_id has to be added to the weak entity Dependents.**

RELATION

N

1

Dependent\_id

Dependent\_id

Relation\_to\_employee

Dependent\_name

Employee\_id

Employee\_id

DEPENDENTS

EMPLOYEE

The relationship ”RELATION” is an example of “**IDENTIFYING RELATIONSHIP**” and the entity set “DEPENDENTS” is an example of **Weak Entity**.

**Degree of Relationship**

A relationships degree indicates the number of associated entities.

A relationship is called a Binary Relationship, if the number of entity sets involved is two.

M

N

STUDENT

COURSE

ENROLLMENT

However, the entity sets involved in a relationship may not always be distinct entity sets. You may call them Unary Relationship.

EMPLOYEE

Manager Subordinates

Manages

1

N

PERSON

1

1

SPOUSE

SPOUSE

Marries

PERSON

PARENTS

CHILDREN

M

N

OFFSPRINGS

A relationship that involves N entities is called N-ary Relationship.

Following is an example of Ternary Relationship.

STUDYING represents the relationship between a STUDENT studying a COURSE through a particular STUDYCENTRE.

STUDYCENTRE

STUDYING

COURSE

STUDENT

n



Identification of a relationship is done by using the Primary Keys of the entities involved in it. Thus, if Relationship R involves Entity Sets E1, E2, E3,….Ek having Primary Keys p1, p2, p3,… pk, the unique identifier of the relationship R is the composite attribute(p1, p2, p3,…pk). An instance of the relationship R is represented by concatenating it’s attribute(r1, r2, …. rm) with the primary keys of the instances of the entities involved in the relationship.

**ENHANCED E-R MODEL**

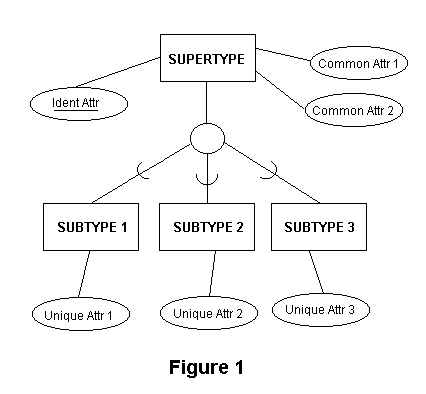
The Enhanced E-R Model incorporates extensions to Chen's original E-R model.  There have been many proposed enhancements.

**GENERALISATION** is an Abstracting process of viewing sets of objects as a single general class by concentrating on the general characteristics of the constituent sets while suppressing or ignoring their differences. It is the union of a number of low-level entity types for the purpose of producing a higher-level entity type. Example : STUDENT is a Generalisation of Graduate or Post-Graduate, Full-Time or Part-Time students. Also EMPLOYEE is a Generalisation of the Classes of Objects like Cook, Waiter, Cashier, etc. Generalisation is an IS\_A relationship. Thus, a cook IS\_A employee, manager IS\_A employee, full time student IS\_A student, graduate IS\_A student.

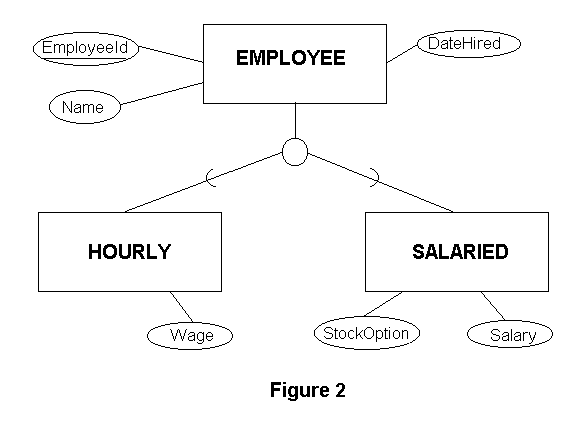
**SPECIALISATION** is the abstracting process of introducing new characteristics to an existing class of objects to create one or more new classes of objects. It involves taking a higher-level entity and using additional characteristics for generating lower-level characteristics. Example, by applying the characteristics “size” to car we can create full\_size, mid\_size, compact or subcompact cars. Thus, Specialisation is the reverse of Generalisation.

**GENERALIZATION and SPECIALISATION HIERARCHY -- Subtypes and Supertypes**

* This is the concept of categorizing or generalizing between **supertypes** and **subtypes** of entities.
* The basic notation used in McFadden, Hoffer and Prescott is shown in figure below.
* The **Supertype** entity is connected to the **Subtype** entities through lines and a circle.  The **U-symbol** indicates the Subtype is a subset of the Supertype.
* All of attributes of the Supertype are also **common** to the Subtype.
* Each Subtype entity has attributes that are **unique** to that Subtype - that is what makes the entity a subtype.
* All entities share a **common** **identifier attribute**.



Consider an **EMPLOYEE** supertype entity. This entity can have several different subtype entities (**HOURLY** and **SALARIED**), each with distinct properties not shared by other subtypes



DA

Wages

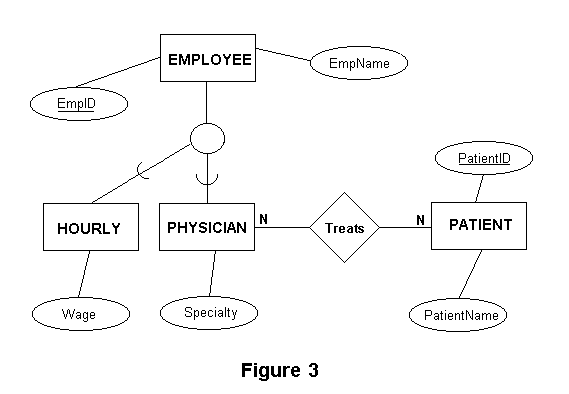
Basic

**Attribute Inheritance and Primary Keys for Generalization Hierarchies.**

* **Subtypes** inherit all of the attributes of the **Supertype**.  Note that an occurrence of a Subtype entity represents an occurrence of the **same** Supertype entity, to which it is connected by the common Primary key value in the supertype tuple and that subtype tuple
* The **primary key** of the supertype and subtype are always identical.
* The **Maximum:Minimum** cardinality between the supertype and subtype are always 1:0.  This means that an occurrence of the supertype, **EMPLOYEE** need not have an occurrence that is **SALARIED** nor **HOURLY** (the employee might be a **CONSULTANT** and have no unique attributes as a subtype), but an occurrence of a **SALARIED** employee entity (or **HOURLY**) **MUST HAVE** an occurrence of the **EMPLOYEE** supertype entity.

**When to Use a Subtype?**

Use a specific subtype if an occurrence of a subtype participates in a relationship. Example:  The subtype **PHYSICIAN** (of the supertype **EMPLOYEE**) participates in the relationship called **Treats** which links to the entity named **PATIENT.**

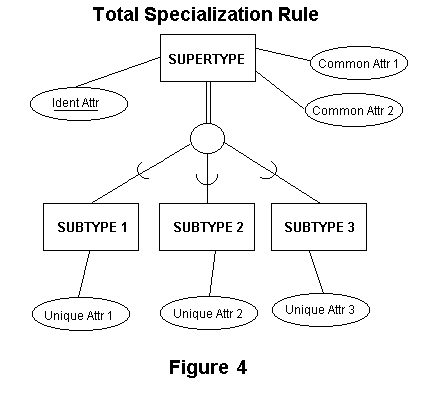


**Completeness Constraint.**

The **Completeness Constraint** addresses the issue of whether or not an occurrence of a Supertype must also have a corresponding Subtype occurrence.

The **Total Specialization Rule** specifies that an instance/object/entity of a Supertype must belong to one of it’s Subtypes.  This is diagrammed with a **double line** from the Supertype to the circle.

The **Partial Specialization Rule** specifies that an instance/object/entity of a Supertype may not belong to one of it’s subtypes. You can have a Supertype named **VEHICLE** and a Subtype named **AUTO**, but an occurrence of a vehicle need not be an automobile.



**Disjointness Constraints.**

This type of business rule deals with the situation where an occurrence of a Supertype may also have more than one Subtype occurrence. This is diagrammed by placing either the letter "**d**" or "**o**" inside the circle on the Generalization Hierarchy portion of the E-R diagram.

The **Disjoint Rule** says that this cannot occur. That is an occurrence of a Supertype must belong to only one of it’s subtypes.  Example:  a **PATIENT** can either be an **IN\_PATIENT** or an **OUT\_PATIENT**, but not the both at the same time.

The **Overlap Rule** says that this can happen. That is an occurrence of the supertype can belong to more than one subtypes. Example:  a **PART** can be either a **MANUFACTURED\_PART** or a **PURCHASED\_PART**.

The subtypes are usually (but may not be) mutually exclusive with no overlap.

**Subtype Discriminators.**

A **Subtype Discriminator** is an attribute of the Supertype that is used to "code" or denote which Subtype(or subtypes) an entity of the supertype belongs to. Example:  We might code **EMPLOYEE** supertypes as either "**H**" or "**S**" for **HOURLY** or **SALARIED** employee subtypes. That is have an attribute in the supertype Employee which will indicate whether the particular employee is of type Hourly or Salaried.

This attribute may enhance system performance during program coding of maintenance programs. Example just by referring at a row of the supertype we know in which subype we can find it’s further details.

**Supertype/Subtype Hierarchies.**

It is possible for a Subtype entity to be a Supertype entity in another relationship.

**12 Codds commandments/rules**

The Relational Model was propounded by Dr. E.F. Codd of IBM in 1972. For a package to qualify as an RDBMS, compliance with the 12 rules is required. In reality, all the 12 rules don’t carry the same degree of importance and some very good RDBMS products which exist today cannot even claim to obey more than 8 or so of these rules.

**RULE 0**

**Any truly relational database must be manageable entirely through its relational capabilities**

This is a single overall rule which tries to cover all others in a single sentence. It simply means that a RDBMS must be Relational, Wholly Relational and nothing but Relational.

**Rule 1 : The Information Rule**

**All information is explicitly and logically represented in exactly one way – by data values in tables**

It means that if an item of data does not reside somewhere in a table in the database, then it does not exist and this should be extended even to information such as tables, columns, views, column names, constraints and all other objects should be contained in a table-form

**Rule 2: The rule of guaranteed access**

**Every item of data must be logically addressable by restoring to a combination of table name, primary key value and column name.**

Though it is possible to retrieve individual data item in many different ways in a Relational/SQL environment, this rule means that any item can be retrieved by supplying the table name, the primary key values of the rows to easily search the row and the column name which is to be found or propagated.

**Rule 3 : The systematic treatment of null values**

It means that NULL values are representation of missing and inapplicable information. This support for null values must be consistent throughout the RDBMS and it should be independent of data type( a null value in a CHAR field must mean the same as null in an integer field)

**Rule 4 : The database description rule**

The description of the database is held and maintained using the same logical structures used to define the data, thus allowing users with appropriate authority to query such information in the same way and using the same language as they would query any other data in the database.

**Rule 5 : The comprehensive sub-language rule**

This means that the RDBMS must be completely manageable though it’s own sub-categories of SQL-language commands for enabling

Data-Definition(DDL)

Data-Manipulation(DML)

Data Authorisation(DCL)

Transaction boundaries(TCL)

Retrieval of data(DQL)

**Rule 6: The view updating rule**

**All views that can be updated in theory, can also be updated by the system**

It simply means that only simple views are updatable, Complex views are non-updatable,(but can be made updateable through the use of INSTEAD OF TRIGGERS in ORACLE). A simple view is a view which is based on a single table, has non calculated columns,no group function, no group by clause, no distinct, no rownum, no rowid or any such pseudocolumns. Simple view are again updatable subject to non-violation of constraints at the base table level.

**Rule 7 : The Insert and Update rule**

A RDBMS has to be capable of Inserting, Updating and Deleting data as a relational set(that is more than one row in a single go). Many RDBMS fail to achieve this and hence fall-back to a single-record-at-a-time procedural technique when it comes to data manipulation.

**Rule 8 : The Physical independence rule**

Users access to the database, via application programs, must remain logically consistent whenever changes to the storage representation or access methods to the data, are changed.

Example, if an index is built and destroyed by the DBA on a table, any user should still retrieve the same data from that table, although a bit slowly.

**Rule 9 : The Logical data independence rule**

Application programs must be independent of changes made to the base tables, with minor changes in the corresponding queries or use of views(makes changes in the query of the view)

TAB1 FRAG1 FRAG2

**A B C D A B A C D**

1 A C E 1 A 1 C E

4 A C F 4 A 4 C F

6 B D G 6 B 6 D G

2 B D H 2 B 2 D H

It should be possible to split a table vertically into more than one fragment as long as such splitting preserves all the original data(is non loss) and maintain the primary key in each and every segment

FRAG1 FRAG2 TAB1

**A B A C D A B C D**

1 A 1 C E 1 A C E

4 A 4 C D 4 A C D

6 B 6 D G 6 B D G

2 B 2 D H 2 B D H

It should be possible to combine base tables into one by way of a non-loss join

**Rule 10 : INTEGRITY RULES**

The relational model includes two general integrity rules

**Integrity Rule 1 (Entity Integrity):**

**If the attribute A of a relation R is a primary attribute of R, then A cannot accept null values**

**Integrity Rule 1 (Referential Integrity):**

**It is concerned with foreign key, that is, with attributes of a relation having domains that are those of the primary key of another/same relation.**

**Rule 11 : Distribution rule**

**A RDBMS must have distribution independence**

This is one of the attractive aspects of RDBMS packages. It simply means that databases built on relational model are suited to client-server architecture.

**Rule 12 : No Subversion rule :**

If an RDBMS supports a lower level language that permits for example, row-at-a-time processing, then this language must not be able to bypass any integrity rules or constraints of the relational language.

**What is a Transaction?**

A collection of operations that perform a single logical function in a database application

OR

A logical unit of work, typically involving several database operations

Following are the Properties of transaction, it is also call ACID properties of transaction:

Atomicity: Either all operations of the transaction are reflected properly in the database or none

Consistency: A correct execution of the transaction must take the database from one consistent state to another

Isolation: A transaction should not make its updates visible to other transactions until it is committed

Durability: After transaction complete successfully, the changes it has made to the database persist, even if there are system failure

**NORMALISATION**

Attribute X can be defined as determinant if it uniquely defines the attribute value Y in a given relationship or entity. To qualify as determinant attribute need **NOT** be a key attribute. Usually dependency of an attribute is represented as X → Y, which means attribute X decides attribute Y, that is, for a given value of X, there is one value of Y

***Example:*** *In RESULT relation, Marks attribute may decide the grade attribute. This is represented as* ***Marks*** → ***Grade*** *and read as Marks decides Grade.*

Consider the following Relation

REPORT (Student#,Course#, CourseName, IName, Room#, Marks, Grade)

Where:

Student# - Student Number

Course# - Course Number

CourseName - Course Name

IName - Name of the inst ructor who delivered the course

Room# - Room number which is assigned to respective instructor

Marks - Scored in course Course# by student Student#

Grade - Obtained by student Student# in course Course#

In the RESULT relation, Marks attribute is not a key attribute. Hence it can be concluded that key attributes are determinants but not all the determinants are key attributes.

Student# and Course# together (called composite attribute) defines EXACTLY ONE value of marks. This can be symbolically represented as

**Student# ,Course#**  → **Marks**

This type of dependency is called as functional dependency. In above example marks is **functionally dependent** on Student# , Course#.

Other funct ional dependencies in above examples are:

Course# → CourseName,

Course# → IName (Assuming one course is taught by one and only one instructor)

IName → Room# (Assuming each inst ructor has his/ her own and non-shared room)

Marks → Grade.

Formally we can define functional dependency as: In a given relation R, X and Y are attributes. Attribute Y is **functionally dependent** on attribute X if each value of X determines EXACTLY ONE value of Y. This is represented as:

X → Y

However X may be composite in nature

Normalisation is a process of organising your data into tables in such a way that there is minimal or no redundancy, data can abide by the integrity/business constraints and there are no insertion, updation and deletion anomalies.

Consider the following data which shows preferences of faculty members for teaching courses. Here, we allows cross-departmental teaching, that is, a faculty member of Computer Science department may have preference for teaching a course of the Mathematics department. The following data is in UNNORMALISED form, since each row may contain multiple set of values for certain columns. Such multiple values for a given column in a single row is called NONATOMIC values. For example, the row corresponding to the preferences of the faculty in the Computer Science Department has two values for the column Professor. For a particular professor say professor Smith, the row indicating his preferences contains three values for the column Course.

Such UNNORMALISED relation contains NONATOMIC values.

|  |  |  |  |
| --- | --- | --- | --- |
| PROF\_DEPT | PROF | COURSE | COURSE\_DEPT |
| Computer Science | Smith | 353 | Computer Science |
| 379 | Computer Science |
| 221 | Decision Science |
| Clark | 353 | Computer Science |
| 351 | Computer Science |
| 379 | Computer Science |
| 456 | Mathematics |
| Chemistry | Turner | 353 | Computer Science |
| 456 | Mathematics |
| 272 | Chemistry |
| Mathematics | James | 353 | Computer Science |
| 379 | Computer Science |
| 221 | Decision Science |
| 456 | Mathematics |
| 469 | Mathematics |

Hence, the above unnormalised data is converted to the following format, so that it does not contain non-atomic values.

|  |  |  |  |
| --- | --- | --- | --- |
| PROF\_DEPT | PROF | COURSE | COURSE\_DEPT |
| Computer Science | Smith | 353 | Computer Science |
| Computer Science | Smith | 379 | Computer Science |
| Computer Science | Smith | 221 | Decision Science |
| Computer Science | Clark | 353 | Computer Science |
| Computer Science | Clark | 351 | Computer Science |
| Computer Science | Clark | 379 | Computer Science |
| Computer Science | Clark | 456 | Mathematics |
| Chemistry | Turner | 353 | Computer Science |
| Chemistry | Turner | 456 | Mathematics |
| Chemistry | Turner | 272 | Chemistry |
| Mathematics | James | 353 | Computer Science |
| Mathematics | James | 379 | Computer Science |
| Mathematics | James | 221 | Decision Science |
| Mathematics | James | 456 | Mathematics |
| Mathematics | James | 469 | Mathematics |

FNF

A Relation is said to be in the FIRST NORMAL FORM or 1NF, if the values in the domain of each attribute of the relation are ATOMIC. In other words, only one value is associated with each attribute of a row, and the value is not a list of values(as in the above example) or a composite value(as shown in the example below).

In this example, the data is not un-normalised, but the column Order\_details contains composite values(value of order number and value of order date)

|  |  |  |  |
| --- | --- | --- | --- |
| Order\_details | Item\_code | Quantity | Price\_per\_unit |
| 1456 of 26/02/1989 | 3687 | 52 | 50.40 |
| 1456 of 26/02/1989 | 4627 | 38 | 60.20 |
| 1456 of 26/02/1989 | 3214 | 20 | 17.50 |
| 1886 of 04/03/1992 | 4629 | 45 | 20.25 |
| 1886 of 04/03/1992 | 4627 | 30 | 60.20 |
| 1788 of 04/04/1992 | 4627 | 40 | 60.20 |

FNF

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order\_No | Order\_date | Item\_code | Quantity | Price\_per\_unit |
| 1456 | 26/02/1989 | 3687 | 52 | 50.40 |
| 1456 | 26/02/1989 | 4627 | 38 | 60.20 |
| 1456 | 26/02/1989 | 3214 | 20 | 17.50 |
| 1886 | 04/03/1992 | 4629 | 45 | 20.25 |
| 1886 | 04/03/1992 | 4627 | 30 | 60.20 |
| 1788 | 04/04/1992 | 4627 | 40 | 60.20 |

The representation of data for the courses that a faculty member would prefer to teach has the following drawbacks : The fact that a given professor belongs to a particular department is repeated a number of times. The fact that a particular course is offered by a particular department is also repeated a number of times. These repetitions can lead to inconsistencies. For example, if a Professor’s department is changed, unless all the rows where that professor appears are changed, we would have inconsistencies in the database. Also, if the relation between courses and the departments to which the courses belong is kept in this table only, then a new course cannot be entered(without NULL values, since may be all the four columns of the above table form a composite primary key) unless some professor prefers to teach it. Also, deletion of the only professor who prefers to teach a given course will cause loss of information of the fact that such a course exists, and also loss of the information about the department to which this course belongs.

SNF

A Relation is said to be in the SECOND NORMAL FORM or 2NF, if it is in the FNF and if all the non-key(non-prime) attributes are fully(wholly) functionally dependent on the relation’s key attribute(s). It ensures that there are no Partial dependencies

Consider the following relation :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Stud\_No** | **Course** | Phone\_No | Major | Prof | Grade |
| 1 | 353 | 5654345 | Comp Sci | Smith | A |
| 2 | 329 | 8765432 | Chemistry | Turner | B |
| 1 | 328 | 5654345 | Comp Sci | Clark | B |
| 4 | 456 | 6753234 | Physics | James | A |
| 5 | 293 | 6754122 | Desc Sci | Cook | C |
| 6 | 491 | 9876543 | Mathematics | Lamb | B |
| 6 | 356 | 9876543 | Mathematics | Bond | Going On |
| 1 | 492 | 5654345 | Comp Sci | Cross | Going On |
| 9 | 379 | 5676453 | English | Broes | C |

Here, the key of the above relation is Stud\_no+Course(Composite Key). The following Functional Dependencies exist :{Stud\_No 🡪 Phone\_no, Stud\_No 🡪 Major, Stud\_no, Course 🡪Grade, Course 🡪 Prof}.

Here, the attribute Phone\_no which is not a part of the key of the relation is called non-key attribute, and is not functionally dependent on the whole key(Stud\_no+Course), but is dependent on a part of the whole key, that is on Stud\_no. Similarly, the attribute major. Also, same is the case with the attribute Prof, which is dependent on a part of the whole key, that is on Course, assuming that a Course will be taught by one Professor only. Only, the attribute Grade is fully functionally dependent on the whole key(Stud\_no+Course).

This relation has several problems :

Redundancy

In the above table, which is in 1NF, the Major and Phone\_no of a student are repeated several times. The aim of a database system is to reduce redundancy, meaning that the information should be stored only once. Storing information several times leads to wastage of storage space and also increase in the total size of the data stored. Such redundant data can lead to inconsistency, as explained in the next point

Update Anomalies

Multiple copies of the same data may lead to update anomalies or inconsistencies when an update is made and only some of the multiple copies of the redundant data are updated. Thus, a change in the Phone\_no of Jones must be made, for consistency, in all tuples pertaining to the student Jones. If any one of the tuples is not changed to reflect the new Phone\_no of Jones, there will be inconsistency in the data.

Also, if another relation also establishes a relationship between a course and a professor who teaches that course, then the information stored in these two relations must be maintained for consistency. For example, the following relation contains the schedule of lectures of courses for the faculties.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course | Prof | Room | Room\_Capacity | Enrol\_Limit |
| 353 | Smith | A532 | 45 | 40 |
| 328 | Clark | H940 | 400 | 300 |
| 329 | Turner | B278 | 50 | 45 |
| 456 | James | D110 | 50 | 45 |

Insertion Anomalies

If this is the only table that shows the relation between a faculty member and the course he/she teaches, then the fact that a given professor teaches a given course cannot be entered in the database unless a student is registered for that course.

Deletion Anomalies

If a student discontinues from a particular course, the information as to which professor is teaching the course will be lost, if that course was only being studied by that student who has now going to discontinue it. This is more true when there is no other relation indicating the relationship between courses and professors teaching those courses.

Hence the following :

|  |  |  |
| --- | --- | --- |
| Stud\_No | Phone\_No | Major |
| 1 | 5654345 | Comp Sci |
| 2 | 8765432 | Chemistry |
| 4 | 6753234 | Physics |
| 5 | 6754122 | Desc Sci |
| 6 | 9876543 | Mathematics |
| 9 | 5676453 | English |

Here, the non-key attributes Phone\_no and Major are fully functionally dependent on the key attribute Stud\_no

|  |  |
| --- | --- |
| Course | Prof |
| 353 | Smith |
| 329 | Turner |
| 328 | Clark |
| 456 | James |
| 293 | Cook |
| 491 | Lamb |
| 356 | Bond |
| 492 | Cross |
| 379 | Broes |

Here, assuming that a course will be taught by one professor only, the non-key attribute Prof is fully functionally dependent on the key attribute Course

|  |  |  |
| --- | --- | --- |
| Stud\_No | Course | Grade |
| 1 | 353 | A |
| 2 | 329 | B |
| 1 | 328 | B |
| 4 | 456 | A |
| 5 | 293 | C |
| 6 | 491 | B |
| 6 | 356 | Going On |
| 1 | 492 | Going On |
| 9 | 379 | C |

Here, the non-key attribute Grade is fully functionally dependent on the key attribute Stud\_No+Course.

Also, consider the following example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order\_No | Order\_date | Item\_code | Quantity | Price\_per\_unit |
| 1456 | 26/02/1989 | 3687 | 52 | 50.40 |
| 1456 | 26/02/1989 | 4627 | 38 | 60.20 |
| 1456 | 26/02/1989 | 3214 | 20 | 17.50 |
| 1886 | 04/03/1992 | 4629 | 45 | 20.25 |
| 1886 | 04/03/1992 | 4627 | 30 | 60.20 |
| 1788 | 04/04/1992 | 4627 | 40 | 60.20 |

Consider the above table in FNF. Assuming that an item(item\_code) will not be repeated in an order(order\_no), the Key attribute(Primary key, in this case Composite Primary key) will be ORDER\_NO+ITEM\_NO.

However, Price\_per\_unit is functionally dependent on Item\_code; and Order\_date is functionally dependent on Order\_No. Thus, the relation is not in 2NF. Therefore, it is split into 3 tables as follows to give 2NF :

|  |  |
| --- | --- |
| Order\_No | Order\_date |
| 1456 | 26/02/1989 |
| 1886 | 04/03/1992 |
| 1788 | 04/04/1992 |

Now order\_date(non key attribute) is functionally dependent on order\_no(key attribute)

|  |  |
| --- | --- |
| Item\_code | Price\_unit |
| 3687 | 50.40 |
| 4627 | 60.20 |
| 3214 | 17.50 |
| 4629 | 20.25 |

Price\_unit(non key attribute) is functionally dependent on Item\_code(key attribute)

|  |  |  |
| --- | --- | --- |
| Order\_no | Item\_code | Qty |
| 1456 | 3687 | 52 |
| 1456 | 4627 | 38 |
| 1456 | 3214 | 20 |
| 1886 | 4629 | 45 |
| 1886 | 4627 | 30 |
| 1788 | 4627 | 40 |

Qty(non key attribute) is functionally dependent on Order\_no+Item\_code(key attribute)

TNF

A Relation is said to be in the THIRD NORMAL FORM or 3NF, if it is **does not allow Transitive Dependencies,** that is, No non-key attributes are functionally dependent on any other non-key attribute

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course | Prof | Room | Room\_Capacity | Enrol\_Limit |
| 353 | Smith | A532 | 45 | 40 |
| 351 | Smith | C320 | 100 | 60 |
| 355 | Clark | H940 | 400 | 300 |
| 456 | Turner | B278 | 50 | 45 |
| 459 | James | D110 | 50 | 45 |

This relation implies the Course and the Professor who will teach that course and the room in which that professor will teach that course, assuming that a course will be taught by only one professor in a particular room. Room\_capacity indicates the seating capacity of that room, and Enrol\_Limit is the enrolment limit for that course and it must be lesser than the Room\_capacity, as the room in which a course will be taught is pre-decided.

Considering that a Course is scheduled in a given Room and since the room has a given maximum number of seats, there is a Functional Dependency Room 🡪 Room\_Capacity, and hence by Transitivity Course 🡪 Room 🡪 Room\_Capacity. Thus, the Functional Dependencies are {Course 🡪 (Prof, Room, Room\_capacity, Enrol\_limit), Room 🡪 Room\_Capacity}. Also, there is another transitive dependency Room 🡪 Room\_Capacity 🡪 Enrol\_limit, since the Enrol\_limit has to be lesser than the Room\_capacity of the Room.

The presence of all these Transitive dependencies will cause the following problems : The capacity of a room cannot be entered in the database unless a course is scheduled in that room(assuming that course is the Primary key). The capacity of a room in which one of the courses is scheduled will be deleted if the only course scheduled in that room is deleted.

Hence we decompose the above into the following two :

Course\_Details

|  |  |  |  |
| --- | --- | --- | --- |
| Course | Prof | Room | Enrol\_Limit |
| 353 | Smith | A532 | 40 |
| 351 | Smith | C320 | 60 |
| 355 | Clark | H940 | 300 |
| 456 | Turner | B278 | 45 |
| 459 | James | D110 | 45 |

Room\_details

|  |  |
| --- | --- |
| Room | Room\_Capacity |
| A532 | 45 |
| C320 | 100 |
| H940 | 400 |
| B278 | 50 |
| D110 | 50 |

In the first relation, Course\_details, the functional dependencies are {Course 🡪 Prof, Course 🡪 Room, Course 🡪 Enrol\_limit}. In the second relation, Room\_details, the functional dependency is {Room 🡪 Room\_capacity}.

These Relations are now in the Second Normal Form.

However, the relation, Course\_details, has a Transitive Dependency {Course 🡪 Room 🡪 Enrol\_limit}. So, if in this table the value of Enrol\_limit is changed, then it may not be proper if the Room does not allow that many number of students for that course. That is change in Enrol\_limit or the Room may demand a change in the corresponding Room or Enrol\_limit respectively or else that rows data will not be treated as synchronous because of the mis-match in Enrol\_limit and Room. Hence we further split it as follows :

|  |  |  |
| --- | --- | --- |
| Course | Prof | Enrol\_Limit |
| 353 | Smith | 40 |
| 351 | Smith | 60 |
| 355 | Clark | 300 |
| 456 | Turner | 45 |
| 459 | James | 45 |

|  |  |
| --- | --- |
| Course | Room |
| 353 | A532 |
| 351 | C320 |
| 355 | H940 |
| 456 | B278 |
| 459 | D110 |

Room\_details

|  |  |
| --- | --- |
| Room | Room\_Capacity |
| A532 | 45 |
| C320 | 100 |
| H940 | 400 |
| B278 | 50 |
| D110 | 50 |

Consider the following in 2NF:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Roll\_No | Name | Department | Year | Hostel\_name |
| 1784 | Ram | Physics | 1 | Ganga |
| 1648 | Sham | Chemistry | 1 | Ganga |
| 1768 | Bharat | Maths | 2 | Kaveri |
| 1848 | Arjun | Botany | 2 | Kaveri |
| 1682 | Nakul | Geology | 3 | Krishna |
| 1485 | Sahadeo | Zoology | 4 | Godavari |

Assuming that the 1st Year, 2nd Year, 3rd Year and 4th Year students stay in Hostels Ganga, Kaveri, Krishna and Godavari respectively, then in such a case, the non-key attribute Hostel\_name is dependent on the non-key attribute Year. Therefore, it is split into 2 tables to avoid unnecessary duplication of data as follows to give 3NF :

Stud\_mast

|  |  |  |  |
| --- | --- | --- | --- |
| Roll\_No | Name | Department | Year |
| 1784 | Ram | Physics | 1 |
| 1648 | Sham | Chemistry | 1 |
| 1768 | Bharat | Maths | 2 |
| 1848 | Arjun | Botany | 2 |
| 1682 | Nakul | Geology | 3 |
| 1485 | Sahadeo | Zoology | 4 |

Hostel\_Mast

|  |  |
| --- | --- |
| Year | Hostel\_name |
| 1 | Ganga |
| 2 | Kaveri |
| 3 | Krishna |
| 4 | Godavari |

**3.4.4. Boyce Codd Normal Form (BCNF)**

A relation is said to be in Boyce Codd Normal Form (BCNF) if and only if all the determinants are candidate keys. BCNF relation is a strong 3NF, but not every 3NF relation is BCNF.

Consider the following Result table structure.

**St**

Student# EmailID Course# Marks

**101** [Davis@myuni.edu](mailto:Davis@myuni.edu) **M4 82**

**102** [Daniel@myuni.edu](mailto:Daniel@myuni.edu) **M4 62**

**101** [Davis@myuni.edu](mailto:Davis@myuni.edu) **H6 79**

**103** [Sandra@myuni.edu](mailto:Sandra@myuni.edu) **C3 65**

**104** [Evelyn@myuni.edu](mailto:Evelyn@myuni.edu) **B3 77**

**102** [Daniel@myuni.edu](mailto:Daniel@myuni.edu) **P3 68**

**105** [Susan@myuni.edu](mailto:Susan@myuni.edu) **P3 89**

**103** [Sandra@myuni.edu](mailto:Sandra@myuni.edu) **B4 54**

**105** [Susan@myuni.edu](mailto:Susan@myuni.edu) **H6 87**

**104** [Evelyn@myuni.edu](mailto:Evelyn@myuni.edu) **M4 65**

In the RESULT table, we have two candidate keys namely **Student#Course#** and **Course# EmaiIId**. Course# is overlapping among those candidate keys. Hence these candidate keys are called as **“overlapping candidate keys”**

The non-key attribute, Marks is non-transitively and fully functionally dependant on key attributes. Hence this is in 3NF. But this is not in BCNF because there are four determinants in this relat ion namely:

Student# (Student# decides EmailiD)

EMailID (EmailID decides Student#)

Student# Course# (decides rest of the attributes in RESULT table)

Course# EMailID (decides rest of the attributes in RESULT table)

All above determinants are not candidate keys. EMailID decides Student# but EMailID on its own is not a candidate key. Similarly Student# decides EMailID of a student but Student# alone is not a candidate key. Only combination of Student# Course# and Course# EMailID are candidate keys.

To make this table BCNF, we need to split this table into the following structure:

**STUDENT TABLE**

**Student# EmailID**

**Student # EmailID**

**101** Davis@myuni.edu

**102** Daniel@myuni.edu

**103** Sandra@myuni.edu

**104** Evelyn@myuni.edu

**105** Susan@myuni.edu

**Student**

**Student # Course# Marks**

**101 M4 82**

**102 M4 62**

**101 H6 79**

**103 C3 65**

**104 B3 77**

**102 P3 68**

**105 P3 89**

**103 B4 54**

**105 H6 87**

**104 M4 65**

Now both the tables are not only in 3NF, but also in BCNF because all the determinants are candidate keys. In the first table, Student# decides EMailID and EMailID decides Student# and both are candidate keys.

In second table, Student# Course# is only determinant and candidate key. Hence it qualifies BCNF definition that every determinant must be a candidate key.

If the table has only one non-composite candidate key and if it is in 3NF, then the table will also be in BCNF.

Basically 2NF and 3NF takes away the redundancy, anomalies which exist among the key and non-key attributes on other hand BCNF takes away the redundancy, anomalies which exist among the key attributes

**Denormalization**

A normalized design will often store different but related pieces of information in separate logical tables (called relations). If these relations are stored physically on separate disk files, writing a query that draws information from several relations can be slow. If many relations are joined, it may be prohibitively slow. There are two strategies for dealing with this.

1) The preferred method is to keep the logical design normalized, but allow the DBMS to store additional redundant information on disk to optimize query response. In this case it is the DBMS software's responsibility to ensure that any redundant copies are kept consistent. This method is often implemented in as indexed views or materialized views.

2) The other approach is to denormalize the logical data design. With proper care, this can achieve a similar improvement in query response, but at a cost―it is now the database designer's responsibility to ensure that the denormalized database does not become inconsistent. This is done by creating rules in the database called triggers, that specify how the redundant copies of information must be kept synchronized. It is the increase in logical database design and the added complexity of the additional constraints(thru triggers) that make this approach hazardous. Moreover, constraints introduce slowing down writes (INSERT, UPDATE, and DELETE). This means a denormalized database under heavy write load may actually offer *worse* performance than its functionally equivalent normalized counterpart.