Course Code: **CSC 422**

Course Title: **Database**

Course Developer/Writer: **Dr. A.T. Akinwale**

&

Miss A. J. Ikuomola

Department of Computer Science

College of Natural Science

University of Agriculture Abeokuta,

Ogun State, Nigeria

UNIVERSITY OF AGRICULTURE, ABEOKUTA

WEEK – ONE

DATABASE DESIGN

Designing a database is an art process similar to building a house. There are many techniques professionals via design databases. Before proceeding forward on database, there is a need to know the basic concepts of database.

WHAT IS DATABASE?

A database is a representation of facts, concepts or instruction in a formalized manner suitable for communication, interpretation or processing by human or automatic means.

A database can be defined as central pool of data which is shared by various user of an organization.

A database system can be defined as a representation of an information system in a computer. A database system consists of the data structure, the suite of programs and the base used to put an information system on a computer.

Database management: comprises prises of software, hardware and organizational techniques to manage a database.

Data processing: is the execution of systematic sequence of operations performed upon data.

By Daniel Martin- He defines database as a collection of data that obeys three criteria:

- Exhaustively
- Non _redundancy
- Appropriate structure

- ❖ Exhaustively means that all the data about the subject are actually present in the database.
- ❖ Non_redundancy means that each individual piece of data exit only once in the database.
- ❖ Appropriate structure means that the data are stored in such a way as to minimized the cost of the expected processing and storage.

Some author says that database is with an “open” structure such database that is open database allow for easy change of field dimension (e.g. increasing a given field size from six to seven digits)

- ❖ For easy addition of new fields
- ❖ For easy change in data linkage (for example, link a customer record to all transactions performed on his account: invoices, payments e.t.c)

The problem with open database is that, it is very costly in terms of processing time, memory space and disk storage.

Customer file

Name

Address

Credit balance

Cash payment

Amount due

Current amount

payment_due_file

vendor number

vendor name

vendor address

invoice amount

date payment due

Name	Address	Credit Balance	Cash Payment	Amount Due	Current Amount

Vendor no	Vendor name	Vend or address	Invoice amount	Date payment due

Exhaustively implies the presence within the database of all information pertaining to a given customer or to a payment. Non_redundancy exclude the possibility that certain pieces of data exist more than once within a database. For example, if the payment _due_ file of the database contains the name and the address of each debtor. And this information is already stored in "customer file", it is therefore redundant.

Pointer: is an arrow to link one record to another in another file.

A database represent a system to be computerized .Whenever this system is referred to in computerized form, it is called a database system. The concept of a database is to solve problem of redundancy. This is so because data is stored only once, hence there is no waste of storage. When data get updated, there is no question of it being in an in consistence state.

Another concept of a database is that it is in position of being stored by many users. For this to happen, there is separation of data organization and access technique from application programs, The user have a view of a database in conformity with the way they want to see them.

It follows therefore that the concept of a database is potentially capable of solving both redundancy and the loss of flexibility problems executed in conventional programming.

The organ that can perform these task is software package and is called Database Management System (DBMS). DBMS is a software that solves the problem of conventional programming. Convention is the usual way of doing things.

THE CONCEPTUAL DATA BASE

Assuming that there is a change in colour of a car from red to black. This change must be captured in the conceptual data base in a faithful manner. This means the colour of the car should be correctly updated to reflect the current colour.The following operation in computer allow us to make change in the conceptual data base.

- Insert information
- Delete information
- Update or modify information
- Retrieve information

Let us suppose that at a certain stage in the life of the university, the need for computerization is felt. This is because it is found very difficult to keep track of the student in the university vis-à-vis the department to which they are affiliated. Consequently, a student record is defined containing the field of interest. Let these be the name of the student, the department to which he is affiliated, his age, the last qualifying exam ination passed by him, and specialization for which he is enrolled in the department.

The Student Record

NAME	DEPT	AGE	LAST EXAM	SPECIALIZATION

After the record has been defined, a programming language like COBOL, PL/1 OR PASCAL which support which supports the notion of a record selected and to program the two stages outlined above selected and used to program the two stages outlined above. Later, the university decides to computerize information regarding faculty numbers and their affiliation to departments. A new record is defined to capture this information and may look as in this figure II.

Faculty Member Record

NAME	DEPT	AGE	QUALIFICATION	RESEARCH INTEREST

The university also tries to computerize the course offered by the various department and the courses taught by the various faculty members respectively. These records are as follow:

The course_ offered record lll

COURSE_NO	DEPT	PRE_REQUSITES

The course_taught record lV

COURSE_NO	MEMBER OF FACULTY MEMBER	ROOM NO IN WHICH LECTURES ARE HELD

The field course_no appears in the course_offered and course _taught records. This means that all records which contain this field carry information about the course number; clearly, this implies that there is a certain amount of information which in term results in a certain amount of waste of storage.

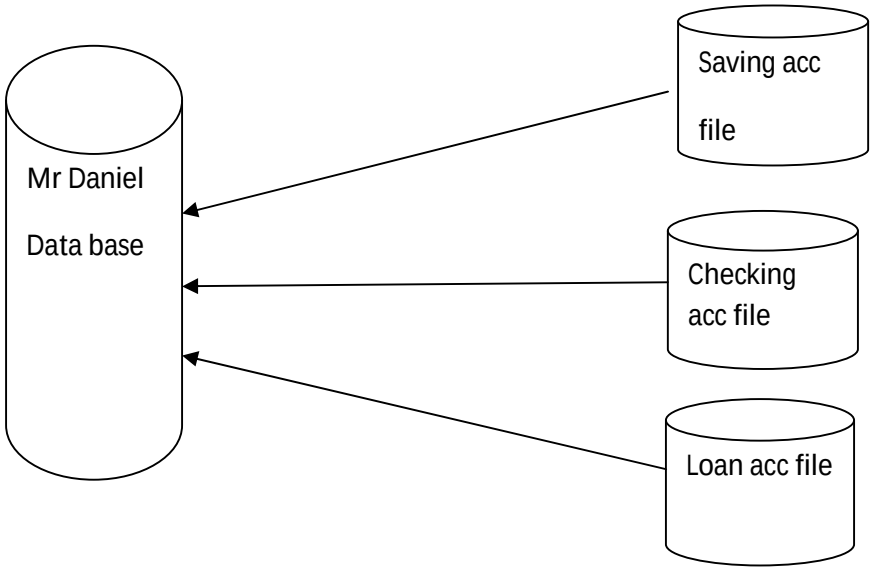
Data base represent a new approach to storing data. In the data base of any firm, the problem in company A may be different from the problem in company B; hence the implementation procedure of their data base may vary with each factor. The data base is introduced a new way of storing and accessing processed data. The data base represents consolidation of files. once files are consolidated, it reduce input and output devices, the rate of update process, the rate of file design, it eliminate the copies of backups and it gives cleaner operating environment.

OBJECTIVES OF DATA BASE

- Data base allow consolidating data entries among files.
- Data base manages shared files

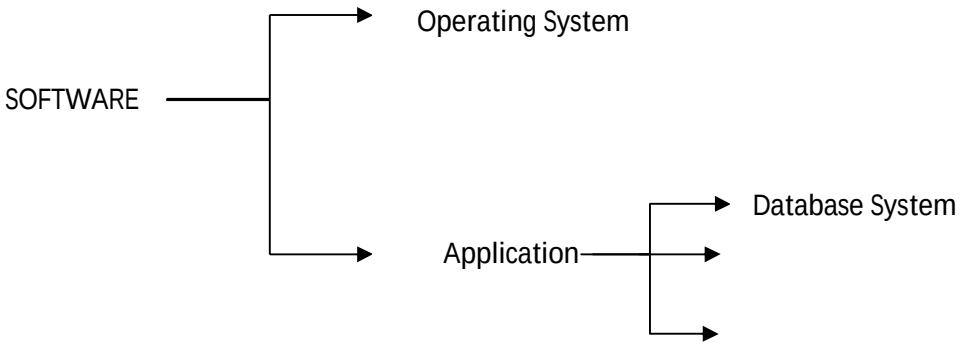
e.g. Mar Daniel has his saving account and checking account at the lst Bank. Assuming, he acquired a car loan from lst Bank when he purchased his car. This means Mr. Daniel's name and

other personal data appear in files belonging to saving account, checking account and loan account any time, Mr Daniel makes deposit, it is certain that his name will appear in several accounts.



This means Data base helps to share files.

- Database allows fewer programs and simpler execution procedures
- Database occupies all the consequences of an incoming transaction at one time.
- Database reduces the rate of data redundancy.
- Database provides the integrity of data.



Database system is designed to handle data store in a database. This system is to update and retrieve information stored in the database when request are issued. During database operation, the operating system would load database system and application program. The application program can execute any instruction until it wants data from database. At this point, the database

gains control and handle the database request s. The database system resides in storage with the application program when there is a need to use a database.

Knowledge Know-how on Database system

DBS – Contains a series of system program

DBS – Manages its database

DBS – Interfaces with application program & operating system

Data Base Management – A database is a collection of carefully integrated files. The data in these files must be managed. There is a special software packages available to manage a database. This software packages is known as Data Base Management System.

A database management system is the software that manages the database and provides facilities for storing, accessing, and maintaining the data. DBMS is developed by computer manufacturers or software house. These software packages tend to be referred to by several names, among them "data base system", "data management system", "DBMS". The software is in effect an extension to the operating system. It acts as an interface between the programs which need to access data in the database and the database itself, allowing the data to be retrieved or updated.

HISTORY OF DATABASE MANAGEMENT SYSTEM

DBMS – have been in use for almost **two** decades. It was only about a decade ago that this field has come to be recognized as a major discipline in computer science. It has been only recently that DBMS have been studied systematically both from the user and the system points – of – view. The beginning of DBMS development was marked by Database Task Group (DBTG) which published a report in 1971, called CODA 71.

The report stated that

- Proposal for the development of Network model
  Another group from IBM Research Laboratories at San Jose under the supervision of E.F Codd published a report in 1970, called CODD 70.

The report stated that

- A proposal for the development of relational model for few years, a great debate raged between the proponents of Database Task Group Report and CODD 70 report. Each side claimed that their views were the better ones. During this period, the entire area of DBMS was in a state of abject turmoil (miserable disorder). There were arguments and counter-arguments. What

CODA 71 claimed as advantages, the CODD 70 group claimed as disadvantages. The basic tenets of each proposal were questioned, examined and re-examined over and over again. Finally, one major fact stood out , that was,

- Network model was shown to be reasonably efficiently implementable because it could handle large size of data base for over billion of bytes.
- Relational model could only support relatively small database.

The proposal of CODA 71 of Network model was the basis of the design and implementation of Data Base Management System. All the experts began to arrive at a consensus. The great database has served the purpose. A new discipline of computer science is born.

In 1976, a study group called ANSI 76 had a new concept on DBMS. This concept opened up a new area of DBMS. It emphasized the role of a DBMS as a tool for representing in computer, a model of real world. This ANSI 76 report considered DBMS as software to manage a large pool of data, called data base. The earlier researcher working towards the view of DBMS before ANSI 76 group put final approval.

- The future shape of DBMS had been decided.
- The structure of DBMS had been put up.

It is now remained for us to realize this structure not only in a neat and clean form but in reasonably efficient manner.

The ANSI study group recognizes three functions that are necessary in order to support database system:

- The enterprise manager function performed by the enterprise manager
- The database administrator function performed by database administrator.
- The application administrator function performed by application administrator.

That is,

The enterprise manager is responsible to ensure that a proper and adequate system analysis is done which meet the need of the enterprise. The database administrator exercises control over the data structure and the storage methods. He is concerned with overall efficiency of the implementation.

The application administrator is responsible to split up the centralized pool of data among various users in such a way that each user;

- Has access to all databases he or she needs.
- Has illusion that the logical structure of the data available to him or her is conformity with the demand.

ANSI=American National Standard Committee on Computers and Information Processes

SPARC=Standard Planning and Requirement Committee.

TYPES OF DATA BASE

There are four types of database:

- Bibliographic database
- Knowledge data base
- Graphics _ oriented data base
- Decision _making data base

- **Bibliographic database:** BDB have data which is free of a format. They display little or no format. Such database are often used library information system .Data could be composed of abstract of books. It could also compose of keywords and key phrases. It is possible using these keywords and key phrases to select documents. If desired, the source of the document could before original document.

- **Knowledge database:** KDB are used in artificial intelligent applications. The data in these KDB is discrete and formatted .in these KDB, there are many kinds of data with only a very few occurrence of each kinds. Clearly, such data bases have the peculiarity that the size of the data is almost as large as the definition of the data.

- **Graphics database:** GDB could possibly be used in computer_aided_design. The data in GDB is characterized as being active. This means the data is capable of being executed. For examples, we could have store a ‘’triangle’’ in the data base. Upon our retrieving the triangle, the computer system could invoke a procedure to draw a triangle or a graphic screen. In these scene, the data namely triangle is an active piece of data. Whereas in bibliography and knowledge data base, data cannot be executed in a computer.

- **Decision-making database:** DMDB are used in corporate management and allied administrative tasks. Using DMDB, it is possible to handle the problems like resource planning, sales forecasting, profitability of business e.t.c. depending upon the kind of data bases handled. Data base management system can be classified as example, bibliography data base management system, knowledge data base management system, graphics_oriented data base management system and decision-making data base management system.

HIERARCHY OF DATA

Character, fact, record, file and data base form a hierarchy of data.

The basic building block is a character. The character consist of upper and lower-case, numeric digits or symbol. Upper or lower-case of letters are A,a B ,b,.....Z,z . Numeric digits are 0,1,2,3.....9.Symbols involves commas, question mark, plus division e.t.c. upper and lower-case letter are called alphabetic character. Numeric digit are called numeric symbol are character. Symbols are called special character. A combination of the three is referred to as alphanumeric characters (# 2B,₦2.50K). A computer can accept both alphanumeric and number and store them in memory. Character are put together to form a fact. A fact is also called a field. A fact or field is a number, an item, a word, a name or a combination of characters.

TYPES OF FIELD IN DATA BASE

- Character /text
- Numeric
- Data
- Logical
- Memo

A field is an individual item of data within a record. Facts are put together to form a record. A record is a related item of data in a file. An employee record in a company would be a collection of facts about one employee. These facts would include the employee's name, address, department, phone, position, pay rate, earning made to date, and e.t.c

Record are combined together to make a file. A collection of related records is a file e.g. A collection of all employee records for one company would be an employee file. What is an inventory file? It is a collection of all inventory records for a particular company.

TWO TYPES OF FILE

- Permanent file or master file
- Transaction file or detail file

The file described as employee file is an example of permanent file. The data stored in a permanent file or master file should be accurate and current. A permanent file of all the customers who own money to a company is an account receivable master file. An account payable master file containing all suppliers to which the organization owes money.

A transaction file is a temporary file which represents the transactions of the organization. The data stored in transaction file can be re –adjusted.

File are combined together to make a database. The heart of most organization is data. A data base is the collection of integrated and related master files. An organization uses data as raw materials to be stored in database. Once the data have been processed, they are called information.

TYPES OF DATA

- Numeric data
- Alphanumeric data

Numeric data is expressed in number e.g. age= 35, date of birth is 1970.
Numeric data contains only numeric character or numbers

Alphanumeric data is composed of combination of letters numbers, or special punctuation characters such as: name –Abeokuta

Address-17, Ibadan road

Date -26th October, 1998

DATA STRUCTURE OR STRUCTURE OF DATA.

Structure of data is the composition of records into files generating information.

Let us take an example of long-distance telephone data. When you make a long distance telephone call, the following item of data is recorded:

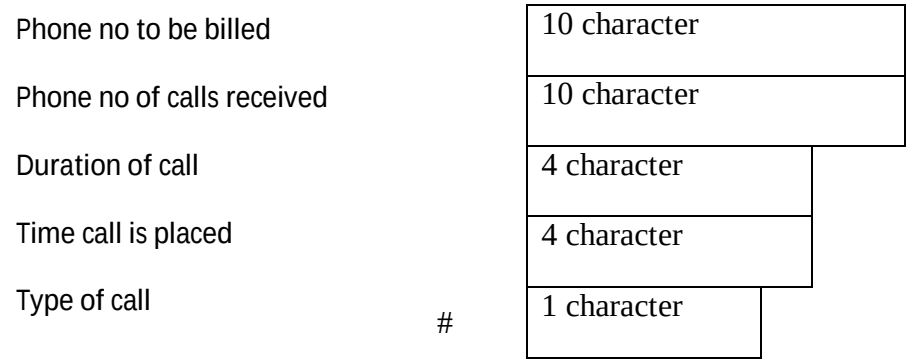
- Telephone number of the person to whom the call is to be billed
- Telephone number of the person receiving the call
- Duration of the call in minutes
- Time that call is placed
- Type of call e.g. person-to person or station –to station

These data need processing for generating bill information.

STRUCTURE OF THE DATA

DESCRIPTION –LONG-DISTANCE TELEPHONE CALL DATA		
FIELD NAMES	TYPE OF DATA	NUMBER OF CHARACTER
Phone no to be billed	Numeric	10
Phone no of call receiver	Numeric	10
Duration of call	Numeric	4
Time call is placed	Numeric	4
Type of call	Alphanumeric	1
TOTAL CHARACTER		29

In data base system, the structure will be like this:



Let assume that we have 5000 call for the month.
The problem will look as this:

Record	Phone to be billed	Phone no of calls received	Duration of call	Time call is places	Type of call
1	221	132	2 min	2 p.m	P
2	-	-	-	-	-
.					
.					
5000	-	-	-	-	-

Each record has five fields of 29 characters. With 5000 record the file requires 5000*29 characters on external storage device. To process these long –distance telephone call date in a bill, the record have to be identified through key record.

The record for this case is the telephone number of the person to whom the call is to be billed.

STRUCTURE OF CUSTOMER RECORD DATA

Field name	Types of data	No of character
Telephone no Name Address City Balance date	Numeric Alphanumeric ‘’ ‘’ Numeric Date	10 20 20 20 8 8
Total		66

Telephone	10
Name	20
Address	20
City	20
Balance	8
Date	8

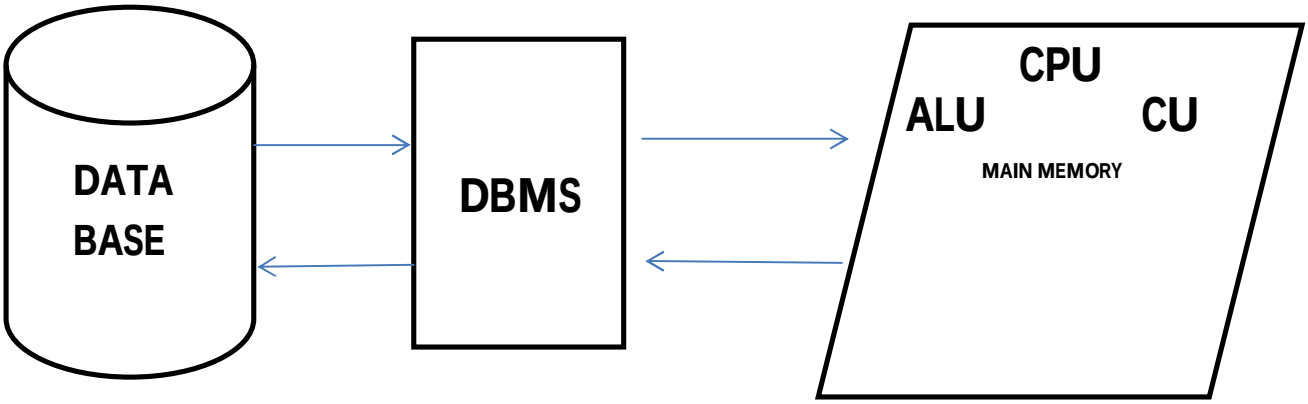
Record	Telephone no	Name	Address	City	Balance	Date
1 2	122 -	Ade	Ifo rd	Lagos	22.00	2/10/98

Each record is 66 characters. With 200 customers, we need to keep 66X200 character of date.

WEEK – TWO

DATABASE APPROACH

The database is closely associated with Data Base Management System (DBMS) software. A database management system (DBMS) is a series of computer programs used to create, store, maintain and access a database. The features offered by a particular DBMS depend on its type and level of sophistication. For example dBASE is a sophisticated DBMS for microcomputers.



Relationship of DBMS, database and application programs

As this figure indicates, an application program written in a high level language accesses the database through the DBMS software. In other words, DBMS software serves as the gate keeper for the database.

VC FILE
STUDENT FILE
DEAN FILE
STAFF FILE

MANUAL DATABASE

Here we have file management system on flat file environment

A flat file is a file or series of files that contain records and fields. These fields are called flat because they have no repeating groups.

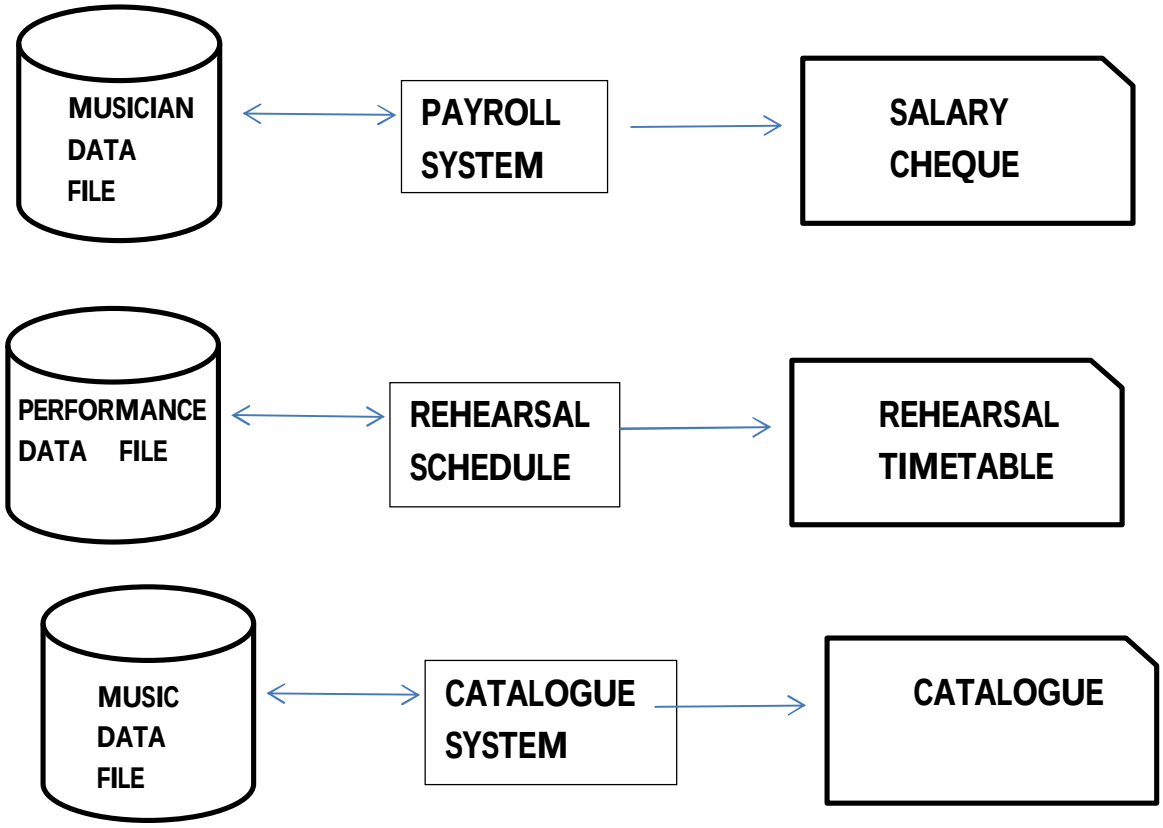
Advantage of Database

- Data and application programs are independent, so the same data can be used by several application programs.
- More information can be generated from the same amount of data. In other words, a given set of data can be manipulated in many ways.
- One-of-a kind requests can be fulfilled easily
- Data duplication is minimal. This is true because one occurrence of each data item is maintained
- Data management is enhanced and improved. This is possible because there is only one set of date for all users.
- More sophisticated security measures can be implemented.
- Data is readily shared between applications – this is eliminating duplication and the problems of maintaining consistency between duplicate values.
- New requests or one-of-a kind requests can be more easily implemented, because the logical interface with the DBMS is simpler than a set of physical interfaces.
- The applications programs are independent of the stored data. If the storage format changes, there is no need to alter the applications programs since they communicate with the DBMS in logical rather than physical terms.
- It can be argued that a single database management system for an integrated database allows for better management of data, since it is effectively in one place under the control of one set of people, namely those who implement the database.
- Finally, the integration or sharing of data between applications puts sophisticated programming within reach of all users of the database.

Disadvantage of Database

- Both DBMS software and extra hardware which might be needed to support the system can be expensive
- A DBMS is much more complex than a file processing system
- Organization puts all their data in a single basket – if anything happens to the basket that is the end of the data in the database.

FILE PROCESSING APPROACH



We have there files

E has three application programs

Paying the musicians

Scheduling rehearsals

Cataloguing music

Sharing data between applications

It is possible to share data among the applications

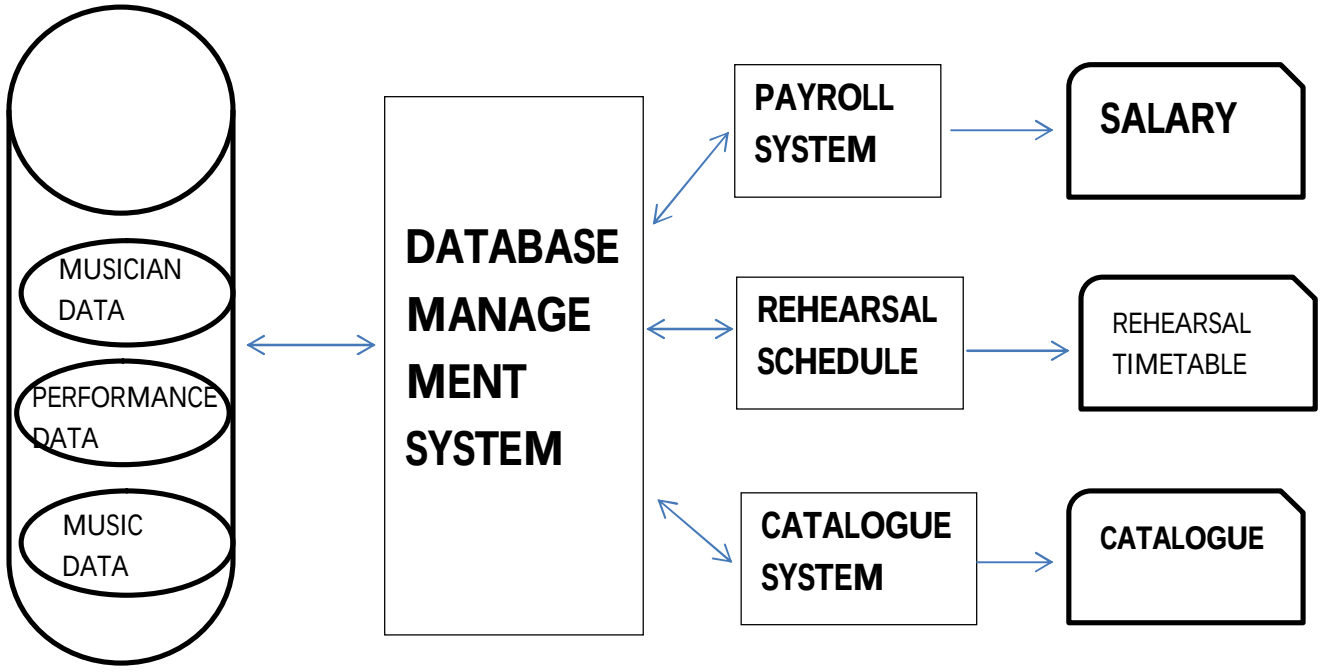
- If the numbers of files increase and applications are generated which use data from one, two or three files the number of interfaces increases rapidly, such situation might be termed as **interface explosion**
- It should be apparent that allowing different application to share data in a traditional file processing environment can cause considerable problems simply because of the number of interfaces required

In file processing approach,

Musician data file
Performance data file } are three separate physical files
Music data file

File	**Data**
Many interfaces	single interface
Share data is not possible	share data is possible
Physical structure	logical structure

DATABASE APPROACH



- One of the fundamental feature of the database approach is that it allows data to be shared between different applications
- In database approach, all the data are integrated into one physical file or a set of related files

- The sets of data are separated only logically within the database
- All access to the data is performed through a database management system (DBMS) a piece of software which understands and manipulates the logical data structures in the file.
- Since all the application programs interface only with the DBMS, they (application programs) require only a single interface to data in database.
- Interface of application programs to data in database can be at a logical rather than a physical level.
- It is not necessary for any application programs to know how a particular data item is stored as long as the DBMS can provide the data in the form required by the application. For example the data item is an integer, it matters nothing to the application if the value is actively stored in binary or character format as long as it is supplied to the application in the format it requires.
- The property of needing to know nothing about the physical storage of the data is termed as data independence.

Data Independence

According to ANSI/SPARC, American National Standards Committee on Computer and Information Processing Standard Planning and Requirements Committee

Every item of data which is used by any application must be present in database

The information which is stored should not change if some files are re-formatted.

A logical description of the data is in the database

University of Agriculture	Student Record – Name	Matric Number	GPA
	Faculty Record		
	Staff Record		

Server	Graphics files
	Database file
	Spreadsheet
	Word Processing Documents

Server Types

File Servers	File Transfer
Print Servers	File Storage and data migration
Application Servers	File update synchronization
Message Servers	File archiving
Database Servers	

File Transfer	users can transfer files between clients and servers
	Users can transfer files between multiple servers
File Security	Password
	Encryption

File Storage and Data Migration – online storage – consists of hard drive storage

Offline storage – removable hard disk

Nearline storage – tape is mouth to P/C

The process of moving data from ne line to offline or near-line storage is called data migration

File update ensuring that each user of a file has the latest version

File archiving the process of backing up files on offline storage devices such as tapes

DATA

- The analyst develops an input/output flow and designs data format
- The programmer/analyst writes a program to create a file. The same program is expanded or a separate program is written to allow the file to be maintained.
- The programmer/analyst writes a program to print the report originally requested.
- The data are collected and formatted while the programs are tested
- The file is created and stored on tape or disk
- The production of the report is accomplished.

With a DBMS – the procedure becomes

- The analyst develops an input/output flow and design data format
- The data are collected and formatted
- The file is created and the report generated using the DBMS

Data collection

- Source document – originate in form of clerically prepared documents – transferring data from secondary storage.
- Inputting data from machine readable source documents
- Direct entry (on-line) input via a keyboard
- Creating machine – readable media (off-line) for subsequent inputting to the computer

The process of capturing raw data for use within a DBMS involves getting the original data to the processing centre, transcribing it, converting it from the one medium to another and finally getting it into the computer.

- Getting the original data to the processing centre
- Transcribing it – data preparation
- Converting it from one medium to another
- Getting it into the computer.

Problems

- Source documents – a great deal of data still originate in the form of a clerically prepared document
- Data transmission
- Data preparation – this is the term given to the transcription of data from the source document to a machine – sensible medium
- Media conversion – data is prepared in a particular medium and converted to another medium for faster input to the computer.

Capturing Raw Data

Database Server	provide	security
		Database optimization
		Data distribution

- Knowing the real data to be put into database

- Organization of raw data in file structure
- Physical loading of raw data. Once the raw data is loaded, it must be maintained and kept up-to-date
- Duplicating raw data

The demand for data in database to be accurate is growing ever stronger

Classification of raw data into specific object type object types are described by listing their characteristics:

- Specify the domain of values for the smallest units of logical data e.g. integer or real
- Specify the units of measurement for logical data e.g. dollars, pound or feet
- Specify keys for certain logical units of data e.g. record types or relations
- Specify integrity constraints on the data e.g. an allowable range of values
- Specify access rules for the data e.g. allow update only if a correct password is supplied

ORGANIZATION OF DATA

The physical organization of the data in a database is described by physical storage structures such as volumes, files or bytes. Physical storage structures are defined by means of a storage definition language.

The storage definition language provides the ability to

- Select the storage medium and perhaps a specific device
- Describe a mapping from logical data to a physical representation e.g. record types map to files
- Specify indices for certain logical units of data e.g. data items or attributes
- Specify a physical ordering for the data
- Specify type conversion for data e.g. binary to decimal
- Specify the form of placement of a data selection e.g. buffers

Data security data editing

Data validation

Data protection or integrity

Data security

Logical data

DBMS

PHYSICAL DATA

The DBMS serves as the interface between the logical and physical data. Logical data entities employed by DBA and the user (the Requestor) are:

- Data items
- Logical records
- And files

Note that a logical file may contain records of more than one type. The physical data units are called

- Data element
- Store records
- Physical records
- And database

A physical record is defined as the data unit input from the hardware on a single access

WEEK – THREE

DATABASE MODELS

TYPES OF DATA MODELS (CALLED DATA STRUCTURES)

Computer based Information System (CBIS):

Design and implementation is done by Database Administrators (DBAs). The scope of responsibilities of DBA depends on the complexity of the database. In small organizations, one person may carry the entire responsibility of database design.

Responsibility of a Database Administrator (DBA)

1. Designing and implementing a database.
2. Establishing security measures.
3. Establishing recovery procedures.
4. Documenting the database.
5. Establishing database performance evaluations.
6. Adding new database functions.
7. Fine-tuning existing database functions.

Generating a database increases cost and creates more complexity in a CBIS operation. Implementation of an effective CBIS requires an online and comprehensive database regardless of its cost and complexity.

CBIS is designed to provide timely and relevant information by performing data analysis, modeling analysis or both.

Data analysis includes various query operations on a database.

Modeling analysis applies some types of model to the data available in the database and provides additional information that is not directly available within the data.

Types of file organization

1. Sequential: All records are stored and accessed one after the other. This method is similar to a cassette tape and is slow. If you want to access the seventh song on the tape, you must either listen to the first six songs or fast forward through them.
2. Random: Random organization enables you to access a record directly regardless of its storage location.
3. Indexed Sequential: In this organization, you can access a file either sequentially or randomly.

DATA MODEL

THE FLAT FILE MODEL:

This is a file or a series of files that contains records and fields. These files are called flat because there are no relationships between them. They have no repeating groups. The model does not allow sophisticated database operations.

Example:

NAME	MAJOR	AGE	GPA
May	MIS	25	3.00
Sue	CS	21	3.60
Debra	MGT	26	3.50
Bob	MKT	22	3.40
George	MIS	28	3.70

Basic data management operations such as: file creation, file deletion, file update, file single data, query can be performed using this model.

This model is limited in its capacity to support complex CBIS requirements.

THE RELATIONAL MODEL:

It is a popular model. It uses a mathematical construct called a relation (table). This table is a table of rows and columns of data.

Rows are records (tuples) and Columns are fields (attributes). Different relations can be linked on the basis of a common field (key).

Example:

Table 3.4

CUSTOMER NUMBER	NAME	ADDRESS
2000	Adams	2020, Broadway
3000	Baker	119, Jefferson
9000	Clark	7521, Madison

Table 3.5

INVOICE NUMBER	CUSTOMER NUMBER	AMOUNT	METHOD OF PAYMENT
111	2000	2000	Cash
222	3000	4000	Credit
333	3000	1500	Cash
444	9000	6400	Cash
555	9000	7000	Credit

Relation model is flexible; the Network model is not flexible.

Customer relation using Customer Number

Table 3.6

INVOICE NUMBER	CUSTOMER NUMBER	AMOUNT	METHOD OF PAYMENT	NAME	ADDRESS
111	2000	2000	Cash	Adams	2020, Broadway
222	3000	4000	Credit	Baker	119, Jefferson
333	3000	1500	Cash	Baker	119, Jefferson
444	9000	6400	Cash	Clark	7521, Madison
555	9000	7000	Credit	Clark	7521, Madison

To clarify this concept, look at the two relations in Table 3.4 and 3.5. As you can see the common field in these two relations is the customer number.

A relational DBMS can use these two relations to generate a report like in Table 3.6.

- The relational model is straightforward.
- Creation and maintenance of this type of database is easy
- Relational models offer a great degree of flexibility.

Operations handled by a Relational model include the following:

- Creation of relation.
- Updating (insertion, deletion and modification).
- Selection of a relation or a sub-relation.
- Join operations (putting two relations side-by-side).
- Projection (Selection of a subset of a field or a subset of a series of fields).
- General query operations.
- Cross operation

Shortcoming of Relational model:

- It cannot deal with complex database operations.
- Establishing many relations may use a great deal of disk space.
- Modification of many relations may be time consuming.

THE HIERARCHICAL MODEL:

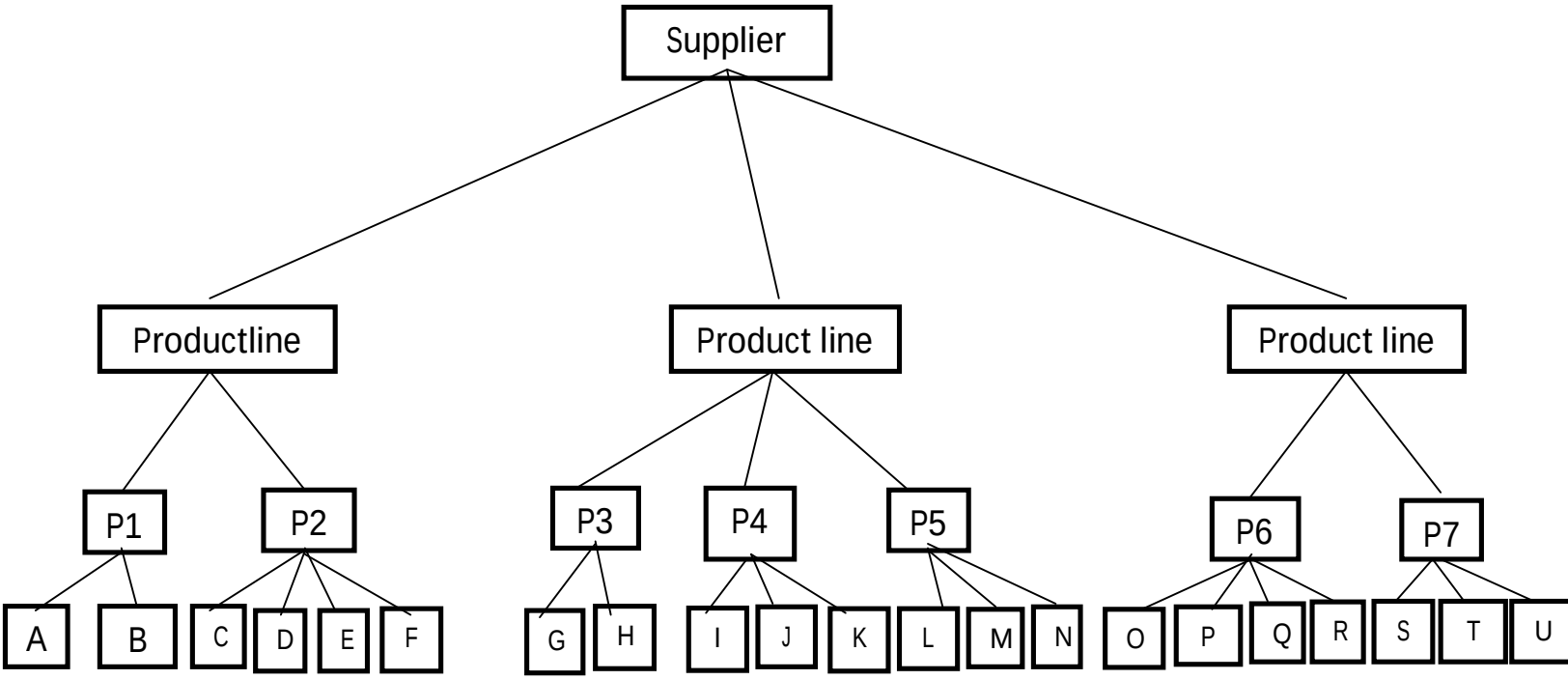


Fig. 3.6

Like the relational model, a hierarchical data model is made up of records called nodes. Each of these nodes can have several fields.

The presentation is similar to a one-dimensional array (a table with only one column or one row) or tree structure.

The relationships between the records are called branches. The node at the top of the hierarchy is called the root. Every node of the tree except the root node has a parent. The nodes with the same parent are called twins or siblings. For example, P1 and P2 in fig. 3.6 are twins or siblings.

The hierarchical model is sometimes called an upside-down tree (a tree with its root up). Fig. 3.6 illustrates an example of a hierarchical model. It indicates that a supplier may supply three

different families of products. In each family, there may be several different product categories. As an example, supplier X may supply soap, shampoo and toothpaste.

Within each product category, there may be many brands of the same product – for example, nine different shampoos or five different toothpastes. Such a relationship is called a one-to-many data structure. This means a parent can have many children. Each child has only one parent.

In the hierarchical model, a search in the parent node can lead you to children nodes and vice-versa. Any updating in a parent node should automatically update the children nodes.

The operations associated with the hierarchical model include file creation, file updating (insertion, deletion, addition, and modification), file queries, retrieval of the next descendant round, and retrieval of the parent record.

Data hierarchical model is flexible.

THE NETWORK MODEL

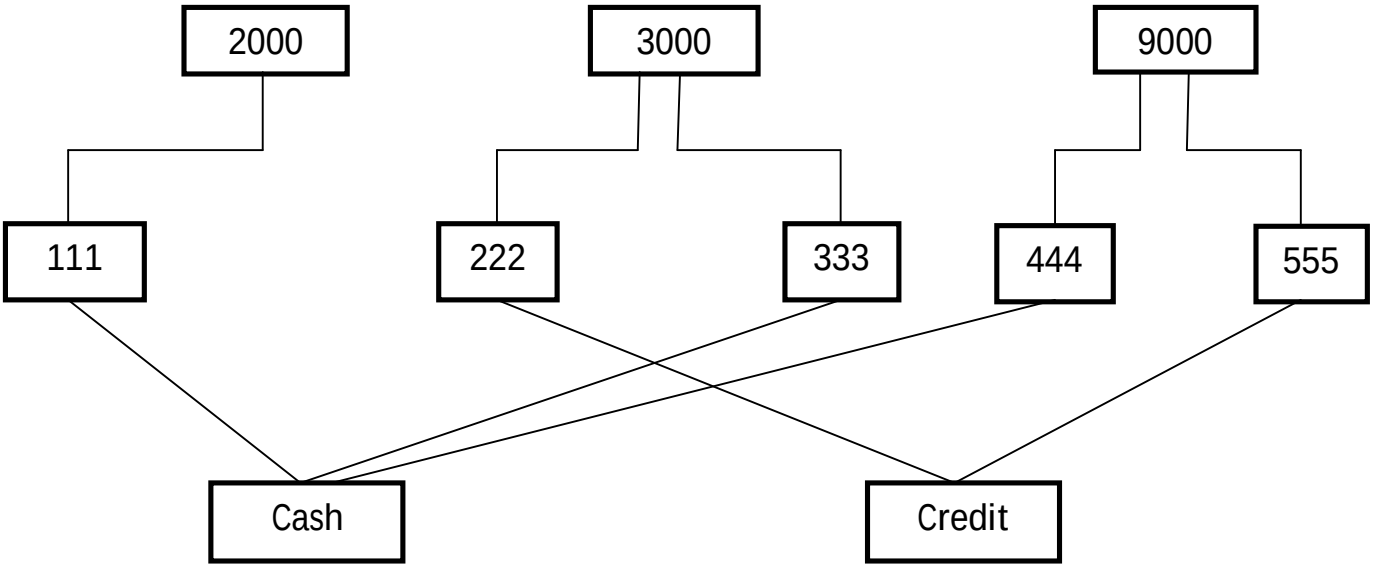


Figure 3.7

A complex network

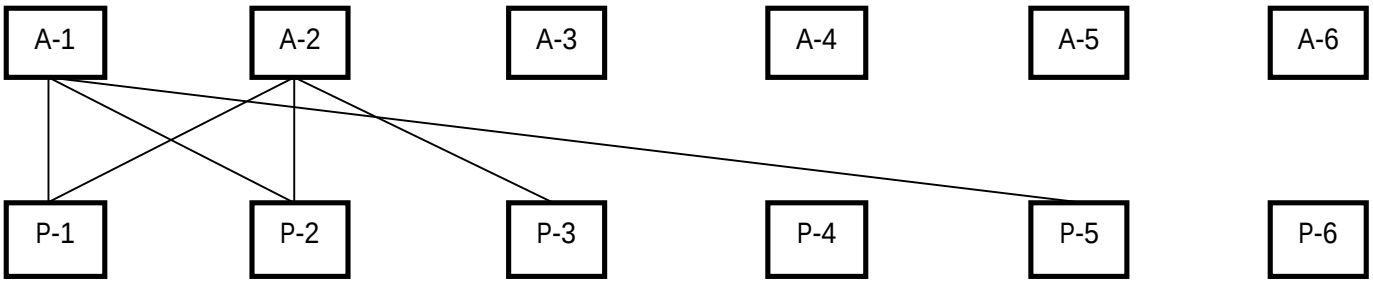


Fig 3.8

The network model is similar to the hierarchical model. The records and fields of a network are organized differently. Fig 3.7 illustrates customer and invoice relations in a network model. In place of related key fields, there is a connection between the invoice number, customer number and the method of payment. In this case, the customer number no longer needs to remain in the invoice record. As fig 3.7 illustrates, invoice numbers are connected to the customer number in the same order in which they were connected in table 3.4.

Operations associated with a network model include:

- File creation.
- File updating (insertion, deletion, addition and modification).
- File queries.

The network model can be considered an enhanced version of hierarchical model.

In these data structure, the relationship can be:

- One-to-many (simple network) and
- Many-to-many (complex network)

One-to-many: Each child (invoice) has two parents (methods of payment and customer number)

Fig 3.8 illustrates a many-to-many relationship. In a real estate agency, each agent is selling several properties. For example, agent A-1 sells properties P-1, P-2 and P-6, while property P-1 has been listed under agent A-1 and A-2. In a many-to-many relationship; the parent-child relationship breaks down because any record can be the parent and any record can be the child.

Operations any Data model in a CBIS environment

1. Basic data management operations: The basic data management operations include database creation, modification, deletion, addition, insertion, and maintenance. These operations are supported even in a flat file management system.

2. Basic arithmetic operations: These include simple arithmetic operations performed on different records and fields in a database including addition, subtraction, multiplication and division. These basic operations may be quite useful for simple query operations, such as calculating the average salary for both male and female employees or finding the maximum and minimum salary for each gender.
3. Projection operation: This function may be a special case of a general query operation that generates a subset of the fields. For example, in a student database that includes each student's name, GPA, age, gender, address and nationality. A projection operation could generate a listing of the names and GPA of all these students or a mailing list for mailing the students' transcripts.
4. Search (Query): This function may include different searches on a database for specific conditions. As an example, a triple criteria search on our example student database is as follows:

   DISPLAY ALL STUDENTS FOR GPA >=3 AND MAJOR="CS" AND AGE<=22

   Query operations can include as many criteria as the number of fields in the database. The search can include an AND search (all criteria specified must be met) an OR search (only one of the specified criteria must be met) and a NOT search (opposite criteria must be met or supply an alternative)

   AND, OR, NOT are referred to as Boolean operations.
5. Sort: Sort operations put the database in a specified order. Data can be sorted with one key or multiple keys in ascending order.
6. Summary: The summary operation may be a special case of basic arithmetic operations and basic query operations. For example, you could generate a sub-total of all MIS students and all accounting students in the student database.
7. Union (Merge) operation: The union (merge) operation enables a user to combine two files, tables or relations thereby generating a third file table or relation that includes all the information from the first two file tables or relations. In other words, the union operation does concatenation (joining) over the existing data.

Table 3.4

File 1

STUDENT	**MAJOR**
Bob	MIS
Barry	CS
James	MIS
Sue	Accounting

File 2

STUDENT	MAJOR
Mary	Marketing
Sherry	MIS
Suzy	Math

File 3

STUDENT	MAJOR
Bob	MIS
Barry	CS
James	MIS
Sue	Accounting
Mary	Marketing
Sherry	MIS
Suzy	Math

Table 3.7 presents the operation on a student database. File 3 is the union of files 1 and 2. Remember to perform the union operation; the two databases must be union compatible. This means they must include the same number of fields and data types.

8. Join operation: This operation combines two or more files, tables or relations within a database on a common field in order to generate a third file table or generation. Table 3.8 illustrates one example of this operation in which the common key is the customer name.

RELATION 1	RELATION 2	
Purchase Number	**Customer**	**Purchase amount**
112	Barry	2000
118	James	5000
129	Susan	1000
135	Bob	1500

RELATION 3

Joining of RELATION 1 and 2

CUSTOMER	PURCHASE NUMBER	PURCHASE AMOUNT
Barry	112	2000
James	118	5000
Susan	129	1000
Bob	135	1500

9. Intersection operation: The intersection generates the intersection of two relations in a third relation containing a common tuple(s) (common rows). The result of the intersection of relation 1 and 2 is relation 3which contains only one row (tuple), the one belonging to the first two relations.

An intersection operation

MAJOR	GPA	STUDENT	MAJOR	GPA
CS	3.60	Tom	ACC	2.90
MIS	3.80	Jerry	CIS	3.70
ACC	2.90	Bob	MGT	3.90

Relation 3

Intersection of Relations 1 and 2

STUDENT	MAJOR	GPA
Tom	ACC	2.90

Union=Set_1+Set_2

Intersection=Set_1*Set_2

Difference=Set_1-Set_2

Difference operation is defined as the set of elements that are in Set A but not in Set B. For example,

[1, 3, 4]- [1, 2, and 4] = [3]

DATA MODEL

Within a model, there is object type. Example of object type is house. House has characteristics such as address (street number and street name), color (red, green etc.), style (bungalow, duplex), price etc.

A set of characteristics that uniquely identifies an object of a house can be used to identify each house within its object type is referred to as a **key**. For example, if the address of a house can be used to each house, then the address characteristic is a key of the object type houses.

Characteristics (attribute)

For each entity set (house), its attributes have certain values. For example, the color attribute has values such as red, green and blue. The set of possible values of an attribute is called the **domain** of the attribute. A set of attribute that uniquely determines an instance of an entity is called a **key**.

A data model is a pattern according to which data are logically organized.

Relationships:

House- Two values of houses; address and color.

Address	**Color**
10, Ibadan road	Red
21, Akin Street	Blue
11, Abole way	Green

House 10, Ibadan road is painted Red.

House 21, Akin street is painted Blue.

The two sets of values immediately carry some information. This information is available because a relationship has been established between the values of address and color.

A relationship is a correspondence between the numbers of two sets.

Equality	[1, 3] = [1, 3]
Inequality	[1, 3] <> [2, 4]
Subset	[1, 3] <= [1, 2, 3, 4]
	Set A is a subset of Set B if

	every element of Set A is also an element of Set B
Proper subset	[1, 3] < [1, 2, 3, 4] Set A is a proper subset of Set B if A is a subset of B and there is at least one element in B that is not in A
Superset	Set A is a superset of Set B if every element of B is also an element of A
Proper superset	Set A is a proper superset of Set B if A is a superset of B and there is at least one element of A that is not in B.

WEEK – FOUR

ADVANCED DATABASE MODEL

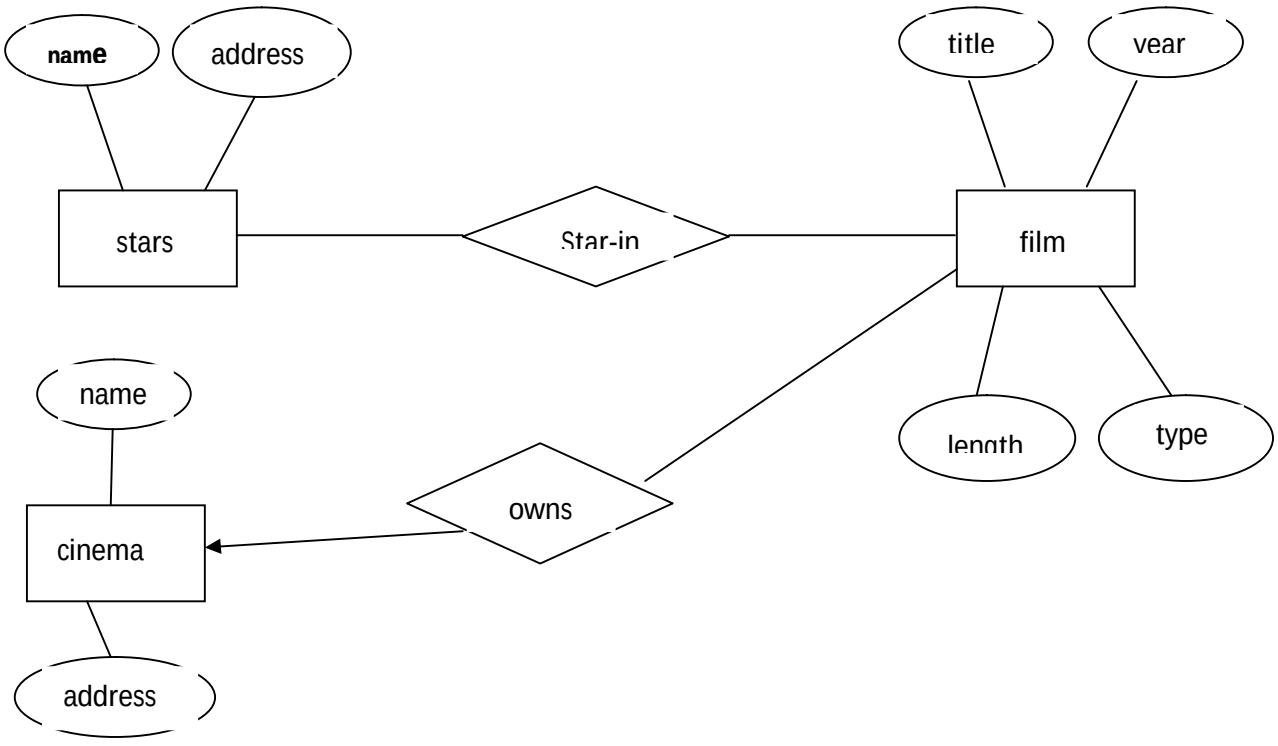
stars (name, address)

film (title, year, length, genre)

cinemas (name, address)

Movies

Title	Year	Length	Genre
Gone with the wind	1939	231	Drama
Star wars	1977	124	Sci-fi



Entity relationship diagram

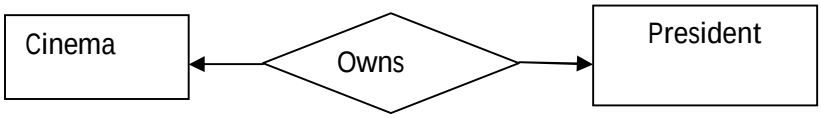
Star_in: is a relationship connecting each film to the stars of that film.

Owns: each film is owned by at most one cinema.

Relationship = 1 : 1

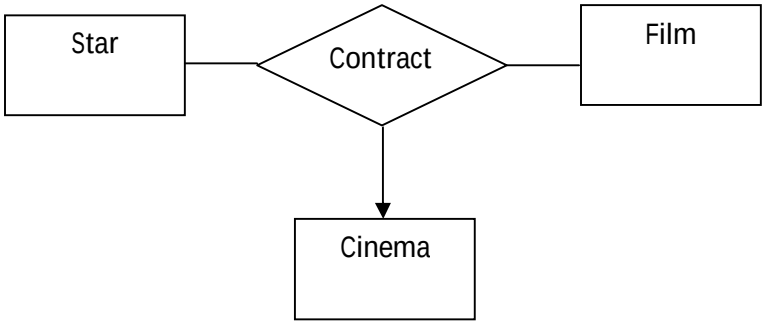
m:1 1:m

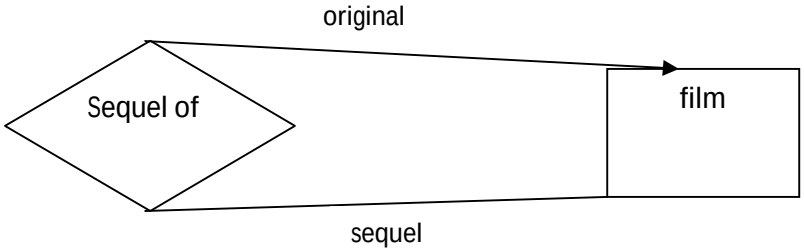
n:m



for a particular star and film, there is only one cinema with which the star has contracted for the film.

A cinema may contract with several stars for a film and a star may contract with one cinema for more than one film.





Rules in relationship

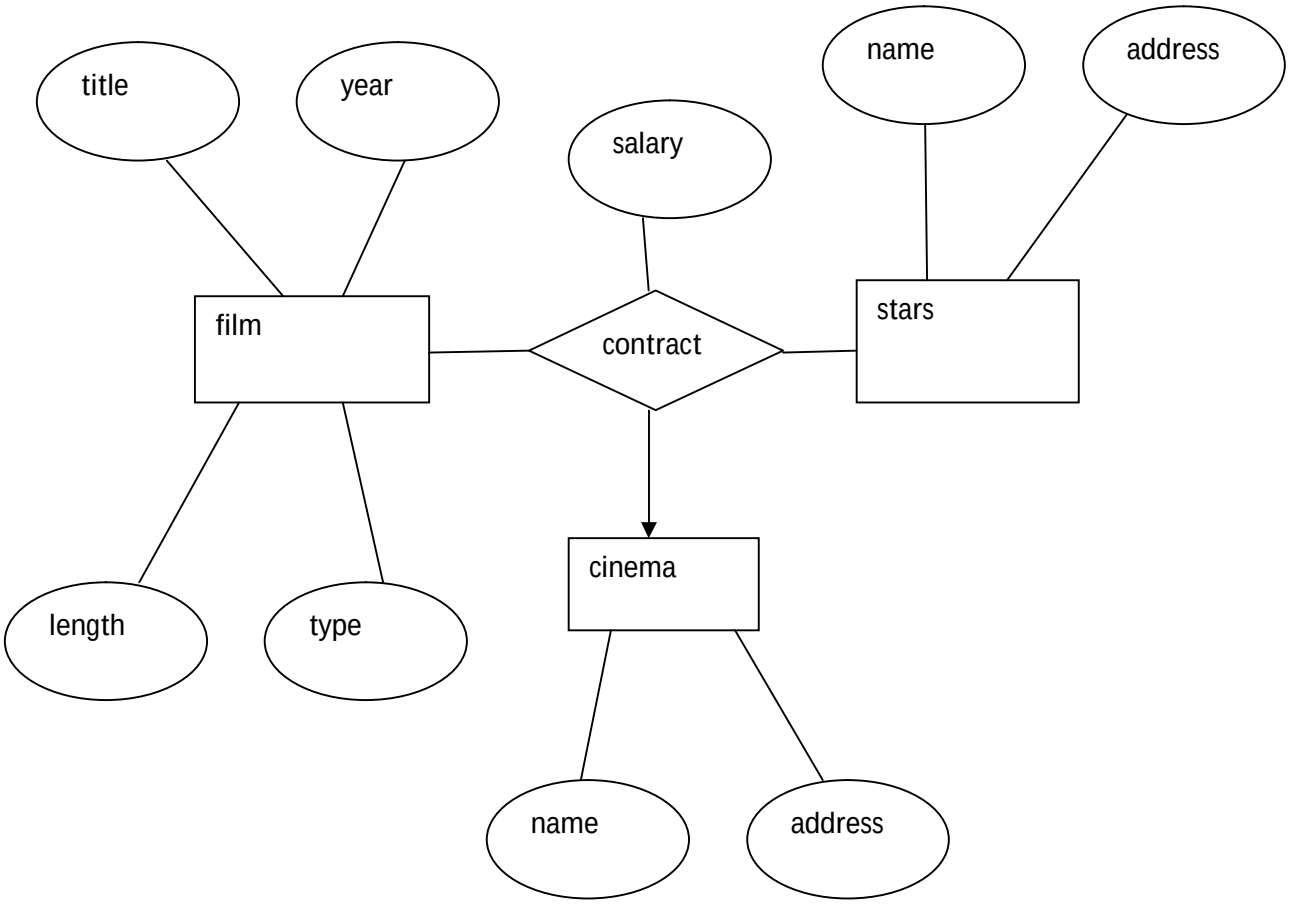
Original = role

Sequel= role

- Film may have many sequels.
- Each sequel, there is only one original film.

  Many – one

  Sequel to original



A relationship with an attribute

Stars might get different salaries for different films. They may pay different salaries to different stars. Different stars in a film may receive different salaries.

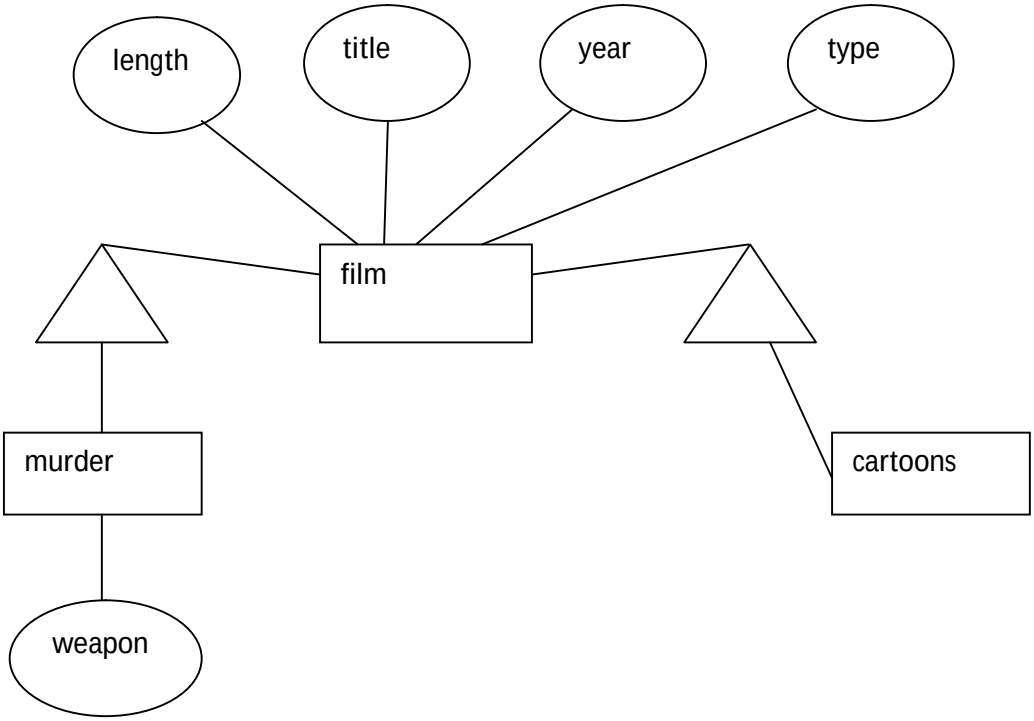
We connect an entity set to its subclasses using a relationship is a (an A is a B)

Notation is triangle

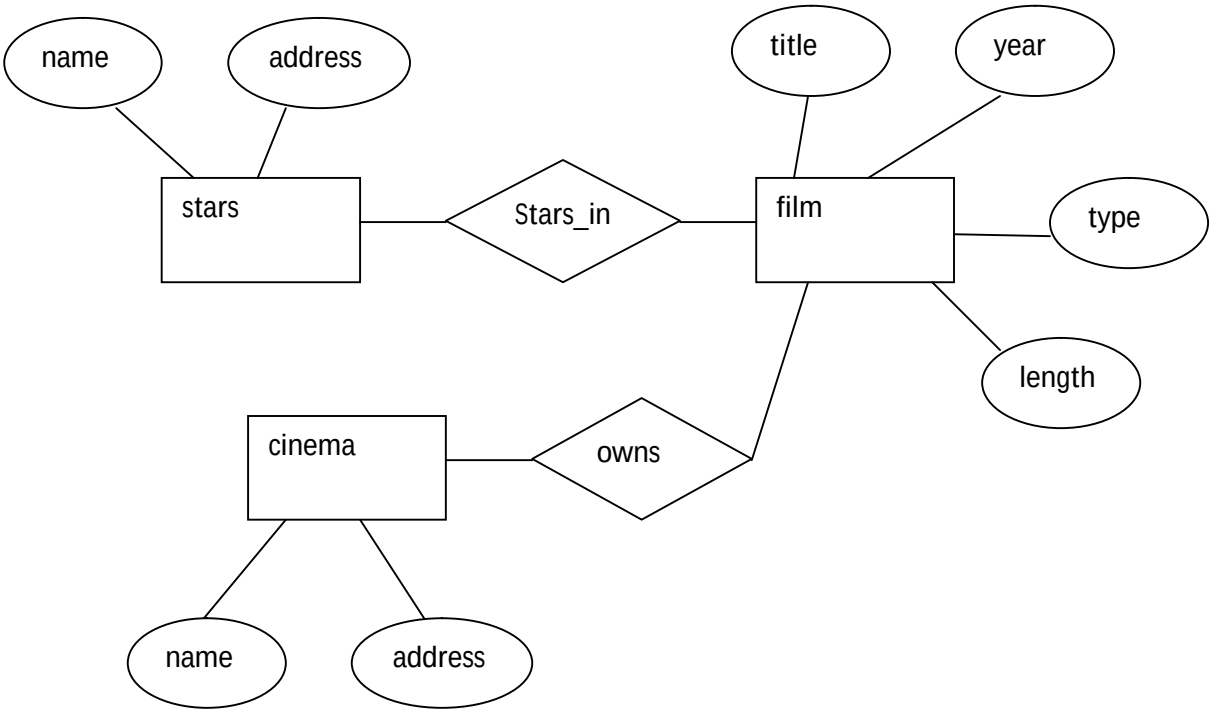
One side of triangle is attached to the subclass and the opposite point is connected to the super class.

Every is a relationship is one-one

is a relates a subclass to its super class.

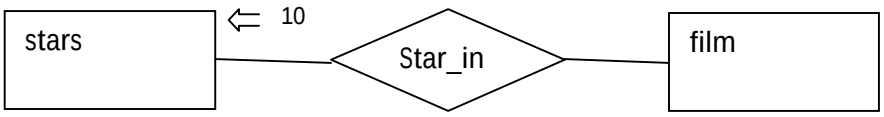


Representing keys in the E/R model



- keys are indicated by underlines = we underline only primary key when we have different types of keys.

Degree constraints



- a film entity cannot be connected by relationship star-in to more than 10 star entities.

Weak Entity Sets

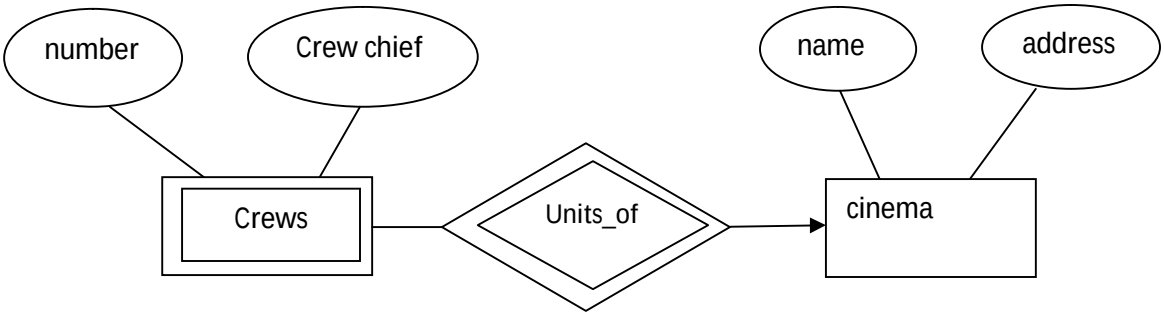
- It is possible for an entity set's key to be composed of attributes, some or all of which belong to another entity set. Such an entity set is called a **weak entity set.**

Causes of weak entity sets

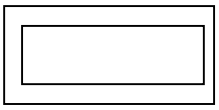
- Entity sets fall into a hierarchy based on classifications unrelated to the "isa hierarchy"
  If entities of set E are sub units of entities in set F, then it is possible that the names of E-entities are not unique until we take into account the name of the F-entity to which the E entity is subordinate.

e.g.

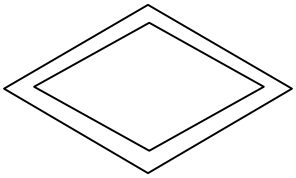
A film cinema might have several film crews.



The crew might be designed by a given cinema as crew1, crew2, and so on.



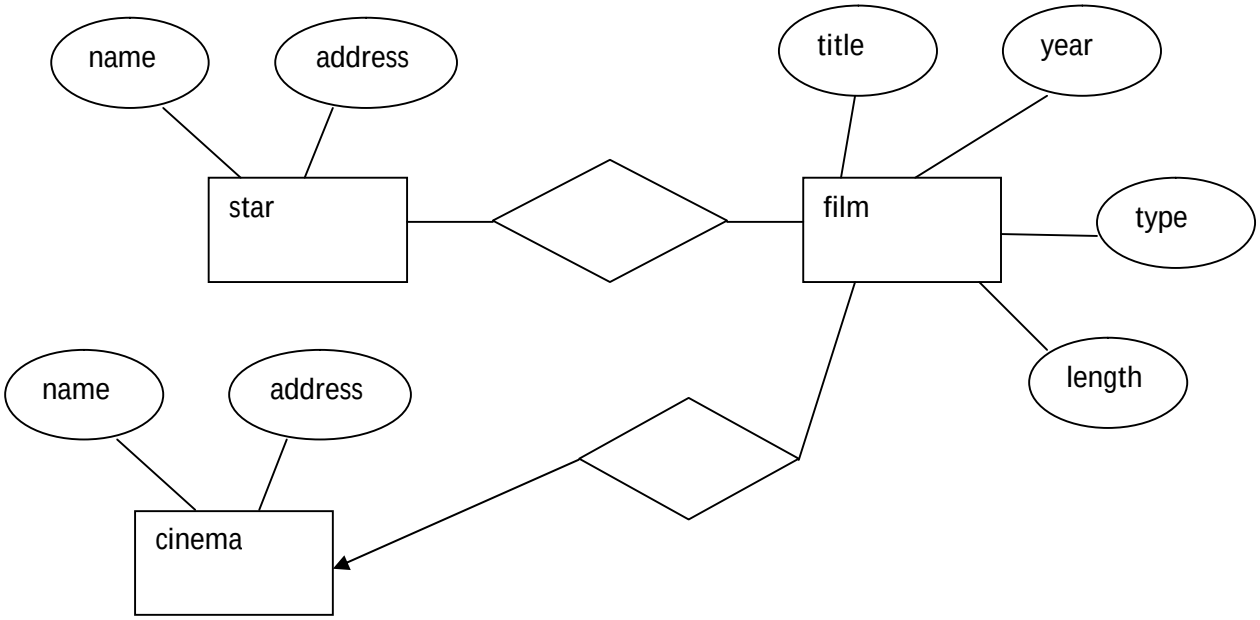
Double triangle indicate a weak entity



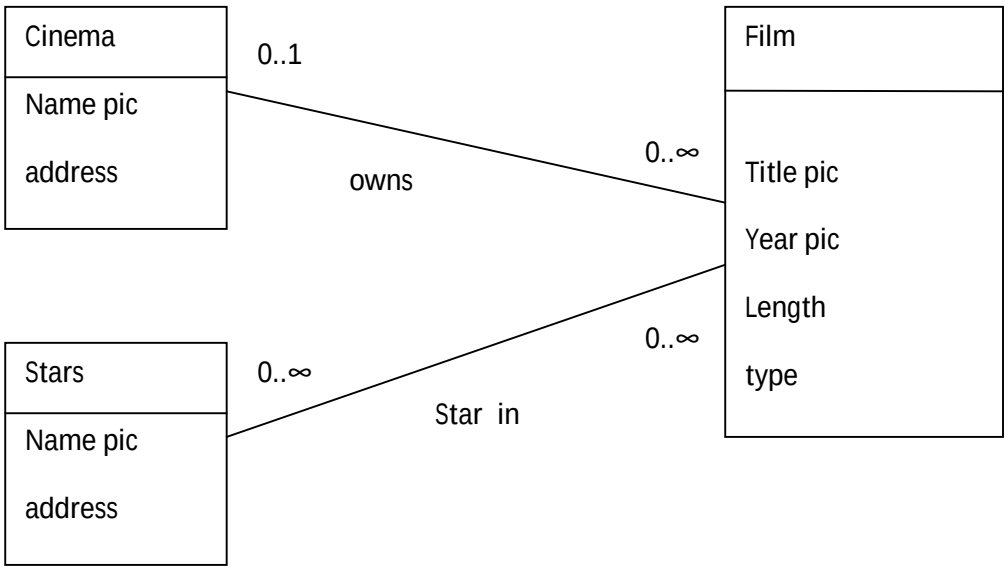
Indicate double diamond = a many-one relationships that helps provide key for the weak entity set

Requirements for weak entity set

- Zero or more of its own attributes
- There many-one relationships are called supporting relationships for E, end the entity sets reached from E are supporting entity sets.



An association is a set of pairs of objects, one from each of the classes it connects.



Two associations = owns, star_in

Every association has constraint on the number of objects from each of its classes.

Constraint as m....n

M.∞ stands for infinity = there is no upper limit

O.∞ no constraint at all on the number of objects.

1.1 = exactly one

* subclasses = UML permit four sub classes

* an aggregation is a line between two classes that ends in an open diamond at one end

0....1 = aggregation is a many-one association

UNIFIED MODELLING LANGUAGE (UML)

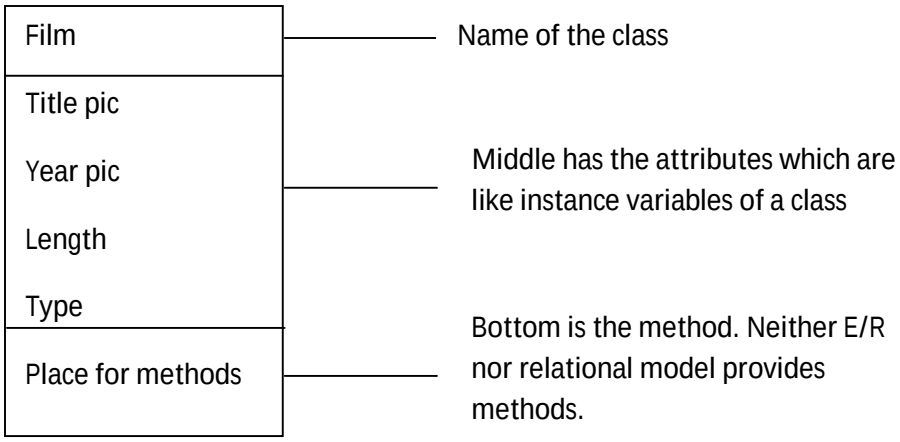
UML differs from E/R model with the exception of multi-way relationships.

UML offers the ability to treat entity set as true classes with methods as well as data.

E/R Model	**UML**
Entity set	Class
Binary relationship	Associations
Attributes on a relationship	Association class
isa hierarchy	subclass
Many-one relationship	Aggregation
Many-one relationship with referential integrity	Composition

- A class in UML is similar to an entity set in the E/R model.

It is divided into three parts



A binary relationship between classes is called an association.

From UML diagram to Relations

Film (title, year, length, type)

Stars (name, address)

Cinema (name, address)

Star_in(filmTitle, filmYear, starName)

Own(filmTitle, filmYear, cinemaName)

Declaration of keys

```
Class film (key (title, year)) {
        Attribute string title;
        Attribute integer year;
        Attribute integer length;
        Attribute enum FilmType
                {drama, horror, comedy}
                Film_Type
        };
```

Relationships in ODL

- Relationship is declared inside a class declaration by the keyword = relationship
  = a type
  = name of the relationship

Relationship set <star> stars

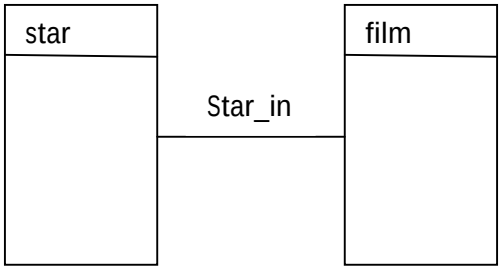
Inverse star:: starred in;

Inverse Relationship

To access the stars of a given film we might like to know the film in which a given star acted.

Relationship set<film> starred in

Inverse film::stars



Object Definition Language ODL

Like UML, the class is the central concept in ODL.

A declaration of a class in ODL in its simplest form is

Class <name> {

<list of properties>

-attribute

-relationship

-method

ODL has structural type

APPLICATIONS THAT USES XML

- Cell phones
- File converter PDF to XML converter
- Voice XML

XML code

```
<?XML version = “1.0” encoding = “ISO-8859-15”?>
        <class_list>
                <student>
                        <name>Robert</name>
                        <grade>A+</grade>
                </student>
                        <name>Leonard</name>
                        <grade>-A</grade>
                </student>
        </class_list>
```

XML declaration, version of XML, type of encoding you are using.

XML element = <student>

<?XML version =”1.0” is an XML document

Encoding = “utf-8” Unicode transformation format is a common choice of encoding for characters in documents because it is compatible with ASCII and uses only one byte for the ASCII characters.

Standalone = “yes”?> indicates that there is no document type definition for this document.

XPath

In XPath, there are seven kinds of nodes:

- Element
- Attribute
- Text
- Name space

- Processing_instruction
- Comment
- Document

XML documents are treated as tress of nodes. The topmost element of the tee is called the root element.

XML document.

```
<?XML version = “1.0” encoding = “ISO-8859-1”?>
<bookstore>
    <book>
        <title lang = “en”>Harry Potter </title>
        <author>J.K. Rowling </author>
        <year>2005</year>
        <price>29.99 </price>
        </book>
</bookstore>
```

Example of nodes in the XML document above

<bookstore> (root element node)

<author> J.K. Rowling </author> (element node)

Lang = “en” (attribute node)

Atomic values are nodes with no children or parent.

e.g. J.K. Rowling

“en”

```
<?xml version = “1.0” encoding = “ISO-8859-1”?>
<bookstore>
    <book>
        <title lang = “eng”> Harry Potter </title>
        <price>29.99 </price>
    </book>
```

```
    <book>
        <title lang = “eng”>learning XML</title>
        <price>39.99</price>
    </book>
</bookstore>
```

The most useful path expressions are listed below:

Expression	**Description**
Nodename	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the documents from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

See listed path expression and the result of the expressions

Path expression	**Result**
Bookstore	Selects all the child nodes of the bookstore element
/bookstore	Selects the root elements bookstore

NOTE: if the path starts with a slash (/) it always represents an absolute path to the element.

```
<?xml version =  “1.0” encoding = “ISO-8859-1”?>
    <bookstore>
        <book category = “cooking”>
            <title lang = “en”> Everyday Italian </title>
            <author> Giada De Laurent </author>
            <year> 2005 </year>
            <price>30.00</price>
        </book>
        <book category = “children”>
```

```
        <title lang = “en”> Harry Potter </title>
        <author> J.K. Rowling </author>
        <year> 2005 </year>
        <price>29.99</price>
    </book>

    <book category = “WEB”>
        <title lang = “en”>Query kick start </title>
        <author> James Mchovern </author>
        <year> 2003 </year>
        <price>39.95</price>
    </book>
</bookstore>
```

1.
    Everyday Italian
    Harry Potter
    Learning XML
2. Everyday Italian
3. 30.00, 29.99, 39.95
4. 39.95
5. Query Kick Start

Bookstore/book/title/ //price = selects all the title elements of the book element of the bookstore element AND all the price element in the document.

C prog. lang != not equal

Query one = 1. Selects all the titles

/bookstore/book/title

2. Selects the title of the first book

/bookstore/book[1]/title

3. Selects all the prices

/bookstore/book/price/text()

4. Select price nodes with price >35

/bookstore/book[price>35]/price

5. Select title nodes with price > 35

/bookstore/book [price > 35]/title

Each step is evaluated against thue nodes in the current node-set.

A step consists of

- An axis (defines the tree_relationship between the selected nodes and the current node)
- A node_test (identifies a node within an axis)
- Zero or more predicates

The syntax for a location step is:

axisname: nodetext [predicate]

Example:

Example	**Result**
child:book	Selects all book nodes that are children of the current node
attribute:lang	Selects the lang attribute of the current node
child:*	Selects all children of the current node
attribute:*	Selects all attributes of the current node
child:*/child:price	Selects all price grandchildren of the current node
bookstore/book[price > 35.00]/title	Selects all the elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

Selecting unknown nodes

XPath wildcards can be used to select unknown XML elements.

Wildcards	Description
*	Matches any element node
@*	Matches any attribute node
Node()	Matches any node of any kind

Path expression	Result
/bookstore/*	Selects all the child nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have any attributes

Selecting several paths

By using the | operator in an Xpath expression, you can select several paths.

Path expression	Result
///book|title|//book|price	Selects all the title and price elements of all book elements
bookstore|book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element no matter where they are under the bookstore element.
//@lang	Selects all attributes that are named lang.

Predicates

Predicates are used to find a specific node or a node that controls a specific value. Predicates are always embedded in square brackets.

Path expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element

/bookstore/book[last() – 1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position() <3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang=”eng”	Selects all the title elements that have an attribute named lang with a value of “eng”
//bookstore/book[price > 35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00

XSLT, XQuery

XPath Axis

An axis defines a node-set relative to the current node

Axis Name	**Result**
ancestor =	Selects all ancestors (parent, grandparent, etc) of the current node
Attribute =	Selects all attributes of the current node
Child =	Selects all children of the current node
Descendant =	Selects all descendants (children, grandchildren, etc) of current node
Following =	Selects everything in the document after the closing tag of the current node
Parent =	Selects the parent of the current node
Self =	Selects the current node

Location Path Expression

A location path can be absolute or relative.

- An absolute location path starts with a slash (/) and a relative location path does not
- An absolute location path

  /step/step/...
- A relative location path

  step/step/...

WEEK – FIVE

DATABASE NORMALIZATION

There are many relationships

-A relationship may be a 1:1, 1: N, n: m

1:1 relationship, for example, is the relationship between an employee's personnel number and social insurance number.

- Each employee has only one personnel number
- Each employee has only social insurance number

1: N is the relationship between an employee's personnel number and salary

- An employee has one personnel number
- An employee has or may have different salaries

N: m relationship is the relationship between house color and house price

- Houses with certain colour may sell at various prices
- Houses at the same price may have various colours

NORMALIZATION

Suppose that the value of the attribute BUILDER determines values for the attribute STYLE and PRICE and that the value for the attribute STYLE determines the values for the PRICE.

Grouping these attributes together in the relation HOMES1 (BUILDER, STYLE, PRICE) has several undesirable properties.

First, the relationship between style and price is repeated in the relationship for each builder who builds a particular style of home. This repetition creates difficulties. If a builder who happens to be the last builder of a certain style, home is deleted from the relationship, then the relationship between the style and its price also disappears from the relation. This is called a deletion anomaly.

- Similarly, if a new builder who happens to be the first builder of a certain style home is added, then the relationship between a style of home and its price will also be added, even though this was not the purpose of the insertion. This is called an insertion anomaly. Such insertions and deletions are anomalous because most such operations will not produce these side effects on the style-price relationship. These anomalies are undesirable since the user is not likely to realize the consequences of the insertion or deletion.
- A second problem with the grouping is the effect of updates of the consistency of the relation. Suppose that the relationship between a style and its price is changed e.g. the

price is increased. To maintain the consistency of the relation, the new style-price relationship should be included for every builder of the style.

- If the relationship HOMES1 (BUILDER, STYLE, PRICE) is normalized, then the consistency and anomaly problems disappear.

Normalization is a step-by-step reversible process of replacing a given collection of relations by successive collections in which the relations have a progressively simpler and more regular structure. The reversibility guarantees that the original collection of relations can be recovered and therefore no information has been lost.

The objectives of normalization are

1. To make it feasible to represent any relation in the database.
2. To obtain powerful retrieval algorithms based on a simpler collection relational operations that would otherwise be necessary.
3. To free relations from undesirable insertion update and deletion dependencies.
4. To reduce the need for restructuring the relations as new types of data are introduced.
5. To make the collection of relations neutral to the query statistics which these statistics are liable to change as time goes.

- The first two objectives apply only to the first step (conversion to first normal form).
- The last three objectives apply to all normalization steps.

THE NORMALIZATION PROCESS

- Unnormalized form:

  Eliminate attributes that have relations as elements

- 1NF

  Eliminate partial dependence of non-prime attributes on keys

- 2NF

  Eliminate transitive dependence of non-prime attributes on key

- 3NF

  Eliminate redundancy in keys

- BCNF

1NF: FIRST NORMAL FORM

i. Relates to the structure of the relation.
ii. It requires that every attribute of a relation be based on a simple domain i.e. a domain consisting of single, simple values.

A relation is in 1NF if every attribute in the relation is based on a simple domain.

Consider the relation;

HOMES (BUILDER, MODEL)——1NF

HOMES1 (BUILDER, STYLE, PRICE)——2NF

HOMES2 (BUILDER, STYLE) COST (STYLE, PRICE)—3NF

HOUSE3 (BUILDER, SUBDIV, PRICE, STYLE)

BUILDER, SUBDIV→STYLE

BUILDER, SUBDIV→PRICE

PRICE→SUBDIV

2NF: SECOND NORMAL FORM

If MODEL is the relation MODEL (STYLE, PRICE) and a builder builds several model homes, then the HOMES relation violates 1NF, the relation can be represented as the 1NF relation HOMES1 (BUILDER, STYLE, PRICE).

Any relation can be put into 2NF by replacing a non-simple domain by its constituent simple domains. The problems in choosing relations from MODEL are strongly tied to the fact that the values of some attributes completely determine the values of other attributes in a relation. This fact will be formalized as the concept of functional dependency.

HOUSES (STYLE, BUILDER)

HOUSES (ID, ADDRESS, LOT, SUBDIV, STYLE, BUILDER)

HOUSES1 (ID, ADDRESS, LOT, SUBDIV, STYLE)

CONTRACTOR (SUBDIV, BUILDER)

- Let A and B be attribute of a relation
- Let DOMAIN (A) be the domain of A

  Let DOMAIN (B) be the domain of B

  Let f be a time-varying function

  f: DOMAIN (A) ⟶DOMAIN (B)

In the mathematical sense, f is not a function because it is allowed to change over time in the same sense that database relations are allowed to change over time.

We can say f: A ⟶B

→f is a functional dependency

→ B is said to be dependent (functional dependent) on A

→ A is said to be determine (functional determine) on B

A $\not\rightarrow$ B = means that there is no functional dependency between A and B

$\longleftrightarrow$ If both A→B and B→A hold then at all times A and B are 1:1 correspondence and the notation A←→B is used.

Let f: $A_1\ A_2\ A_3 \ldots A_n \rightarrow B$

g: $A_1\ A_2\ A_3 \ldots A_m \rightarrow B$ where $m<n$

Assume $f(a_1\ a_2\ a_{3\ldots}a_n) = g(a_1\ a_2\ a_3 \ldots a_m)$ for all a_i in A_i, $1 \leq I \leq n$

That is the attributes A_{m+1}, A_{m+2}, $A_{m+3} \ldots A_n$ are extremely in f.

In this case, B is said to be partially dependent on $A_1\ A_2\ A_3 \ldots A_n$.

If there is no g with the above property, then B is fully dependent on $A_1\ A_2\ A_3 \ldots A_n$.

- Partial dependencies can cause insertion/deletion anomalies and consistency problems.

The second normalization (2NF) removes partial dependencies of non-prime attributes on keys.

Second Normal Form (2NF): A relation R is in 2NF if R is in 1NF, and each non-prime attribute in R is fully dependent upon every key.

Given the relation (keys are underlined)

HOUSES (<u>ID</u>, <u>ADDRESS</u>, <u>SUBDIV</u>, STYLE, BUILDER).

In the HOUSES relation, the keys are ID, ADDRESS and SUBDIV.

The non-prime attributes are STYLE and BUILDER. The relation is therefore not in 2NF. To place the HOUSES relation into 2NF, it is split into two relations:

HOUSES1 (<u>ID</u>, <u>ADDRESS</u>, <u>SUBDIV</u>, STYLE)

CONTRACTOR (SUBDIV, BUILDER)

- Access to the builder information is still possible from the HOUSES1 relation through the SUBDIV attribute common to both relations.

For example, to determine the builder of a certain house, the SUBDIV attribute value is obtained from the appropriate tuple in the HOUSES1relation.This value is then used to search the CONTRACTOR relation and determine the corresponding builder.

Notice that it is possible to keep information about a builder for a sub-division independent of the HOUSE1 relation. This capability eliminates one of the insertion anomalies discussed earlier.

The next normalization step converts relations to Third Normal Form or 3NF by eliminating transitive dependence of non-prime attributes on keys.

Suppose that A, B and C are three subsets of a relation R. Suppose that the following time-independent conditions hold:

$A \longrightarrow B$

$B \not\longrightarrow A$

$B \longrightarrow C$

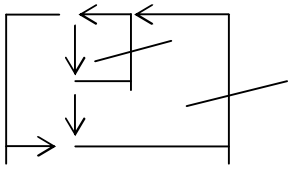
$A \longrightarrow C$

$C \not\longrightarrow A$

A

B

C



Transitive dependence of C on A

NOTE: $C \longrightarrow B$ is neither prohibited nor required

If the above conditions hold, then C is transitively dependent on A under R

Price is transitively dependent on Builder under HOME1.

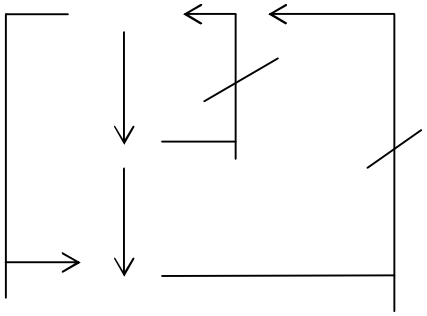
In the special case where $C \rightarrow B$ also holds, both B and C are transitively dependent on A under R.

Transitive dependencies also lead to the insertion/deletion anomalies and consistency problems.

Builder

Style

Price



Transitive dependence

Consider now the problem of changing the style of house that is built by a builder. In this case, the value of the price attribute also has to be changed. If it is not, then the database will show an inconsistency.

Consider also the problem of inserting and deleting tuples. If a new HOMES1 tuple is inserted for a new style home, then the relationship between style and price is also created.

Similarly, if a builder is deleted from the HOMES1 relation, and if this is the last or only builder of a particular style, then all information about the particular style-price relationship is also deleted.

The transitive dependency of BUILDER on PRICE can be eliminated by splitting the HOMES1 relation into the two relations.

HOMES2 (BUILDER, STYLE), COST (STYLE, PRICE)

HOMES1 (BUILDER	STYLE	PRICE)
CADILLAC	DUPLEX	65000
DELZOTO	DUPLEX	65000
HOWLETT	BUNGALLOW	45000
JOINT	RANCH	50000
METRO	BUNGALOW	45000
MONZA	DUPLEX	65000
TEREX	RANCH	50000
WIMREY	RANCH	50000

- These two relations cannot contain any transitive dependencies since they are each only of order two.
- The price now appears only once for each style.
- No information has been lost since the price of a style of house can be obtained by using the style values from the HOMES2 relation to access the COST relation.

Both the HOMES2 and COST relations are in 3NF

HOMES2 (BUILDER	STYLE)
CADILLAC	DUPLEX
DELZOTO	DUPLEX
HOWLETT	BUNGALOW

JOINT	RANCH
METRO	BUNGALOW
MONZA	DUPLEX
TEREX	RANCH
WIMREY	RANCH

COST (STYLE PRICE)

BUNGALOW	45000
DUPLEX	65000
RANCH	50000

WEEK – SIX

NORMALIZATION / DECOMPOSITION

RELATIONS IN THIRD NORMAL FORM

Third Normal Form (3NF) – A relation is in 3NF if R is in 2NFand no non-prime attribute of R is transitively dependent on any key of R.

Any relation in 3NF has the property that every non-prime attribute of the relation is neither partially dependent nor transitively dependent on any key. This means that the non-prime attributes are independent of each other.

Third Normal Form or 3NF does not exclude prime attributes from exhibiting partial and transitive dependencies. It has been found that partial and transitive dependencies among prime attributes can also lead to consistency and update problems.

To eliminate these problems, the statement of 3NF has been reformulated to avoid reference to the concepts of prime attribute, full dependency and transitive dependency.

BOYCE CODD NORMAL FORM (BCNF)

A relation R is in BCNF if it is in 1NFand for every set of attributes C of R, if any attribute not in C is functionally dependent on C, then each and every attribute in R is functionally dependent on C.

If a relation is in BCNF, it immediately follows that it is in 3NF. There are examples of relations in 3NF, but not in BCNF (Bernstein, 1975).

For example, consider a modified set of functional dependencies describing part of the HOUSES2 model shown below.

HOUSES2 (BUILDER, SUBDIV, PRICE, STYLE)

Suppose that contracts are awarded to builders in various subdivisions. Within any subdivision, a builder's contract specifies that only one style of house be built by the builder. Suppose further that for each subdivision in which a builder has a contract, the builder charges a fixed price per house. Also, the price charged determines the subdivision. That is, a builder charges different prices for the same style house for each subdivision in which the builder has a contract.

The attribute SUBDIV is now transively dependent on the key BUILDER. This transitive dependency does not create certain problems.

Suppose the price a builder charged in an old subdivision is also to be charged in a new subdivision i.e. the PRICE→SUBDIV dependency is modified.

In a BCNF relation R, every functional dependency in R must be of the form K→A where K is a key and A is any attribute. The following can be asserted:

1. All non-prime attributes must be fully dependent on each key.
2. All prime attributes must be fully dependent on all keys of which they are not a part.
3. No attribute (prime or not) can be fully dependent on any set of attributes that is not a key.

Factors – To link relationship of network data mobile, object types, characteristics, and relationships.

Value of the attribute BUILDER determines values for the attribute STYLE and PRICE.

Value of the attribute STYLE determines the value for PRICE.

Unlike mathematical relations, database relations are time-varying since tuples may be inserted, deleted or updated.

An index implies a selection mechanism or a pointer structure to desired data.

DESIGN OF RELATIONAL DATABASE SCHEME

Conversion from ODL or E/R to relational database cause redundancy (mean where a fact is repeated in more than one tuple)

Decomposition = breaking a relation schema (set of attributes) into two smaller schema

INF = a relation is in INF if every attribute in the relation is based on a simple domain

= relational model requires that each component of each tuple be atomic

Atomic means = data type should be integer or storing not struct or record that can reasonably have its value broken into smaller components

House (Builder model) House (Builder, style, price)

2NF = the second normalization removes partial dependencies of non prime attribute on key.
Houses (ID, Address, Sub-div, Style, Builder)
- The keys are ID , Address, Sub-div
- the non prime attributes are style and builder
House 1 (ID, Address, Sub-div, Style)
Constructor (sub-div, Builder)

3NF = eliminating transitive dependences of non prime attributes on keys.
A relation R is in 3NF if and only for every nontrivial FD X→ A

1. X is a super key r
2. A is prime = member of at least one key.

Suppose that A, B, C are those subsets of attributes of a relation R

$A \rightarrow B$ $B \nrightarrow A$

$B \rightarrow C$ $C \nrightarrow A$

$A \rightarrow C$

C is transitively dependent on A under R; price is transitively dependent on builder under home 1.

Transitive dependences lead to the insertion and deletion anomalies and consistency problems.

The transitive dependency of builder on price can be eliminated by splitting the Home 1relation into the two relations.

Home 1 (Builder,	style,	Price)
Cad	duplex	65,000
Del	duplex	65,000
How	bungalow	45,000
Joint	story building	50,000
Metro	bungalow	45,000
Moon	duplex	65,000
Tear	story building	50,000
Wimp	story building	50,000

Home 2 (Builder,	style)	cost (style,	price)
Cad	duplex	duplex	65,000
Del	duplex	bungalow	45,000
How	bungalow	story building	50,000
Joint	story building		
Metro	bungalow		
Moon	duplex		
Tear	story building		
Wimp	story building		

Boyce coddy norm form

- eliminate redundancy

- eliminate anomalies

Kind of anomalies

Redundancy = occurrence of tuples

Insertion anomalies

Deletion anomalies

Update anomalies

Decomposition Relations

Given a relation R with schema {A1, A2, …., An} we may decompose R into two relations S and T with schema {B1,B2,…, Bm } and {C1, C2,…..,Ck}

R (A1,A2,…..,An) = S (B1, B2,…..Bm) ∪ T (C1, C2,……,Ck)

S ∪ *T* are projection form

R = {title, year, length, filmtype, studio name, star name}

S = {title, year, length, filmtype, studio name}

T = {title, year, Star name}

S = movie 1

Title	year	Length	Film type	Studio name
Star wars	1997	124	Colour	Fox
Might	1991	104	Colour	Disney
Ways	1992	95	Colour	Paramount

T = movie 2

Title	Year	Star name
Star wars	1997	Carrie
Star wars	1997	Mark
Star wars	1997	Ford
Might	1991	Ester
Ways	1992	Dana
Ways	1992	Mike

There is redundancy for film type

T – Movie 2 = title and year appear several times

Despite decomposition, anomaly will not occur in BCNF

BCNF = A relation is in BCNF if and only if whenever there is a nontrivial dependency A1 A2….. An → B for R, it is the case that {A1, A2,….,An} is a super key for R

That is the left side of every nontrivial functional dependency must be a super key.

Thus, an equivalent statement of the BCNF condition is that the left side of every nontrivial functional dependency must contain a key

Decomposition into BCNF

Title	year	Film type	Studio name	Length	Studio Address
Star	1997	Color	Fox	124	Hollywood
Might	1991	Color	Disney	104	Vick
Way	1992	Color	Paramount	95	Hollywood
Add	2001	Color	Paramount	102	Hollywood

$R \rightarrow S \cup T$

Title	year	Film type	Studio name	Length
Star	1997	Color	Fox	124
Might	1991	Color	Disney	104
Way	1992	Color	Paramount	95
Add	2001	color	Paramount	102

Studio name	Studio address
Fox	Hollywood
Disney	Vick
Paramount	Hollywood

R is not in BCNF

S is in BCNF

T is in BCNF

3^{RD} Normal form

It is relaxation of BCNF requirement

A relation R is in 3^{rd} NF if:

Whenever A1, A2,…..,An → B is a non trivial dependency either

1. {A1, A2, ……, An} is a super key or
2. B is a member of some key

Note that the different between this 3NF and BCNF condition is the clause or B is a member of some key.

3^{rd} NF allows the right hand side attribute as a member of the key.

When these relations are not in BCNF, there will be some redundancy left in the schema.

Projecting functional dependencies

When we decompose a relation schema, we need to check that the resulting schemas are in BCNF.

Suppose we have a relation R, which is decomposed into relation S and some other relation

Let F be the set of functional dependencies known to hold for R.

To compute the functional dependencies that holds in S d the following:

R (S1, S2, S3)

Consider each set of attribute X that is contained in the set of attributes of S

Compute X^+

Then for each attribute B such that

1. B is an attribute of S
2. B is in X^+
3. B is not in X

The functional dependency X→ B holds in S

E.g. R (A, B, C, D)

A → B

B → C are given for R

Let S (A, C) be one of the relations on a decomposition of R

We shall complete the dependencies that hold in S

We must compute the closure of each subset of [A, C] which is a set of attributes of S

A^+ = A → B AB

B → C ABC

A^+ = {A, B, C} A → B B is not in S [A, C]

A → C C is in S [A, C]

We do not claim that A → B is a dependency for S

C is the schema for S

We assert dependency A → C for S

C^+ = A → B

B → C

= C

AC^+ = ABC for A → B

B → C = ABC

AC → B B is not in S [A, C] there is no new dependency

R (A B C D E)

S (A, B, C)

A → D

B → E

DE → C

A^+ =

A → D = AD A → D

B → E = AD

DE → C = AD

B^+ =

A → D =

B → E = BE B → E

DE → C =

C^+

A → D =

B →E =

DE → C = C

AB^+ the only dependency we need for S is AB → C

A → D = ABD AB → D

B → E = ABDE AB → E

DE → C = ABCDE AB → C

AC^+ = C is in the schema of S so we get dependency for S

A → D = ACD AC→ D

B → E =

DE → C =

BC^+

A → D =

B → E = BCE BC → E

DE → C =

ABC^+

A → D = ABCD ABC → D

B → E = ABCDE ABC → E

DE → C = ABCDE

Find the implied FD's

Suppose we have a relation ABCD with some FD's F. If we decide to decompose ABCD into ABC and AD, what are the FD's for ABC and AD

F = AB → C

C→ D

D → A

{A, B, C}

A^+ = C → A

B^+ = AB → C

C^+ = CD C → D BC → A

CDA` C → A

AB^+ = ABC AB → C

ABCD AB → D

AC^+ = ACD AC → D

BC^+ = BCD BC → D

ABCD BC → A

{A, D}

A^+ =

$D^+ = AD = D \rightarrow A$

In ABC with FD's

$A \rightarrow B$

$B \rightarrow C$

Project at AC

1. $A^+ = ABC$ yields $A \rightarrow B$, $A \rightarrow C$
2. $B^+ = BC$ yields $B \rightarrow C$
3. $C^+ = C$
4. $AB^+ = ABC$ yields $AB \rightarrow C$
5. $AC^+ = ABC$ yields $AC \rightarrow B$
6. $BC^+ = BC$
7. $ABC = ABC$

Resulting FD's

$A \rightarrow B$

$A \rightarrow C$

$B \rightarrow C$

Consider the resolution R(A, B, C, D, E, F)

FD's are $AC \rightarrow B$

$BD \rightarrow F$

$F \rightarrow CE$

Find all the key of R

Find non trivial dependence key

Computing the closure of attribute

1. Let X be a set of attribute that eventually will become the closure
   -first we initialize X to be {A1, A2, ….., An}
2. We repeatedly search for some functional dependency B1, B2, ….., $Bn \rightarrow C$

Such that all B1, B2, …..,Bn are in the set of attribute X but cc is not .We add C to the set X.

3. Repeat 2 as many times as necessary until no more attributes can be added to X.

4. The set X after no more attributes can be added to it is the correct value of $\{A1, A2,\ldots, An\}^+$

Algorithm for decomposition

Input; relation schema R and set of FD's for R

1. Complete keys for R based on FD
2. Repeat until no more BCNF violations
   a. Pass any R' with AA → BB that violates BCNF
   b. Decompose R' unto R1 (AA, BB) and R2 (AA, CC) where CC is all attribute in R' except (AA, BB)
   c. Compute FD for R1 and R2
   d. Compute key for R1 and R2 based on FD

WEEK – SIX

ALGORITHM DERIVING CANDIDATE KEYS FROM FDS

Input: a set S of FDS that contain only subsets of a header H

Output: the set C of super keys that hold as candidate keys in all relation universes over H in which all FDS in S hold

```
begin
    C := ∅   // found candidate keys
    Q :={H} // super keys that contain candidate keys
while Q <>∅ do
let K be some element from Q;Type equation here.
Q :=  Q – {K};
minimal := true;
for each X→Y in S do
   K := (K−Y) ∪X ;
    if  K′ in K then
minimal  := false ;
Q :=  Q ∪ { K′ } ;
end if.
end for
if minimal and there is not a subset of K in C then remove all super keys of K from C ;
C := C ∪{ K }
    end if
```

end while

end.

Lossless join property:

- Can recover any instance of the decomposed relation from corresponding instance of the smaller relations.
- It is required to be reversible.
- It wants no information loss in the process.

The decomposition is lossless since it does not lose any information contained in the original relation.

- It does not generate any spurious tuples which leads to false or misleading information
- It is called non – additive join since it does not add any new tuples
- It allows to get back exactly what we started with before decomposing.

Remember we decompose to do away with redundancy and all of the problems associated with the duplication data.

Supplier	Parts	Project
Bello	2	bks
John	2	ruler

Supplier	Part	Part	projects
Bello	2	2	bks
John	2	2	ruler

Project	supplier
Bks	Bello
Ruler	John

Supplier	part	project
Bello	2	Books
Bello	2	Rulers
John	2	Books
John	2	Rulers

Supplier	Part	projects
Bello	2	bks
John	2	rulers

The question is: What conditions must be satisfied in other to guarantee that joining R_1 and R_2 back together takes back to original R?

A decomposition D = {R_1, R_2} of R has the lossless join property with respect to the set of FD's f if

- FD($R_1 \cap R_2$) → ($R_1 - R_2$) is in f^+ or
- FD($R_1 \cap R_2$) → ($R_2 - R_1$) isin f^+

MULTIVALUE DEPENDENCIES

4NF

- A relation R is in 4NF if for every nontrivial MVD X─>─>Y in R, X is a super key.
- That is all FD's and MVD's follow from key ─>other attributes(i.e. no MVD's and no FD's besides key functional dependencies).
- 4NF is stronger than BCNF.
- Because every FD is also a MVD.

Multivalued Dependencies

A multivalued dependency (MVD) has the form X─>─>Y where X and Y are sets of attributes in a relation R.

X─>─>Ymeans whenever two rows in R agree on all the attributes of X then we can swap their Y components and get new rows that are also in R.

X	Y	Z	
a	b_1	c_1	
a	b_2	c_2	
a	b_1		$c_2$must be in R too
a	b_2	c_1	
-	-	-	
-	-	-	

MVD student(s_1d, c_1d, club)

s_id —>—> c_id

Given: Relation R and set of MVD's for R

Definition: R is in 4NF with respect to its MVD's if every nontrivial MVD AA—>—>BB, AA contains a key.

Note: Since every FD is also in MVD, 4NF implies BCNF

Trivial M VD

A trivial MVD is when the RHS is a subset of its LHS. AA—>—>BB where BB is a subset of AA or (AA U BB) does not contain all the attributes of R.

Nontrivial MVD

AA—>—>BB where BB is not a subset of AA and (AA U BB) does not contain all attributes of R.

MVD

Definition:

AA —>—>BB is an MVD for relation R if:

For all tuples t, u in R,

If t [AA] = u [AA] then there exists a V in R such that:

1. v[AA] = t[AA]
2. v[BB] = t[BB]
3. v[CC] = u[CC]
   where CC is all attributes in R except (AA U BB).

MVD is when there are 2 independent 1 : N relationships, A : B and A : C are mixed in the same relation. You represent an MVD by X—>—>Y

1. Y is a subset of X
2. X U Y = R both attributes together equals the relation.

Complete MVD + FD rules

- FD is reflexivity

  augumentation

  transitivity

- MVD is complementation
  If X─>─>Y then
  X ─>─> attributes (R) - X – Y

- MVD is augumentation
  If X─>─>Y and
  V ⊆ W then
  XW─>─>YV

- MVD is transitivity
  If X─>─>Y and
  Y─>─>Z then
  X─>─>Z – Y

- Replication (FD is MVD)
  If X─>─>Y then
  X─>─>Y
- Coalescence
  If X─>─>Y and
  Z⊆Y and there is W is disjoint
  from Y such that
  w─>z then
  x─>z.

Algorithm for Decomposing a relation into 4NF relation (same ideas as BCNF)

Input: Relation schema R and set of FD's and MVD's for R

-1- Compute keys for R based on FD's

-2- Repeat until no more 4NF violations

-2a- Pick any R′ with AA─>─>BB that violates 4NF

-2b- Decompose R′ into R_1(AA, BB) and

R_2(AA,CC) where is all attributes in R′ except(AA U BB)

-2c- Compute FD's and MVD's for R_1 and R_2

-2d- Compute keys for R_1 and R_2 based on FD's

Algorithm for decomposing a relation into BCNF

Input: Relation schema R and set of FD's for R

-1- Compute keys for R based on FD's

-2- Repeat until no more BCNF violations;

-2a- pick any R′ with AA —> BB that violates BCNF

-2b- decompose R′ into R_1(AA,BB) and

R_2 (AA,CC) where

CC is all attributes in R′ except (AA U BB).

-2c- compute FD's for R_1 and R_2

-2d- compute keys for R_1 and R_2 based on FD's

5NF

L has no join dependency

If a relation is already in 3NF and each of its keys consists of a single attribute, it is also in 5NF.

5NF is also called project-join normal form.

L it is the highest normal form.

L there may be some relations that are in 4NF, but still have some redundant information. However, there are no violating MVD's and/or FD's so we cannot use either of these dependencies to decompose the relation.

- Consider each of the following proposed rules regarding dependencies
- For each, if it is false give a counter example

If it is true, give a brief argument.

- Why it is true e.g. by applying the closure to desire FD's.
- You may assume the relation to which they apply is R(A, B, C, D)

a) If A─>─>B then A─>B

If A─>B then A─>─>B false. A─>B

A─>─>B

BC─> D

b) If A─>B and BC─>D then AC─>D true

Applying closure to AC^+ = ABCD

ABCD = contains D

The transitive rule also works.

Note that you can't reduce ABC─> BD to AC─>D by receiving B from both sides.

c) If A─>B and

B─>─>C then

A─>─>BD

Applying the transitive rule to A─>B and B─>C yields A─>C

Applying the promotion rule to A─>C the yields A─>─>C

Applying complementation rule to A─>─>C then yields A─>─>BD

Algorithm Testing for the Lossless Join Property

1) Create a matrix S with one row i for each relation R_i in the decomposition D, and one column j, for each attribute A_j in R.
2) Set S(i, j) i=b_{ij} for all matrix entries

(* each b_{ij} is a distinct symbol associated with indices (i, j) *) .

3) For each row i representing relation schema R_i for each column j representing attribute A_j.

If R_i includes attributes A_j then set S (i, j) i=a_j

(* each a_j is a distinct symbol associated with index j *)

4) Repeat the following until a loop execution results in no changes to S.

For each functional dependency X─>Y in f, for all rows in S which have the same symbol in the columns corresponding to attributes in X

Make the symbols in each column that correspond to an attribute inY be the same in all these rows as follows:

If any of the rows has an "a" symbol for the column, set the other rows to that same "a" symbol in the column. If no "a" symbol exist for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows for the attribute and set the other rows to that "b" symbol in the column.

5) If a row is made up entirely of "a" symbol then the decomposition has the lossless join property – otherwise, it does not.

Decomposition and Lossless (non additive) Joins

A decomposition D= (R_1, R_2,...,R_m) of R has the lossless (non additive) join property with respect to set of dependencies F on R if for every relation instance r of R that satisfies f,

the following holds (* is the natural join operation)

*($\prod$<R_1>(r), …,$\prod$<R_m>(r)) = r

Lossless join refers to loss of information not loss of tuples.

Additional tuples represent erroneous information and hence add more information.

Properties of Lossless Join Decomposition.

1) A decomposition D = {R_1, R_2} of R has the lossless join property with respect to a set of functional dependencies f on R if and only if either
   - The FD (($R_1 \cap R_2$) ─> (R_1 - R_2)) is in f^+or
   - The FD (($R_1 \cap R_2$) ─> (R_2 – R_1)) is in f^+
2) If a decomposition D = {R_1,R_2, …, R_m} of R has the lossless join property with respect to a set of functional dependencies f on R and if a decomposition D_1 = {Q1, Q_2,…, Q_k} of R_1 has the lossless join property with respect to the projection of f on R_1, then the decomposition

$D_2 = \{R_1, R_2,\ldots, R_{i-1}, Q_1, Q_2,\ldots,Q_k, R_{i+1},\ldots, R_m\}$ of R has the lossless join property with respect to f.

Decomposition to have Lossless join Operation and Preserve Dependencies

1) Find a minimal cover G for F

   F* is the set of functional dependencies specified in R*)

2) For each left- hand side X that appears in G

   Create a relation schema $\{X \cup A_1 \cup A_2 \cup A_m\}$ where $X \rightarrow A_1, X \rightarrow A_2, X \rightarrow A_3,\ldots, X \rightarrow A_m$, are all dependencies in G with X as left hand side.

3) Place all the remaining (unplaced) attributes in a single relation schema

4) If none of the relation schemas contains a key of R, create one or more relation schema that contains a key of R, create one more relation schema that contains attributes that form a key for R

   R= {SSN, EName, Pnumber, PName , Plocation, Hours }

   R1== {EName, Plocation}

   R2={SSN, PNumber, Hours, PName, Plocation }

   FD $\longrightarrow$ SSN EName,

   PNumber $\longrightarrow$ PName, PLocation.

   SSN, PNumber $\longrightarrow$ Hours

	SSN	EName	PNumber	PName	Plocation	`Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a6

Employee(ENo eName Mgr Dept Salary)

R = {SSN, EName, Pnumber, Pname, Plocation, Hours}

R_1 = {SSN, EName}

R_2 = {PNumber, PName, PLocation}

R_3 = {SSN, PNumber, Hours}

FD = SSN $\longrightarrow$ EName

PNumber $\longrightarrow$ PName, PLocation

SSN, Pnumber $\longrightarrow$ Hours

	SSN	EName	PNumber	PName	PLocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4a_5	b_{26}	
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

Original matrix S at start of algorithm

	SSN	EName	PNumber	PName	PLocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	a_2	a_3	a_4	a_5a_6	

Matrix S after applying the first two functional dependencies = last row is all “a” symbols

So we stop.

WEEK - SEVEN

MULTIVALUED DEPENDENCIES

- 4^{th} = fourth normal form
- Join dependencies
- 5^{th} = fifth normal form
- Inclusion dependencies
- Template dependencies
- Domain key norm form

Multivalue dependencies and Fourth Normal form

Multivalued dependencies are a consequence of first normal form which disallowed an attribute in a tuple from having a set of values or a list of values or a combination of both.

- Whenever two independent 1: N relationship A : B and A:C are mixed in the same relation by representing all possible combination an MVD may arise.

Definition:

A multivalued dependency (MVD) $X \rightarrow\!\!\!\!\rightarrow Y$ specify on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation instance r of R: if two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties.

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$
- $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$
- $t_3[R-X-Y] = t_2[R-X-Y]$ and
  $t_4[R-X-Y] = t_1[R-X-Y]$

whenever $X \rightarrow\!\!\!\!\rightarrow Y$ holds, we say X multi determines Y.

Note that because of the symmetry in the definition whenever $X \rightarrow\!\!\!\!\rightarrow Y$ holds in R, so does

$X \rightarrow\!\!\!\!\rightarrow (R-X-Y)$

Note also that (R-X-Y) is the same as R – (X U Y) = Z

Hence X—>—>Y implies X—>—>Z and therefore it is sometimes written as X—>—>Y/Z

EName—>—> DName

EName—>—> DName or

EName—>—> DName/ DName

EMP

EName	DName	DName
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Emp-Projects

EName	DName
Smith	X
Smith	Y

emp-Dependents

EName	DName
Smith	John
Smith	Anna

Smith works on project with PName X, Y. Has two dependent with DName John and Anna

- If we stored only the first two tuples in Emp
  <Smith X, John>
  <Smith Y, Anna>
  We would incorrectly show association between project X and John and between project Y and Anna

  We must store the other two tuples
  <Smith X, Anna>
  <Smith Y, John>
  to show that {X, Y} and {John, Anna} are associated only with Smith that is, there is no association between PName and DName in Emp.

An MVD X—>—> Y in R is called a trivial MVD if

a) Y is a subset of X or
b) X U Y = R

Enp-Project has the trivial MVD EName—>—> PName

An MVD that satisfies neither a) or b) is called a nontrivial MVD

Emp = is a nontrivial MVD

the values of X, Y of PName are repeated with each value of DName.

This redundancy is clearly undesirable.

Emp is in BCNF because no FD holds in Emp. We need to define a fourth normal form that is stronger than BCNF.

Properties of MVD

Assume that all attribute are included in universal relation schema

R = { A_1, A_2,…….A_n} and that

X, Y, Z and W are subset of R

1. Reflexive rule for FD if X ≥ Y, then X —>Y.
2. Augmentation rule for FD X—> Y ⊨ XZ —> YZ
3. Transitive rule for FD { X —>Y, Y —> Z} ⊨ X—>Z
4. Complementation rule for MVD X —>—>Y ⊨

   X—>—>{R – (X U Y)}

5. Augmentation rue for MVD if X—>—> Y and

   W ≥ Z then

   WX —>—>YZ

6. Transitive rule for MVD { X—>—>Y, Y—>—>Z} ⊨ X —>—>Z – Y
7. Replication rule (FD to MVD) X—>Y = X—>—>Y
8. Coalescence rule for FD as MVDs

If X —>—>Y and there exist W with the properties that

(a) W ∩ Y is empty
(b) W —> Z

(c) $Y \geq Z$ then $X \twoheadrightarrow Z$

Fourth Normal Form

A relation schema R is in 4NF with respect to a set to dependencies f if for every non trivial multivalued dependency $X \twoheadrightarrow Y$ in f^+, X is a super key for R.

EMP relation is not in 4NF because in the non trivial MVDs EName ⟶⟶ PName and EName ⟶⟶Dname

EName is not a super key of EMP

We decompose EMP into EMP-Project and EMP – dependent

Both EMP – project and EMP – dependent are in 4th NF. Because Ename ⟶⟶ Pname is a trivial MVD in EMP – project.

In fact no non trivial MVDs hold in either EMP – project or EMP – dependent

Emp

EName	PName	DName
Smith	X	John
Smith	Y	Anna
Smith	x	Anna
Smith	y	John
Brown	w	Jim
Brown	X	Jim
Brown	y	Jim
Brown	z	Jim
Brown	w	Joan
Brown	x	Joan
Brown	y	Joan
Brown	z	Joan
Brown	w	Bob

emp - project

EName	PName
Smith	X
Smith	Y
Brown	W
Brown	X
Brown	Y
Brown	Z

emp – project

EName	DName
Smith	John
Smith	Anna
Brown	Jim

Brown	x	bob	Brown	Joan
Brown	y	bob	Brown	Bob
Brown	z	bob		

Bigger in size — smaller in size

16 tuples — 11 tuples

48 facts — 22 facts

Update anomalies is avoided

Input another project into Brown

We need to insert into three columns

EMP is not in 4NF

PROJECTING MVDs

R (A B C D E) S (A B C)

MVDA —>—> CD

Chain

A —>—> C holds in S

A —>—> B by complementation

Let us verify that A —>—>C holds in S

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b	c_2	d	e

A—>—> CD

A	B	C	D	E
a	b_1	C	d_1	e_1
a	b	c_2	d	e
a	b_1	c_2	d	e_1
a	b	C	d_1	e

The last row has un-subscripted symbols in all the attributes of S, that is A, B and C. This is enough to conclude that A—>—>C holds in S

We	street	city	title	year
C. fish	123 Maple str.	Holly	Star wars	1977
C. Fish	5 locust st.	Malibu	Star wars	1977
C. fish	123 maple st.	Holly	Empire back	1980
C. Fish	5 locust st.	Malibu	empire back	1980
C. Fish	123 maple st	Holly	Return Mecca	1983
C. Fish	5 locust st.	Malibu	Return Mecca	1983

name —>—> street city

Reasoning about multi valued dependencies

- Trivial MVD

  A_1, A_2, A_3….. A_n—>—> B_1 B_2 B_3……B_m

  Holds in relation if $\{B_1 B_2 \ldots\ldots B_m\} \subseteq \{ A_1 A_2;\ldots..A_n\}$

- Transitive rule $A_1 A_2\ldots\ldots A_n$—>—>$B_1 B_2\ldots\ldots.B_m$ and

  $B_1 B_2\ldots\ldots B_m$—>—>$C_1 C_2\ldots\ldots.C_K$

  then

  $A_1 A_2\ldots\ldots..A_n$—>—>$C_1 C_{2,}\ldots\ldots, C_K$

- FD promotion every FD is an MVD = That is, if

$A_1, A_2, A_3 \ldots\ldots A_n \rightarrow\rightarrow B_1\ B_2 \ldots\ldots B_m$ then

$A_1, A_2, A_3 \ldots\ldots A_n \rightarrow\rightarrow B_1\ B_2 \ldots\ldots B_m$

- Complementation rule

If $A_1, A_2, A_3 \ldots\ldots A_n \rightarrow\rightarrow B_1\ B_2\ B_3 \ldots\ldots B_m$ is an MVD for R then R also satisfies $A_1, A_2 \ldots\ldots A_n \rightarrow\rightarrow C_1\ C_2 \ldots\ldots C_K$ where the C's are all attributes of R not among the A's and B's

name $\rightarrow\rightarrow$ street city compliment for

name $\rightarrow\rightarrow$ title year

Fourth Normal form

Relation R is in fourth normal form 4NF if whenever

$A_1, A_2, A_3 \ldots\ldots A_n \rightarrow\rightarrow B_1\ B_2 \ldots\ldots B_m$ is a non trivial MVD,

$\{ A_1\ A_2\ A_3, \ldots\ldots A_n\}$ is a super key.

name $\longrightarrow$ street, city

name $\longrightarrow$ year title

R_1 (name, street, city) R_2 (name, year, title)

Multivalue dependency

$X \rightarrow\rightarrow Y$ tells us that it we find two rows of the tableau that agree in X, then we can form two new tuples by swapping all their component in the attributes of Y

R (A B C D)

$A \rightarrow B$

$B \rightarrow\rightarrow C$

Prove that $B \rightarrow\rightarrow C$, $A \rightarrow\rightarrow$ Cholds in R

A	B	C	D		A	B	C	D
a	b_1	c	d_1	A—> B	a	b	c	d_1
a	b	c_2	d		a	b	c_2	d

A	B	C	D
a	b	c	d_1
a	b	c_2	d
a	b	c_2	d_1
a	b	c	d

A—>—> C

Given two tuples of R that agree on A, they must also agree in B, A —>B

MVD

X —>—>Y and any FD whose right side is a (not necessarily proper) subset of Y, say Z then X —>Z

R (A	B	C	D)	A	B	C	D	
MVD	A —>—>BC			a	b_1	c_1	d_1	A—>—>BC
				a	b_2	c_2	d_2	

FD D —>C

We claim that A ——>——> C

A	B	C	D	
a	b_1	c_1	d_1	
a	b_2	c_2	d_2	
a	b_2	c_2	d_1	D —>C
a	b_1	c_1	d_2	

A	B	C	D	
a	b_1	c_1	d_1	
a	b_2	c_2	d_2	
a	b_2	c_2	d_1	We have proved A⟶C
a	b_1	c_1	d_2	

Computing the closure of a set of Attributes

Input: A set of Attributes {A1,A2, A3,...,An} and a set of FD′s .

Output : The closure $\{A_1, A2, A3, \ldots, An\}^+$

1. If necessary split the FD's of S so each FD in S has a single attribute on the right
2. Let X be a set of attributes that eventually will become the closure

   Initialize X to be {A1 ,A2, A3,…,An}
3. Repeatedly search for some FD

   B1 B2 B3…, Bm → C

   Such that all of B1, B2,….,Bm are in the set of attributes X, but C is not. Add C to the set X and repeat the search. Since X can only grow and the number of attributes of any relation scheme must be finite, eventually nothing more can be added to X, and this step ends.
4. The set X after no more attributes can be added to it, is the correct value

   $\{A_1, A2, A3, \ldots, An\}^+$

Algorithm for projecting a set of functional Dependencies

Input: A relation R and a second relation R1 computed by the projection R1 = ∏2 (R).

Also a set of FD′s S that hold in R .

Output : The set of FD′s that hold in R1

method

1. Let T be the eventual output set of FD′s.Initially, T is empty .
2. For each set of attributes X that is a subset of the attributes of R1 , compute X^+. This computation is performed with respect to the set of FD′s S and may involve attributes that are in the schema of R but not R1. Add to T all nontrivial FD′s X →A such that A is both in X^+ and an attribute of R1.
3. Now, T is a basis for the FD′s that hold in R1, but may not be a minimal basis. We may construct a minimal basis by modifying T as follows:
   a) If there is an FD F in T that follows from the other FD′s in T, remove F from T.
   b) Let Y → B be an FD in T with at least two attributes in Y ,and let Z be Y with one of its attributes removed. If Z → B follows from the FD′s in T (including Y → B) ,then replace Y → B by Z → B.

   Repeat the above steps in all possible ways until no more changes to T can be made.

Decomposition Algorithm

Input: A relation R_0 with a set of functional dependencies S_0.

Output: A decomposition of R_0 into a collection of relations, all of which are in BCNF

Method: The following steps can be applied recursively to any relation R and set off FD′s S. Initially, apply them with R = R_0 and S = S_0.

1. Check whether R is in BCNF. if so nothing more needs to be done. Reduction {R} as the answer.
2. If there are BCNF violations, let one be X → Y.use algorithm closure X^+. Choose R1 = X^+as one relation schema and let R2 have attributes of X and those attributes of R that are not in X^+.

3. Use algorithm for projecting a set of functional dependencies to compute the set of FD′s for R1 and R2: let these be S1 and S2 respectively.
4. Recursively decompose R1 and R2 using this algorithm. Return the union of the results of these decompositions.

In database design, to test for lossless decomposition, one may use the following algorithm

1. Create a matrix S such that
    - a row for each Ri in D and
    - a column for each Aj in R
2. Each S (i , j) = bij for all i ,j
3. for i =1 to n

    for j =1 to m

    if Aj an element of Ri , set bij = aj
4. repeat for each X→Y in f for all rows where they correspond in X.if there is an a in Y set all Y′s to a

    else

    pick some bij and set all of the Y′s that bij

    until

    no change in S.
5. If there exists a row with all a′s, then it's a lossless decomposition

3NF Decomposition algorithm

Let Fc be a canonical cover for F,Type equation here.

i = 0;

for each functional dependency

$\alpha \rightarrow \beta$ in Fc do

if none of the schemas Rj i$\leq j \leq i$

contains $\alpha\beta$.then

begin

i = i + 1;

Ri = $\alpha\beta$

end

if none of the schemas Rj $1 \leq j \leq i$

contains a candidate key for R then

begin

i = i + 1;

Ri = any candidate key for R,

end

return (R1, R2,R3,…, Ri)

Canonical Cover algorithm

Fc computation algorithm

Fc = F

repeat

apply union rule (right side of fd)

find fd with extraneous attributes (left | right side)

and delete these

until Fc does not change.

BCNF Decomposition algorithm

result := {R}

done := false;

compute F^+

while (not done) do

if (there is a schema Ri in result that is most in BCNF) then

Begin

let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on Ri such that $\alpha \rightarrow Ri$ is not in F^+ and $\alpha \cap \beta$ =ø ;

result := (result – Ri) ∪ (Ri $-\beta$) ∪ (α, β),

end

else

done:= true

When we decompose a relation, we have to use natural joins or Cartesian products to put the pieces back together .This takes computational time.

Comparison of BCNF and 3NF

It is always possible to obtain a 3NF without sacrificing lossless join on dependency preservation.

- If we do not eliminate all transitive dependencies, we may need to use null values to represent some of the meaningful relationships.
- Repetition of information occurs.
- If we must choose between BCNF and dependency preservation, it is generally better to opt for 3NF.
- If we cannot check for dependency preservation efficiently, we either pay a high price in system performance or risk the data.
- The limited amount of redundancy in 3NF is then a lesser evil.
- BCNF
- Loss-less join
- Dependency preservation

Are goals of relational design.

Algorithm : 3rd NF with a lossless join and dependency preservation

Input: A relation R and a set of FD that holds in R .

Output: A decomposition of R into a collection of relations, each of which is in 3NF.

The decomposition has the lossless join and dependency preservation properties.

Method: perform the following steps

1. Find a minimal basis for F, say G
2. For each functional dependencies X → A in G,use XA as the schema of one of the relations in the decomposition.

3. If none of the sets of relations from step 2 is a super key for R, add another relation whose schema is a key for R.

R (A B C D E)

FDs

$AB \rightarrow C$

$C \rightarrow B$

$A \rightarrow D$

Use closure AB for $C \rightarrow B$, $A \rightarrow D$

A B D = D is included but not C

We conclude that the first FD $AB \rightarrow C$ is not implied by the second and third FD′s. we get a similar conclusion if we try to drop the second or third FD.

- We must also verify that we cannot eliminate any attributes from a left side.
- We start the 3NF synthesis by taking the attributes of each FD as a relation schema.

{A,B,C}

{C, B}

{A, D}

It is never necessary to use a relation whose schema is a proper subset of another relations schema, so we can drop S2 {A, B,C} and {A,D}.

R has two keys = {A,B,E} {A,C,E}.Neither of the keys is a subset of the schema chosen so far. We must add one of them R4 {A, B, E} = Final decomposition is R1 {A, B, C} R2 {A,D} R3 {A, B, E}

Relation (Town, street, code)

FD; town ,street

Professor (SSN, name, age, rank, specialty)

Compute canonical cover

Attribute and Dependency Preservation

R = {A1, A2, …, An} is to be decomposed into a set D of relation schemas,

D = {R1, R2,…,Rn}.The decomposition is said to satisfy the attribute preservation condition if

R = {A1,A2, …, An} = R1∪ R2 ∪ …∪Rn.

If a decomposition is not dependency preserving, some dependency is lost in the decomposition.

Decomposition and lossless (nonadditive) joins.

A decomposition is called lossless (nonadditive) when natural joins applied to the relations in the decomposition do not generate spurious tuples. Loss is lossless refers to the loss of information and not the loss of tuples. Actually the loss of information occurs because of added (spurious) tuples in the joins. A decomposition which is not lossless is called lossy.

WEEK –EIGHT

RELATIONAL ALGEBRA

In the relational data model, attribute relationships are represented by relations. Relational data model, unlike, network model, does not provide links to represent associations. Instead, another relation is used to represent an association.

Relations that represent associations can be existing relations in the database or they can be generated (created) from existing relations by using relational operators.

The relational operators can be described using either the relational algebra or relational calculus.

Relational algebra is a set of operators that constructs the required relation from given relations.

The relational calculus gives a definition of the desired relation

Some terminology and definitions

∃ there exists

∀ for all

∧ and

∨ or

- not complement

∈ belongs to member of

⊆ subset

∅ empty

: such that

[,] (, and) delimiters

Definitions that will be needed

Given the tuples r = <r1, ……,rm> and

S = <s1, ……,sn>

the concatenation of r with s is the

(m+n) tuple defined by

$\bar{rs}$ = <r1,…., rm,s1,….,sn >

for example, if r = <1, 2, x> and

s = <a, 2, 3> then

$\bar{rs}$= <1, 2, x, a, 2, 3 >

2 Let R be an n-array relation, r∈ R a tuple of R, and {D_1, …., D_n} the domains of R; then

1 r [D_i] designate the ith component (value of D_i) of r.

2 If A ⊆ {D_1, ….., D_n}, then

(a) r [A] is a tuple containing only those components specified by A

e.g if r = <a, 2, f> and

R is R (D_1, D_2 , D_3) then

r [D_1, D_3] = <a, f>

(b) R [A] = {r[A]: r∈R}, e.g. if R (D_1, D_2, D_3) is

R[D_3, D_2] =

f	2
g	1
f	3
g	3
f	2

R (D_1, D_2, D_3)

a	2	f
b	1	g
c	3	f
d	3	g
e	2	f

then R[D_1] =

a
b
c
d
e

Let T (x, y) be a binary relation. The image set of x under T is defined by

g_T (x) = {y_i <x, y> ∈ T}

For example, if R is

R (D_1, D_2)

1	a
1	b
2	c
1	d

then

g_R (D_1 = 1) = { <a>, <b>, <d>}

g_R (D1 =2)= {<a>}

g_R (D1= 3) = {∅}

4. Given an n-array relation R over a set f domains {D_1,….., D_n} and a k-tuple (k≤ n) of domains A (A ⊆ {D_1, ……,D_n}) then

1 $\bar{A}$ = {D_1,…., D_n} – A ($\bar{A}$ *contains all the domain not in A*).

2 If r is an n-tuple of R, then

g_R (r [$\bar{A}$]) = {s:s ∈ R [A] < r [$\bar{A}$, s] > ϵ R [$\bar{A}$, A]}

if the relation R is

R (D_1	D_2	D_3	D_4	D_5)
1	a	x	f	2
2	a	y	g	3
1	b	x	f	2
2	c	y	b	3
3	a	x	f	1
1	b	y	f	2
2	a	x	b	3

And A ={D_3, D_2, D_4}, then $\bar{A}$ = {D_1, D_5}

Let r = <1, a, x, f, 2>;

Then r [A] = <x, a, f>

And r [$\bar{A}$] = <1, 2>

And

g_R (r [$\bar{A}$] =g_R (<1, 2>)]) = {<x, a, f>, <x, b, f>, <y, b, f>}

5. Two sets of attributes A and B are compatible if they are of the same degree and the corresponding domain are of the same data type.

BUYER (NAME,	ITEM)
SMITH	A
JONES	B
ADAMS	A
SMITH	B
JONES	A
SMITH	C

PRODUCT (CODE	COST	PRICE)
A	5	8
B	4	4
C	6	9

The relations deposit the products manufactured by a company (their code, production costs, and selling price), and the buyers of those products (their names and the products they buy).

For each operator, the expression on the left-hand side of the equality sign is the relational algebra expression for the operator. The expression on the right-hand side is the relational calculus definition of the operator.

The first three relational operators are required to obtain any subset of a given relation. For example, suppose that a user waits those tuples in the PRODUCT relation where the PRICE attribute value is less than or equal to 8.

Restriction operator can express this requirement as

PRODUCT [PRICE $\leq$8] = (code, cost, price)

A	5	8
B	4	4

More finally, restriction is defined as

R [AθV] = {r:r ∈R ∧ (r[A]=v)}

Where A is an attribute of R

θ(theta) is one of the conditional operators $<, \leq, >, \geq, =, or \neq$

V is a literal value

The restriction operator is equivalent to a qualification, containing a single condition, on a single relation. It requires a specific data value (8 in the preceding example) in the condition involving two attributes of the same relation. For example, suppose that one would like to know which products are being sold at cost. The selection operator can specify this data selection as

PRODUCT [PRICE = COST] = (CODE, COST, PRICE)

B 4 4

In this case, the qualification specifies a condition involving the two attributes PRICE and COST in the relation PRODUCT. The two attributes must be compatible. That is, they must be of the same data type. The result relation contains only those tuples of PRODUCT where the COST attribute value is equal to the PRICE attribute value. More formally, selection is defined as

R [A θ B] = {r:r ∈ R (r [A]θr [B])}.

Projection operator can be used to perform this selection. For example, if a user wants to know names of all buyers of products, this data selection specified as

BUYER [NAME] = (NAME)

Smith

Jones

Adams

In addition, any duplicate tuples are also eliminated. More formally, projection is defined as

R [A] = {r [A] : r ∈ R}

The cross-product operator forms all possible combination of the tuples of two relations.

For example, the cross product of BUYER and PRODUCT is

BUYER ⊗ PRODUCT = (NAME, ITEM, CODE, COST, PRICE)

1.	Smith	A	A	5	8
2.	Jones	B	A	5	8
3.	Adams	A	A	5	8
4.	Smith	B	A	5	8
5.	Jones	A	A	5	8
6.	Smith	C	A	5	8
7.	Smith	A	B	4	4
8.	Jones	B	B	4	4
9.	Adams	A	B	4	4
10.	Smith	B	B	4	4
11.	Jones	A	B	4	4
12.	Smith	C	B	4	4
13.	Smith	A	C	6	9
14.	Jones	B	C	6	9

15. Adams	A	C	6	9
16. Smith	B	C	6	9
17. Jones	A	C	6	9
18. Smith	C	C	6	9

More formally, cross product is defined as

R $\otimes$ S = $\{(\bar{rs}) : r \in R\ s\ \in S\}$

Suppose that a user wants a list of buyer names, the products they buy and the cost and price of each product. The answer to this query is contained in two relations, BUYER and PRODUCT. A new relation containing the answer to the query can be constructed by taking the join of BUYER and PRODUCT according to a join condition.

The join condition is expressed on two compatible attributes, one from each of the original relations.

BUYER [ITEM = CODE] PRODUCT = (NAME ITEM CODE COST PRICE)

Smith	A	A	5	8
Jones	B	B	4	4
Adams	A	A	5	8
Smith	B	B	4	4
Jones	A	A	5	8
Smith	C	C	6	9

More formally, join is defined as

R [Aθ ***B***] S = $\{(\bar{rs})$: r $\in R\ \ s \in S\ (r\ [A]\theta\ s\ [B])\}$

Where again θ is one of $< \leq > \geq = or\ \neq$

The attribute A and B must be compatible.

Join

(1) Generalize join forms a new tuple from a BUYER and PRODUCT tuple whenever the join condition is satisfied.
(2) A natural join is a join where the conditional operator is equality.

“Find the buyers who buy each type of product”

Data selection can be represented by a division operator as

BUYER [ITEM÷ $CODE$] product

= (Name)

Smith

More formally, division is defined as

R[A÷ B] S = {r [$\bar{A}$]: r ∈ R ∧ s [B]⊆ g_R (r [$\bar{A}$])} where the attribute A and B are compatible.

The result relation consists of the projection of the tuples in the dividend relation on those attributes (NAME) not in the dividend attribute that satisfy the division.

Finally, the projection of PRODUCT on CODE is

PRDUCT [CODE] = (CODE)

A

B

C

Operation of mathematical sets ∪ =Union

∩ = intersection

– = difference

Find those buyers who purchase products whose price is greater than 5 but less than 9 would be expressed as

(BUYER [ITEM = CODE] (PRODUCT [PRICE > 5]))

PRODUCT [PRICE < 9] [NAME]

Operation of any Data model

- Basic data managemet operations
- Basic arithmetic operations
- Sort
- Summary
- Union =set1+ set2 ∪

- Intersection = set1 *set2 ∩
- Difference = set1-set2 - =[1, 3, 4]- [1, 2, 4] = [3]
- Division = set1/set2
- Equality = [1, 3] = [1,3] = true
- Inequality = [1, 3] <> [2, 4]
- Subset =⊆ = [1, 3]<= [1, 2, 3, 4]
- Proper subset= [1,3]< [1, 2, 3, 4]
- Super set = [1, 2, 3,4]>= [1, 3]
- Proper superset =[1, 2, 3, 4] > [1,3]

Query =search

Relational operation.operates

Concatenation of $\bar{r}\bar{s}$

R (D1 D2 D3) r2 <a, 2, f> r [D1 D3] = <a, f>= r [A]

R (D1 D2 D3) R(D1 D3)= R[A]

Binary relation = g_R (D1 =1) = {<a>, <b>, <c>}

Many relation = g_R r [$\bar{A}$]

WEEK NINE

RELATIONAL ALGEBRA

Four broad classes

Set operations = union, intersection, difference

R∪ S = the union of R and S

Is the set of element appears in R and S or both. An element appears only once in the union even if it is present in both R and S.

R ∩ S = the intersection of R and S is the set of elements that in both R and S

R – S = the difference of R and S is the set of element that are in R but not in S. Note that R-S is difference from S-R. The letter is the set of element that are in S but not in R.

2. Operations that remove part of relation

Selection = eliminate some row tuples

Projection = eliminate some column

R [D1	D2	D3]
a	2	f
b	1	g
c	3	f

R [D1]
a
b
c

R [D3	D2]
f	2
g	1
f	3

R [D1	D2]
1	A
1	B
2	C
1	d

g_R (D1 = 1) = {<a>, <b>, <d>}

g_R (D1 = 2) ={<c>}

g_R (D1 = 3) = {∅}

R (D1	D2	D3	D4	D4)

r = <1, a, x, f, 2>

1	a	x	f	2	R [A] = {x, f, a}
2	a	y	g	3	R [A] = {1,2}
1	b	x	f		
2	c	y	b	2	

A = {D3 D2 D4} $\bar{A}$ = {D1 D5}

Projection = a new relation that has only some of R column

π A1, A2,……, An (R) is a relation that has only the columns attributes A1, A2, ….., An of R.

Movie

title	year	length	In column	Studio Name	Producer
Star war	1977	124	True	fox	12345
Might ducks	1991	104	True	Disney	07890
Wayous world	1992	95	true	Paramount	99999

π title, year, length (movie)

Title	year	Length
Star wars	1977	124
Might ducks	1991	104
Wagnus world	1992	95

π in color (movie) =

Incolor
true

Selection = operator applied to a relation R provides a new relation with a subset of R's tuples. The tuples in the resulting relation are those satisfy some condition C that involves the attributes of R. We denote this operation

б C (R) or $б_f$ (R) f is formula

б length ≥ 100 (movie)

Title	Year	length	Incolor	Studio name	Producer
Star wars	1977	124	True	Fox	12345
Might ducks	1991	104	True	disney	67890

б length ≥ 100 AND studio name = “fox” (movie)

Title	year	length	incolor	Studio name	Producer
Star war	1977	124	true	fox	12345

Operations that combine the tuples of two relation including Cartesian product or cross product

Join operation = selecting pair tuples from two relations

Cartesian product, cross product or product of two set R and S is the set of pairs that can be formed by choosing the first element of the pair to be any element of R and the second on element of S. This product is denoted by R ⊗ S

The components from R precede the components from S in this order.

R

A	B
1	2
3	4

S

B	C	D
2	5	6
4	7	8
9	10	11

B is an attribute of both schemes

We have used R.B and S.B in the schema for R × S

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Natural join

We find a need to join two relations by pairing only those tuples that match in some way. The simplest sort of match is the natural join of two relations R and S.

Natural join is denoted by R ⋈ S

Let A1, A2, …., An be attribute in both the schema of R and the schema of S then a tuple r from R and a tuple s from S are successfully paired if and only if r and s agree on each of the attribute A1, A2, …., An

A	B	C	D
1	2	5	6
3	4	7	8

B	C	D
9	10	11

A tuple that fails to pair with any tuple of other relation in join is sometimes said to be a dangling tuple.

U

A	B	C
1	2	3
6	7	8
9	7	8

V

B	C	D
2	3	4
2	3	5
7	8	10

U⋈ V

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Theta- joins

The natural join forces us to pair tuples using one specific condition.

Theta join refers to an arbitrary condition which we shall represent by C

$R \bowtie_C S$

1. Take the product of R and S
2. Select from the product only those tuples that satisfy the condition C

$U \bowtie_{A<D} V$

A	U.B	U.C	V.B	V.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

$U \bowtie_{A<D \text{ AND } U.B \neq V.B} V$

A	U.B	U.C	V.B	V.C	D
1	2	3	7	8	10

Combining operations to form queries

What are the title and years of movies made by fax that are at least 100 minute long ?

To compute

1. Select those movies tuples that have length ≥ 100
2. Select those movies tuples that have studio name = “fox”
3. Compute the intersection of 1 and 2
4. Project the relation 3 onto attributes title and year.

π title, year

↓

n

↙ ↘

б length ≥ 100 б studio name = “fox”

↓ ↓

Movie movie

π title, year (б length ≥ (movie) ∩ б studio name = “fox” (movie))
Movie1 (title, year, length, filmType, studioName)
Movie2 (title, year, starName)
Find the stars of movie that at least 100 minute long
π title, year (б length ≥ 100 (movie1⋈ movie2)
Renaming
= P_s (R) = change the name to S
P_{ayo} (bell) = change the name to ayo from bell
P_s (x, c, d) (S) is a relation named S but its first column has attribute X instead of B

Uncle (x, y) P
↙ ↘
U S
Grandparent (p, y) mother (P, X)
Sister (S, X)

G. parent X grandchild (y, x) is parent (x, z)
↓ parent (z, y)
Parent Z sister (x, y) = parent (p, x) parent (p, y)
↓
Chid Y brother (x, y) = parent (z, x)
Parent (z, y)
Male (x) x≠ y

Male = {Adam, Bill}
Female = {Anne, Beth}
Person = male ∪ female = {Adam, Bill, Anne, Beth}

Male × person {Adam, Bill} {Adam, Bill, Anne, Beth}

Adam	adam	bill	adam
Adam	bill	bill	bill
Adam	anne	bill	anne
Adam	beth	bill	beth

Father		mother		parent	
C1	C2	C1	C2	C1	C2
Adam	bill	anne	bill	adam	bill
Adam	beth	anne	beth	adam	beth
				Anne	bill
				Anne	beth

As a collection of tuples of feets
Father <adam, bill>
Father <adam, beth>
Mother <anne, bill>
Mother <anne, beth>
Parent <adam, bill>
Parent <adam, beth>
Parent <anne, bill>
Parent <anne, beth>

A and B are isomorphic

Deductive database
Parent (X, Y) ← father (X, Y)
Parent (X, Y) ← mother (X, Y)
Father (adam, bill)
Father (adam, beth)
Mother (anne, bill)
Mother (anne, beth)
Grandparent (X, Z) ← parent (X, Y), parent (Y, Z)
Parent (X, Y) ← father (X, Y)
Parent (X, Y) ← mother (X, Y)

Father

X	Y
adam	Bill
Bill	cathy

a

Parent

Y	Z
Adam	Bill
Bill	Cathy
Cathy	dove

b

X	F.Y	P.Y	Z
Adam	Bill	adam	Bill
Adam	Bill	Bill	cathy
Adam	Bill	Cathy	Dove

Bill	Cathy	Adam	Bill
Bill	Cathy	Bill	Cathy
Bill	cathy	Cathy	dove

X	F.Y	P.Y	Z
Adam	Bill	bill	Cathy
Bill	cathy	Cathy	dove

X	F.Y	Z
Adam	Bill	Cathy
Bill	Cathy	Dove

Natural join
Taking the Cartesian product of the two relations
Selecting those tuples which have identical attribute on the columns with the same attribute
Filtering out the super flows columns

F (X, Y) ⋈ P (Y, Z)
F (X, Y) ⋈ P (Y, Z) is defined as

$$\pi\, X, F.Y, Z\ б\ F.Y = P.Y\ (f\ (X, Y) \times P\ (Y, Z))$$

1. Cartesian product F (X, Y) × P (Y, Z)
2. Selected the same tuple of the same value F.Y and P.y

Projection X, F.Y, Z on X and Z

$$\pi X, Z\ (F\ (X, Y) \bowtie P\ (Y, Z))$$

Or

Grandfather (X, Y, Z) ← father (X, Y) , parent (Y, Z)

Or

Grandfather (X, Z) ← father (X, Y), parent (Y, Z)

Consider the domain

{Sarah, Diane, Pamela, Simon, David, Peter }

r (X, Z) ← r (X, Y), r (Y, Z)

r (a, c)∈ P for which there exist no b (b ≠ a and b ≠ c) such that r (a, b) ∈ P and r (b, c) ∈ P

a
b
c
d

The relation is position over

Over (a, b) over (a, b)

Over (a, d) over (b, c)

Over (b, d) over (c, d)

Deductive database

Over (X, Z) ← over (X, Y), over (Y, Z)

Over (a, b)

Over (b, c)

Over (c, d)

Or

Over (X, Y) ← on (X, Y)

Over (X, Z) ← on (X, Y), over (Y,Z)

On (a, b)

On (b, c)

On (c, d)

(b) (d) (f)

(a)Type equation here.

(C) (e) (g)

Edge (a, b) edge (c, e)

Edge (a, c) edge (d, f)

Edge (b, d) edge (e, f)

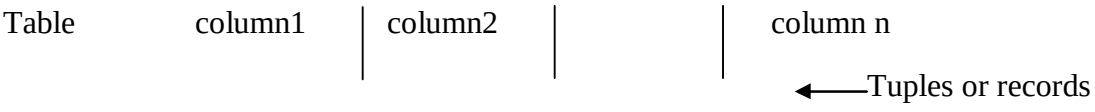
Edge (b, e) edge (e, g)

Path (X, X)

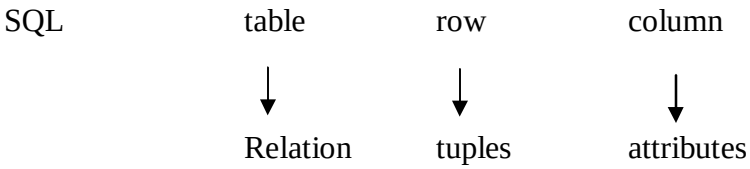
Path (X, Z) ← edge (X, Y) path (Y, Z).

WEEK – TEN

SQL



- A query issued against a DBS also results in a table
- Column has data type
- This data type describes the attribute of the tuples/records



- A table can have up to 254 columns with the same or different data types
- A set of values (domain)

ORACLE BASIC DATA TYPES

⇒ Char (n) = fixed_length character data n long
= maximum size for n is 255 bytes

= e.g. char (40)

⇒ Varchar 2(n) = variable_length character data string
= maximum size for n is 2000

= e.g. varchar (80)

⇒ Number (o, d) = numeric data type for integers and reals
o= overall number of digits

d= number of digits to the right of the decimal point

o= maximum number is 36

d= maximum value = -84 to +127

e.g. number(8)

Number (5, 2) = this cannot contain any thing larger than 999.99 without resulting in an error.

Data type derived from number are integer = int

Decimal = dec

Smallint

Real

- Date – for storing date and time.
  Date format = DD-MMM-YY

  23-OCT-94

  07-JAN-07

- Long = character up to length 2GB

In Oracle SQL = no data type 'Boolean'. It can be simulated by using char (1) or number (1).

It may have value null. Null is different from empty string '' or 0.

Example database

EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, DEPT. NO.)

7369 Smith clerk 7902 17-Dec-80 800 20

Data type = EMPNO: number (4), ENAME: varchar2 (30);

JOB: char (10), MGR: number (4), HIREDATE: date,

SAL: number (7,2), DEPTNO: number(2)

DEPT (DEPTNO | DNAME | LOC)

DEPTNO	DNAME	LOC
10	store	Chicago
20	research	Dallas

SALGRADE (	GRADE	LOWSA	HIGHSAL)
	1	700	1200
	2	1201	1400
	3	1401	2000

QUERIES

SQL is used to retrieve information from database.

Form:

Select distinct <columns>

From <table>

Where<condition>

Order by<column>asc/desc

Select is also called projection

Select loc, DEPTNO

From DEPT;

Select* (asterisk symbol * is used to denote all attributes)

From DEPT;

Select may contain arithmetic expression

Select ENAME, DEPTNO, SAL* 1.55

From EMP;

Operators

a) for numbers: abs, cos, sin, exp, log, power, mod, sqrt, +,-,*,/
b) for strings: char, concat (string1, string2),
lower, upper,

replace (string, search_ string, replacement_string)

` substr(string, m, n)

length, to_date, translate

c) for date: add.month
month.between
next.day
to.char

- Distinct = after the keyword select, forces the elimination of duplicates from the query result.
- Order by = works with column.

Select ENAME, DEPTNO, HIREDATE

From EMP

Order by DEPTNO asc, HIREDATE desc

ENAME	DEPTNO	HIREDATE
Ford	10	03-DEC-81
Smith	20	17-DEC-80
Blake	30	01-MAY-81
Ward	30	21-FEB-81
Allen	30	20-FEB-81

SELECTION OF TUPLES/RECORDS

Where = for records retrieval

= simple operator = and, or, not

= condition may be pattern matching

List the job title and the salary of those employees whose manager has the number 7698 or 7566 and who earn more than 1500

Select JOB SAL

From EMP

Where (MGR = 7698 or MGR = 7566)

And SAL > 1500;

1. Comparison operator =, /= or <>,<= are allowed

Further comparison operators are

2. Set condition <column> not, in <list of values>

```
select *
from DEPT
where DEPTNO in (20, 30)
```

3. NULL value <column> is, not, null

```
Select *
From EMP
Where MGR is not null
```

4. Domain conditions <column> not between <lower bound > and <upper bound>

```
select EMPNO, ENAME, SAL
from EMP
where SAL between 1500 and 2500,
select ENAME
from EMP
where HIREDATE between '02-APR-81' and '08-SEP-81'
```

String Operations

Like = uses two operators %, _ (percentage, and underline)

Percentage = wild card

Underline = position marker

If one is interested in all tuples of the table DEPT that contain two C in the name of the department, the condition would be

```
Where DNAME like %C%C%.
```

% sign means that any sub(string) is allowed there even the empty string. Underline stands for exactly one character where DNAME like %C_C% would require that exactly one character appears between the two Cs.

1. upper (string)
   DNAME = UPPER (DNAME)
2. lower (<string>) converts letters to lower case
3. initcap(<string>) converts initial letter to upper case
4. length (<string >) returns the letter of the string
5. substr(<string>n, m)
6. substr('DATABASE SYSTEMS', 10, 7) returns 'SYSTEMS'

Aggregate function

1. count = count rows

How many tuples are stored in the relation EMP?

```
select count (*)

from EMMP
```

How many different job titles are stored in the relation EMP?

```
select count (distinct JOB)

from EMP
```

2. MAX
3. MIN

List the minimum and maximum salary

```
Select min (SAL), max(SAL)

From EMP
```

4. SUM = sum of all salaries of employees working in the department 30

```
Select sum (SAL)

From EMP

Where DEPTNO = 30
```

QUERIES

Join tuples

```
Select distinct [<alias a_k>.]<column i>, …

                [<alias a_l>.]<column j>

                from<table 1>[<alias a_1>], …., <table n >[<alias a_n>]

                where [<condition>]
```

For each salesman, we now want to retrieve the name as well as the number and the name of the department where he is working.

```
select ENAME , E.DEPTNO, DNAME
From EMP E, DEPT D
Where E.DEPTNO = D.DEPTNO
And JOB = 'SALESMAN';
```

```
select ENAME , DEPTNO, DNAME
from EMP, DEPT
Where EMP.DEPTNO = DEPT.DEPTNO
And JOB = 'SALESMAN';
```

E and D are table aliases for EMP and DEPT respectively.

PROJECT (PNO, PNAME, PMGR, PERSONS, BUDGET, PSTART, PEND)

EMP (EMPNO, EName, JOB, MGR, HIREDATE, SAL, DEPTNO)

DEPT (DEPTNO, DNAME, LOCATION)

SALARYGRADE (GRADE, LOWSAL, HIGHSAL)

e.g. for each project, retrieve its name, the name of its manager, and the name of the department where the manager is working.

```
selectEName, DName, PName
from EMP E, DEPT D, PROJECT P
where E.EMONO = P.MGR
and D.DEPTNO  = E. DEPTNO;
```

It is even possible to join a table with itself:

e.g list the names of all the employees together with the name of their managers

```
select E1.Ename, E2.Ename
from EMP E1, EMP E2
where E1.MGR = E2.EMPNO
```

The join columns are MGR for table E1 and EMPNO for table E2

SUBQUERIES

A respective condition in the **where** clause can have one of the following forms;

1. set-valued subqueries
   expression [not] in <subquery>

   expression<comparison operator> any/all <subquery>

An expression can either be a column or a computed value.

2. test for (non) existence
   [not] exists <subquery>

In **where** clause condition, using subqueries can be combined arbitrarily by using the logical connectives **and**, **or**

e.g. list the name and salary of employee of department 20 who are leading a project that started before December 31, 190

```
selecteName, salary
from EMP in
        (select PMGR
        from PROJECT
        where PSTART <'31-OCT-90')
        and DEPTNO = 20;
```

The subquery retrieves the set of those employees who manage a project that started before December 31, 1990. If the employee working in department 20 is contained in this set (in operator), this tuple belongs to the query result set.

List all employees who are working in a department located in BOSTON

```
select *
from EMP
where DEPTNO in
        (select DEPTNO
        from DEPT
        where LOC =  'BOSTON');
```

A subquery may rise again in a subquery in its **where** clause

List all those employees who are working in the same department as their manager

```
Select *
From EMP E1
Where DEPTNO in
    (select DEPTNO
    from EMP [E]
    where [E.]EMPNO  = E1. MGR);
```

The subquery in this example is related to its surrounding query since it refers to the column E1.MGR. For Each tuple in the table E1, the subquery is evaluated individually.

Condition of the form <expression><comparison operator> [any/all] <subquery> are used to compare a given <expression> with each value selected by <subquery>.

Retrieve all employees who are working in department 10 and who earn at least much as any (i.e. at least one) employee working in department 30

```
Select *
fromEMP
where SAL>= any
    (select SAL
    from EMP
    where DEPTNO = 30)
```

List all employees who are not working in department 30 and who earn more than all employees working in department 30

```
Select *
From EMP
where SAL> all
    (select SAL
    from EMP
    where DEPTNO = 30)
and DEPT NO <> 30;
```

For all and any, the following equivalences hold:

In ⇔ = any

Not in ⇔<> all or != all

Often a query result depends on whether certain rows do (not) exit in (other) tables. Such type of queries is formulated using the **exits** operator.

List all the departments that have no employees

```
Select  *
from DEPT
where not exits
        (select * from EMP
        where DEPTNO = DEPT.DEPTNO)
```

Operations on result sets

SQL supports three set operators which have the pattern

<Query 1><set operator><query 2>

union

intersect

minus

Assuming that we have a table EMP2 that has the same structure and columns as the table EMP,

- all employee numbers and names from both tables

```
select EMPNO, ENAME
from EMP
        union
select EMPNO, ENAME
from.EMP2
```

- Employees who are listed in both EMP and EMP2

```
select *

from EMP

    intersect

select *

from EMP2
```

- Employees who are only listed in EMP

```
select *

from EMP

      minus

select *

from EMP2
```

Each operator requires that both tables have the same data type for the columns to which the operator is applied.

Grouping

Group by<column(s)>

This clause appears after the where clause and must refer to columns of tables listed in the **from** clause e.g. for each department, we must retrieve the minimum and maximum salary

```
Select DEPTNO, min (SAL), max (SAL)

From EMP

Group by DEPTNO
```

Result

DEPTNO	min (SAL)	max(SAL)
10	1300	5000
20	800	3000
30	900	2850

If a group contains less than three rows, this type of condition is specified using the having clause. As for the select clause also in a **having** clause only<group_column(s)> and aggregation can be used

e.g. retrieve the minimum and maximum salary of clerks for each department having more than three clerks

```
select  DEPTNO,Min(SAL), max (SAL)
from EMP
where JOB='CLERK'
group by DEPTNO
having count(*)>3;
```

1. Select all rows that satisfy the condition specified in the **where** clause
2. From these rows form groups according to the **group by** clause
3. Discard all groups that do not satisfy the condition in the **having** clause
4. Apply aggregate functions to each group
5. Retrieve values for the columns and aggregations listed in the **select** clause

Some comments on tables

Accessing tables of other users

Select* from <user><table>,

Adding comments to definitions

- Comment on table

  Comment on table<table> is '<text>',

- Comment on column

  Comment on column<table>.<column> is '<text>'

Comments on tables and columns are stored in the data dictionary. They can be accessed using the data dictionary views USER.TAB.COMMENTS and USER. COL. COMMENTS.

Modifying table and column definition

A column can be added using the **alter table** comment

Alter table<table>

Add <column><datatype>[default<value>]

[<column constraint>]

If more than only one column should be added at one time respectively **add** clause needs to be separated from by colons

A table constraint can be added to a table using

Alter table<table>

Add (<table constraints>)

When the size of strings that can be stored needs to be increased

Alter table<table>

Modify <column><datatype>

Default <value><column constraints>

It is now possible to rename a table, column and constraint

Deleting a table

A table and its row can be deleted by using the commend

Drop table<table>[cascade contraint]

View

To create a view (virtual table) has the form

Create [or replace]view<view name>[<column(s)>] as<select statement>[with check option [constraint<name>]]

Replace recreates the view if it already exists.

The following view contain the name, job title and annual salary of employees working in department 20

Create view DEPT20 as

Select EName, JOB, sal* 12 ANNUAL_SALARY

From EMP

Where DEPTNO = 20

ANNUAL SALARY is specified for the expression SAL*12 and this alias is taken by the view

Alternative

Create view DEPT20(E name, job, ANNUAL_SALARY) as

Select ename, JOB, SAL*12

From EMP

Where DEPTNO = 20;

A view can be used in the same as a table, that is row can be retrieved from a view or rows can be modified.

In Oracle, SQL, no insert, update, or delete modification on views are allowed that use one of the following constraints in the view definition

- Join
- Aggregate functions such as sum, min, max, out etc
- Set-valued subqueries (in, any, all) or test for existence (exits)
- Group by clause or distinct clause

A view can be deleted using the command

Delete <view_name>

DATA DEFINITION IN SQL

Creating tables

The SQL command for creating an empty table has the following form:

Create table<table> (

<Column 1><data type> [not null] [unique] [<column constraint>],

---

<Column n ><data type> [not null] [unique] [<column constraint>],

[<Table constraint(s)>]

);

- For each column, a name and a data type must be specified
- The column name must be unique within the table definition
- Column definitions are separated by comma
- There is no difference between names in lower case letters and names in upper case letters
- In fact the only place where upper and lower case matter are strings comparison
- A **not null** constraint is directly specified after the data type of the column and the constraint requires defined attribute values for that column different from **null.**
- The keyword unique specifies that no two tuples can have the same attribute .value for this column

e.g. the create table statement for our EMP table has the form:

```
create table EMP(
        empno number(4) not null,
        empName varchar2(30) not null,
        job varchar2(10)
        mgr number (4)
        hiredate date
        salary number(7,2)
        deptNo number(2)
        );
```

Note: Except for the columns, empNo and empName null values are allowed.

Checklist for creating tables

- What are the attributes of the tuples to be stored?
- What are the data types of the attributes?
- Should varchar2 be used instead of char?
- Which columns build the primary key?
- Which columns do (not) allow null values?
- Which columns do (not) allow duplicates?
- Are there default values for certain columns that allow null values?

DATA MODIFICATIONS IN SQL

After a table has been created using the **create table** command, tuples can be inserted into the table or tuples can be deleted or modified.

Insertions

i. Insert statement
Form;

Insert into <table> [(column i,…, column j>]

Values(<value i, ….,value j>);

For each of the columns, a corresponding (matching) value must be specified.If a column is omitted, the value **null** is inserted instead.

e.g. a)Insert into PROJECT(PNO, PNAME, PERSONS, BUDGET, PSTART)values(313.'DBS', 7411, NULL, 1500.42,'10-OCT-94');

or

b)Insert into PROJECTvalues (313, 'DBS', 7411, NULL, 1500.42, '10-OCT-94', null);

If there are already some data in other table; these data can be used for insertions into a new table

Insert into <table>[(<column i,…,column j>)]<query>

Create table OLDEMP (

ENO number (4) not null,

HDATE date

We can now use the table EMP to insert tuples into this new relation:

Insert into OLDEMP (ENO, HDATE)

Select EMPNO, HIREDATE

From EMP

Where HIREDATE<'31-DEC-60';

Updates

For modifying attribute values of (some) tuples in a table, we use the update statement;

Update<table> set

<column i> =<expression i>,….

<column j> =<expression j>,….

Where<condition>

Note

- That the new value to assign to <column i> must be matching the data type.
- An **update** statement without a **where** clause results in changing respective attributes of all tuples in the specified table.

e.g. a) the employee JONES is transferred to the department 20 as a manager and his salary is increased by 1000

update EMP set,

JOB = 'MANAGER', DEPTNO = 20, SAL = SAL+1000

Where ENAME = 'JONES';

b)All employees working in the departments 10 and 30 get a 15% salary increase:

Update EMP set

SAL = SAL * 1.5

Where DEPTNO IN (10,30);

We can use query instead of expression:

e.g. all salesmen working in department 20 get the same salary as the manager who has the lowest salary among all managers

update EMP set

SAL = (select min(SAL) from EMP

Where JOB = 'MANAGER')

Where JOB = 'SALESMAN' and

DEPTNO = 20;

The query retrieves the minimum salary form all managers. This value is assigned to all salesmen working in the department 20.

Deletions

Delete from<table>

[where<condition>],

Note if the **where** clause is omitted, all tuples are deleted from the table.

e.g. delete all projects (tuples) that have been furnished before the actual date(system date):

delete from PROJECT

where PEND<sysdate;

Note sysdate is a function in SQL that return the system date.Another SQL function is **user** which returns the name of the user logged into the current oracle session.

Commit and rollback

- ⇒ A sequence of database modifications i.e. a sequence of insert, update, delete statements is called a transaction. Modifications of tuples are temporarily stored in the database system. They become permanent only after the statement **commit** has been issued.
- ⇒ As long as the user has not issued the **commit** statement, it is possible to undo all modifications since the last commit. To undo modifications, one has to issue the statement rollback.
- ⇒ **Note** that any data definition command such as create table results in an internal commit.
- ⇒ A commit is also implicitly executed when the user terminates an oracle session.

Select

from

where

group by

order by

inner join merge rows

insert rows

update - rows

delete - rows

Constraints

In creating table, two types of constraints are provided:

- ⇒ column constraints
- ⇒ table constraint.

Column constraints are associated with a single column.Table constraints are associated with more that one column.

Constraint <name> primary key unique not null

- Constraint can be named in case of violation due to insertion. Two constraints here are **unique** and **not null**
- The most important type of integrity constraints in a database are primary key constraints.
- A primary key constraint enables a unique identification of each tuple in the table. Based on the primary key, the database system ensures that no duplicates appear in a table.

e.g

create table EMP(

EMPNO number (4) constraint pk_emp

Primary key;

)

- Defines the attribute EMPNO as the primary key for the table.
- Each value for the attribute EMPNO must appear only once in the table EMP.

e.g. we want to create a table called PROJECT to store information about projects. For each project, we want to store the

i. number of the project
ii. name of the project
iii. the employee number of the project's manager
iv. the budget
v. the number of persons working on the project
vi. The start date
vii. The end date of the project

We have the following conditions

a. A project is identified by its project number
b. The name of a project must be unique
c. The manager and the budget must be defined

Create table PROJECT (

PNonumber(3) constraint prj-pk primary key,

PNamevarchar 2(60) unique

PMgr number (4) not null,

Persons number (5)

Budget number (8.2) not null

Pstart date

Pend date

);

A **unique** constraint can include more than one attribute:

unique(<column i>,…,<column j>) is used.

If it is required that no two projects have the same start and end date, we have to add the table constraint.

Constraint no-same-dates unique (PEnd, PStart).This constraint has to be defined in the create table command after both columns PEnd and PStart have been defined.

Three types of constraints:

- not null
- primary key
- unique

Check constraint = to restrict possible attribute values

Foreign key constraint = to specify interdependences between relations

Check constraint

syntax

Constraint<name> check <condition>

Columns in a table must have values that are within a certain range or that satisfy certain conditions. If a check constraint is specified as a column constraint, the condition can only refer that column

e.g

- The name of employees must consist of upper case letters only
- The minimum salary of an employee is 500
- The numbers must range between 10 and 100

Create table EMP

(

eName varchar2(30) constraint check_name

Check (eName = upper(eName)),

SAL number (5,2) constraint check_sal

Check (SAL>= 500)

Deptno NUMBER (3) constraint check_deptno

Check (DEPTNO betwen10 and 100);

Condition can refer to all columns of the table. Not only simple conditions are allowed.

A check condition can include a not null constraint

SAL number(5.2) constraint check_sal

Check (SAL is not null and SAL>=500)

It is allowed to use and, or, not are allowed in the condition.

e.g. at least two persons must participate in a project and project's start date must be before the project's end date.

Create table PROJECT(

Persons number (5)constraint check_person

check(person>2)

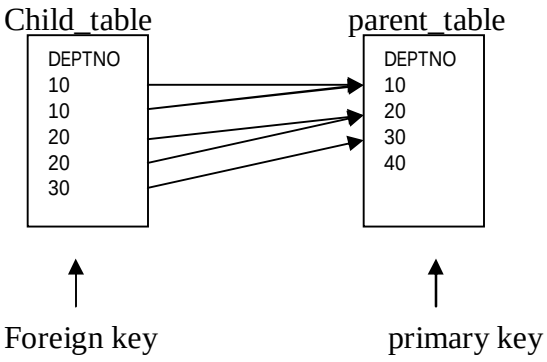
Constraint dates_ok check (PEND >PSTART),

In this task definition, check_person is a column constraint and date_ok is a table constraint.

The database system automatically checks the specified conditions each time a database modification is performed on this relation.

Foreign Key Constraints

EMP



Constraint<name> foreign key <columns>

References<table> [(<columns>)]

On delete cascade

- ✓ A foreign key constraint or referential integrity constraint can be specified as a column constraint or as a table constraint

- ✓ This constraint specifies a columns or a list of columns as a foreign key of the referencing table
- ✓ The referencing table is called the child_table and the referenced table is called the parent _table
- ✓ The clause **foreign key** has to be used in addition to the clause **references** if the foreign key includes more than one column
- ✓ The clause **references** defines which columns of the parent_table are referenced.

e.g. each employee in the table EMP must work in a department that is contained in the table DEPT

```
create table EMP(

EMPNO number (4) constraint pk_emp primary key;

-----

DEPTNO number (3) constraint fk_deptno

References DEPT (DEPTNO));
```

- ✓ Since in this table definition, the referential integrity constraintincludes onlyone column, the clause **foreign key** is not used.
- ✓ It is very important that a foreign key must refer to the complete primary key of a parent_key, not only a subset of the attributes that build the primary key.
- ✓ In order to satisfy a foreign key constraint, each rows in the child_table has to satisfy on of the following two conditions
    1. The attribute value (list of attribute values) of the foreign key must appear as a primary key value in the parent_table or
    2. The attribute value of the foreign key is null

According to the above definition for the tableEMP, an employee must not necessarily work in a department.

e.g. each project manager must be an employee

```
create table PROJECT(

PNO number (3) constraint prj_pkpimary key;

PMGR number(8) not null

        Constraint fk_pmgr reference EMP,

);
```

A constraint can be disabled using the command

Alter table<table_name> disable

Constraint <name>| primary key |unique <columns>

Cascade

To disable a primary key, one must disable all foreign key constraints that depend on this primary key. The clause **cascade** automatically disables foreign key constraints that depend on the (disabled) primary key.

Triggers

- ⇒ Is a procedure
- ⇒ Such a procedure is associated with a table and is automatically called by the database system whenever a certain modification (event) occurs on that table.Modificationson a table may include:
  - i. Insert
  - ii. Delete
  - iii. Update operations

Structure of triggers

A trigger definition consists of the following components:

i. Trigger name
   Create or replace trigger<trigger name>

ii. Trigger time point
   Before/after

iii. Trigger event(s)
   Insert or update of <columns>

   Or delete on <table>

iv. Trigger type (optional)
   For each row

v. Trigger restrictions (only for each row trigger!)
   When <condition>

vi. Trigger body
   <PL/SQL block>

- ❖ The clause replace re-creates a previous trigger definition having the same <trigger name>
- ❖ A trigger can be invoked before or after the triggering event
- ❖ A single event is an insert, an update or a delete. Events can be combined using logical connective or
- ❖ In order to program triggers effectively, it is essential to understand the difference between a row level trigger and a statement level trigger.A row level trigger is defined

using the clause **for each row**.A row trigger executes once for each row after (before) the event.A statement trigger is executed once after (before) the event.If the update affects 20 tuples, the trigger is executed 20 times for each row at a time. A statement trigger is only executed once.

When combining the different types of triggers, there are twelve possible trigger configurations:

Event	**Trigger before**	**Time point after**	**Trigger statement**	**Type row**
Insert	×	×	×	×
Update	×	×	×	×
delete	×	×	×	×

Only with a row trigger it is possible to access the attribut.values of a tuple before and after the modification.

- For an **update** trigger, the **old** attribute value can be accessed using :old<column>
  And the **new** attribute value can be accessed using: new<column>
- For an insert trigger only: new<column> can be used
- For a **delete** trigger only: old<column> can be used

In these cases:

:new<column> refers to the attribute values of <column> of the inserted tuple

:new<column> refers to the attribute value of <column> of the deleted tuple

In a row trigger thus it is possible to specify comparisons between old and new attribute values in the PL/SQL blocks e.g.

if :old.sal<:new. Sal then

<Sequence of statement>

- **When** clause can be used in combination with a for each row trigger.

The trigger body consists of a PL/SQL block. Rollbackand commit can be used in this block.

Three constructs = If inserting
If updating } exit
If deleting

e.g.

```
create or replace trigger emp.check
after insert or delete or update on EMP for each row
begin
        if inserting then
        <PL/SQL block>
        end if
        if updating then
        <PL/SQL block>
        end if
        if deleting then
        <PL/SQL block>
        end if
end;
```

In the PL/SQL blocks of a trigger, an exception can be raised using the statement:

```
        raise_application_error(
```

Raise_application_error can refer to old/ new values of modified rows:

```
raise_application_error(-20010,'salary increase form'
                        || to_char (:Lold.sal)||`to'
                        to_char (: new.sal)||' is for high')
raise_application_error(-20030,'employee id'||
                        to_char(:new.EMPNO)||' does not exit');
```

e.g triggers

Suppose we have to maintain the following integrity constraint:

i. The salary of an employee different from the president cannot be decreased and must also be increased more than 10%
ii. Depending on the job title, each salary must lie within a certain salary range

We assume a table SALGRADE that stores the minimum (MINSAL) and maximum (MAXSAL) salary for each job title (JOB).

Since the above condition can be checked for each employee individually, we define the following row trigger:

```
Create or replace trigger check_salary_EMP

After insert or update of SAL, JOB on EMP for each row

When (new.job|= 'PRESIDENT')------ trigger restriction

declare

minsal, maxsal SALGRADE.MAXSAL%TYPE;

begin

        ------ retrieve minimum and maximum salary for job

    select MINSAL, MAXSAL into minsal, maxsal

    from SALGRADE

    where JOB = :new. JOB;

    ----- if the new salary has been decreased or does not lie within the salary range raise an exception.


    if (:new.SAL<minsal or

    :new. SAL>maxsal) then

    raise_application_error(-20230, 'salary range exceeded')


    else if (:new. SAL<:old. SAL) then

    raise_application_error(-20225, 'salary has been decreased')


    else if (:new. SAL> 1.1 *:old.SAL) then

raise_application_error(-20235, 'more than 10% increase');

end if

end:
```

WEEK – ELEVEN

PL/SQL

Procedure language SQL allows users and designers to develop complex database applications that require the usage of control structures and procedural elements such as

- Procedures
- Functions
- Modules

The basic construct in PL/SQL is a block statement in a PL/SQL block include

- SQL statements
- Control structures (loops)
- Condition statement (if-then-else)
- Exception handling
- Calls of other PL/SQL blocks

PL/SQL blocks that specify procedures and functions can be grouped into packages. A package is similar to a module and has an interface and an implementation part.

PL/SQL offers a mechanism to process query results in a tuple-oriented way, that is, one tuple at a time. For this, cursors are used. A cursor is a pointer to a query result and used to read attribute values of selected tuples into variables. A cursor is used in combination with a loop construct such that each tuple read by the cursor can be processed individually.

STRUCTURES OF A PL/SQL BLOCKS

1. [<block header>] – it specifies whether PL/SQL block is a procedure, function and a package.

2. [declare

<constants>

<variables>

<cursors>

<user defined exceptions>]

Begin

3. <PL/SQL statements>
4. [exception
   <exception handle>]

End;

Declarations constants, variables cursors and exceptions must be declared in the decision section of that block.

<variable name> [constant] <data type> [not null] [:= <expression>];

Boolean type may be true, false, null.

The not null clause requires that declared variable must always have a value different from null. Expression is used to initialize a variable.

Constant mean once a value has been assigned to the variable, the value cannot be changed.

Declare

hire_date date;

job_title varchar2(80) := 'salesman';

emp_found Boolean;

salary_incr constant number (3,2) :=1.5;

….

Begin

…..

End;

Two allowed data type

- column
- many columns

a. Emp.empno%TYPE: refers to the data type of a column EMPNO in the relation EMP.
b. DEPT%ROW TYPE: specifies a record suitable to store all attribute values of a complete row from the table DEPT.

Such records are typically used in combination with a cursor. A field in a record can be accessed using

<record name> <column name>

e.g DEPT, deptno

A cursor declaration specifies a set of tuples (as a query result) such that the tuples can be processed in a tuple-oriented way (i.e one tuple at a time) using the fetch statement.

Format or syntax for cursor

Cursor <cursor name> [<list of parameters>] is <select statement>;

Cursor name should not be any PL/SQL variable.

List of parameters = <parameter name> <parameter type>

Examples of a parameter type are char, varchar2, number, date, Boolean, integer.

Parameters are used to assign values to the variables that are given in the select statement.

e.g we want to retrieve the following attribute values from the table EMP in a tuple-oriented way: the job title and name of those employees who have been hired after a given date, and who have a manager working in a given department.

Cursor employee_cur (start_date date

Dno number) is

Select JOB, ENAME

From EMP

Where HIREDATE > start_date

And exit.

(Select * from EMP where E.MGR = EMPNO and DEPTNO = DNO)

Before a declared cursor can be used in PL/SQL statements, the cursor must be opened and after processing the selected tuples, the cursor must be closed. We use open statement to open a cursor.

Open <currsor_name> [<list of parameter>]

Close statement is used to disable a cursor

Close <cursor name>

Open

- Select statement is processed.
- The cursor references the first selected tuple
- Selected tuples can be processed one tuple at a time using the fetch command

Close

Fetch <cursor_name>

Into <list of variables>;

- Fetch command assigns the selected attribute values of the current tuple to the list of variables.
- After the fetch command, the cursor advances to the next tuple in the result set.

Note that the variables in the list must have the same datatypes as the selected values.

After all the tuples have been processed, the close command is used to disable the cursor.

- PL/SQL does not permit the use of create table statement way to assign a value to a variable.

1. Declare

```
Counter integer := 0;

Begin

    Counter = counter + 1;

End
```

2. Select statement

```
Select <columns>

Into <metching list of variables>

From <tables>

Where <condition>
```

Here, select retrieves one tuple/record, two records pls use cursor.

Instead of a list of single variables, a record can be given after the keyword into. Also, in this case, select statement must retrieve atmost one tuple.

```
Declare

    Employee_rec EMP%ROW TYPE;

    Max_sal EMP.SAL%TYPE;

Begin

    Select EMPNO, ENAME, JOB, MGR, SAL, COMM, HIREDATE,

                                        DEPTNO

    Into employee_rec

    From  EMP
```

```
Where EMPNO = 5698;

Select max(SAL)

Into   max_sal

From    EMP;

End;
```

Loops

1. While loops
2. Two types of for loops
3. Continous loops: continous loop are used in combination with cursors.

1. Label name: name is used incase of nested loop.

   inner loops are completed unconditionally using the exit.

   ```
   Label name

   While <condition> loop

       <sequence of statements>

   End loop [label name]
   ```

2. Label name

   ```
   For <index> in [reverse] lower bound..upper bound loop

   <sequence of statement>

   End loop [label name]
   ```

   - Index is declared implicitly
   - Index can be constant or expression
   - Reverse causes the iteration to proceed downwards from the higher bound to the lower bound.

Cursor for loops can be used to simplify the usage of a cursor;

Label name

For <record_name>

In <cursor name>

<list of parameters> loop

<sequence of statements>

End loop [label name]

It is possible to specify a query instead of <cursor name> in a for loop.

For <record name>

In (select statement) loop

<sequence of statements>

End loop;

e.g for sal_rec

in (select SAL + COMM total from EMP) loop

end loop;

If – then – else

If <condition> then

<sequence of statements>

Else <sequence of statement>

End if;

Except create a table, other command such as delete, update, insert,

Commit are used in PL/SQL.

If update or delete statements are used in combination with a cursor these commands can be restricted to currently fetched tuple. In these cases the clause where current of <cursor name> is added as shown in the example.

The example below illustrates how a cursor is used together with a continous loop.

```
Declare
        Cursor emp_cur is select from EMP;
        Emp_rec      EMP%RPWTYPE
        Emp_sal      EMP.SAL%TYPE
        Begin
              Open emp_cur;
               Loop
                     Fetch emp_cur into emp_rec
                      Exit
                      When emp_cur %NOT FOUND
                       Emp_sal = emp_rec.sal;
                       <sequence of statement>
                 Endloop
                 Close emp_cur
          End.
```

Each loop can be completed unconditionally using the exit clause.

Exit <block label>

When <condition>

- Using exit without a block label causes the completion of the loop that contains the exit statement.
- A condition can be a simple comparison of values.
- In most cases, the condition refers to a cursor.
- %NOT FOUND is a predicate that evaluates to false if the most recent fetch command has read a tuple.
- The value of <cursor name> %NOT FOUND is null before the first tuple is fetched.
- The predicate evaluates to true if the most recent fetch failed to return a tuple and false otherwise.

%found is the logical opposite of %not found.

EX2: the following PL/SQL block performs the following modifications. All employees having KING has their manager get a 5% salary increase.

```
Declare
        Manager EMP.MGR%TYPE;
        Cursor emp_cur (mgr_no number) is
                        Select SAL
                        From EMP
                        Where MGR = mgr_no
      For update of SAL;
          Begin
```

```
Select EMPNO
 Into manager
 From EMP
 Where ENAME = 'KING';
For emp_rec in emp_cur (manager) loop
 Update EMP
  Set SAL = emp_rec.sal * 1.05
  Where current of emp_cur;
End loop;
Commit;
End;
```

Note that the emp_rec is implicitly defined.

EXCEPTION HANDLING

- Two types of exceptions
  - System-defined exceptions
  - User-defined exceptions

System-defined exceptions are always automatically raised e.g

CURSOR_ALREADY OPEN

INVALID_CURSOR

NO_DATA_FOUND

TOO MANY ROWS

ZERO_DIVIDE

User defined exceptions:

1. it uses raise command
   raise <exception name>
2. when <exception name>
   then <sequence of statement>
3. raise_application_error e.g
   declare

```
        emp_sal EMP.SAL%TYPE;
        emp_no EMP.EMPNO%TYPE;
        too_high_sal exception;
begin
        select EMPNO, SAL into emp_no, emp_sal
        from EMP
        where ENAME = 'KING';
            if emp_sal * 1.05 >4000
            then raise too_high_sal
            else
            update EMP
             set SQL…..
            endif;
```

```
exception
    when NO_DATA_FOUND …no tuple selected
then rollback;
when too_high_sal
then insert into high_sal_emps values (emp-no);
commit;
end;
```

raise_application_error = display error or warming messages on the screen.

It has two parameters:

<error number> -is a negative number btw -20000 to -20999.

<message-text> -is a string with < 2048 characters.

To-char is used to convert chart to numeric.

e.g if emp_sal * 1.05 > 4000

___then raise_application_error (-20000, 'salary increase for employee with id' // to-char (emp.no) // 'is too high');

user-defined exception: which must be declared by the user in the declaration part of a block where the exception is used/implemented.

- System defined exception are always automatically raised whenever corresponding errors or warming occur.
- User defined exceptions in contrast must be raised explicitly in a sequence of statements using raise <exception name>

- After the keyword exception at the end of a block, user defined exception handling routines are implemented.

An implementation has the pattern

When <exception name> then <sequence of statements>;

Example of system defined exceptions;

Exception name	**remark**
1. cursor_already_open	You have tried to open a cursor which is already open.
2. INVALID_CURSOR	Invalid cursor operation such as fetching from a closed cursor.
3. NO_DATA_FOUND	A select….into or fetch statement returned no tuple.
4. TOO_MANY_ROWS	A select….into statement returned more than one tuple.
5. ZERO_DIVIDE	You have tried to divide a number by 0.

```
Declare
    Emp_sal EMP.SAL%TYPE;
   Emp_no EMP.EMPNO%TYPE;
    Too_high_sal exception;
Begin
    Select EMPNO
```

```
        SAL into emp_no,emp_sal
    From EMP
    Where ENAME = 'KING';

If  emp_sal * 1.05 > 4000 then
Raise too_high_sal
Else
    Update EMP
    Set sal….
Endif;
Exception
        When NO_DATA_FOUND …. No tuple selected
        Then  rollback;
        When too_high_sal then
        Insert into high_sal_emps
        Values (emp_no);
    Commit;
End;
```

If a PL/SQL program is executed from the SQL plus shell, exception handling routines may contain statements that display error in warning messages on the screen.

For this, the procedure

Raise_application_error can be used.

This procedure has two parameters <error-number> and <message-text>.

Error-number is a negative integer defined by the user and must range between -20000 and -20999.

Error-message is a string with a length upto 2048 characters.

// = the concatenate operator can be used to concatenate single strings to one string.

In order to display numeric variables, these variables must be converted to string using the function to_char. e.g

If emp_sal * 1.05 > 4000 then

Raise_application_error (-20010,'salary increase for employee with id' // to_char (emp_no) // 'is too high');

ETHERNET

- Procedure/language/ SQL ; to implement complex data structure and algorithm
- The basic construct in PL/SQL is a block
- In block constants and variables can be declared
- Variables can be used to store query results
- Statements in a PL/SQL block include :

SQL statements

Control structure (loops)

Control statements(if-then-else)

Exception handling

Cells of other PL/SQL blocks

PL/SQL blocks that specify procedures and functions can be grouped into packages

- A package is similar to module and it has an interface and an implementation part
- Oracle offer several predefined packages e.g

Input/output routines

File handling

Job scheduling

One importance of PL/SQL is that it offers a mechanism to process query results in a triple oriented way i.e one triple at a time. For this, cursors are used. – A cursor is a pointer to query result. –A cursor is used to read attribute values of selected turples into variables. – A cursor is used in combination with a -loop construct such that each turple read by the cursor can be processed individually.

Structure of PL/SQL blocks.

– PL /SQL is a block structured language
- each block builds a program unit
- blocks can be nested.
– a PL/SQL block has an optional declare section

- A part containing PL/SQL statement
- An optional exception handling part

[<block header>] specifies procedure, function, package

[declare

<constants>

<variables>

<cursors>

<user defined exceptions>]

Begin

<PL/SQL statement >

[Exception

<exception handling>]

End.

Declaration

-constant, variables, cursors and exceptions used in a PL/SQL blocks must be declared in the declare section of that block. Eg.

Variables and constant can be declared as follows:

<variable name> [constant] <data type> [not null]

[:= <expression>];

Valid data type = char(n) (n=255 bytes and 2000 in oracle)

- Char(40)

- Varchar2(n) (n=2000 and 4000 in oracle)

Number(o.d) …integer or real

O =overall number of digits

D= number of digits to the right of the decimal point

O=38

D= - 84 to +127

Eg. Number(8)

Number(5.2)

Date = DD-MM-YY 13-oct-06

Long character date up to a length of 2GB.

In oracle-SQL there is no date type booleen.

- Attribute may have the special value null (for unknown)
- This value is different from 0
- It is also different from the empty string

Boolean date may only be true, false or null. The "not null" clause requires that the declared variable must always have a value different from null

-<expression> is used to initialize a variable.
–if no expression is specified, the value null is assigned to the variable. –
the clause constant states that; once a value has been assigned to the variable, the value cannot be changed – the variable becomes a constant. Eg.

```
Declare
        Hire  date       date;  rhg'opo
      Job title          var char 2(80)   I = "sales man" ;
      Emp-found        Boolean;
      Salary_iner       constant          number(3.2)  =1.5
      …….
Begin      …. End;
```

Instead of specifying a date type, one can also refer to the date type of a table column so called anchored declaration. Eg.

EMP. empno % type refers to the date type of the column empno in the relation EMP. DEPT % ROW TYPE specifies a record suitable to store all attribute. Values of a complete row from the table DEPT.

CURSOR

- A cursor declaration specifies a set of turples
- A turple can be processed in a turple-oriented way. Eg, one turple at a time using the fetch statement.
- A cursor declaration has the form;

  Cursor <cursor name> [(< list of parameters >)]
  Is < select statement >;

-cursor name = not the name of any PL/SQL variable
-parameters has the form < parameter name > < parameter type>
Parameter type are char, var char, number, date, Boolean.

E.g. We want to retrieve the following attribute values from the table EMP in a turple-oriented way. The job title and name of those employees who have been hired after a given date and who have a manager working in a given department

```
Cursor  employee_cur (start_date,  date, dno, number)   is
      Select job, Ename
      From EMP   E
      Where HIREDATE > start_date
       AND      exits
         (
            Select
            From  EMP
            Where E.mgr=Emp no
```

AND DEPT NO =dno);

-before a cursor can be used, it must be opened using the open statement.

Open < cursor name > [(<list of parameters>)]

The associated select statement Then is processed and the cursor reference the first selected turple. –selected turples then can be processed one turple at a time using the fetch command.

Fetch < cursor name > into < list of variables >;

- fetch command assigns the selected attribute values of the current turple to the list of variables.
-after the fetch command, the cursor advances to the next turple in the result set
-note that the variables in the list must have the some date type as the selected values.
-after all turples have been processed, the close command is used to disable the cursor

Close < cursor name >

```
Declare
        Cursor emp.cur is
         Select from emp;
       emp-rec    EMP % ROW TYPE;
       emp-sal    EMP.SAL %TYPE;
Begin
     Open  emp.cur;
       Loop
              Fetch emp.cur into emp-rec;
              Exit
              When emp.cur % NOT FOUND;
               Emp-sal  I = emp-rec.sal;
    End loop
   Close  emp.cur;
End;
```

Exceptions are used to process errors .

% NOT FOUND is a predicate that evaluates to false if the most recent FETCH command has read a turple.

The predicate evaluates to true if the most recent FETCH failed to return a turple,and false otherwise.

% FOUND is the logical opposite of % NOT FOUND.

Language elements

(1) variable assignments

(2) control structures - loops (while and for)
-if –then- else

(3) procedure and function calls

1. variable assignments

```
declare
        counter   integer   i = 0;
       begin
         counter  i =counter + i;
while =
     while < condition > loop
           < sequence of statement >;
     End  loop
For  < index >  in [ reverse ]
        < lower bound > .. <upper bound> loop
        <sequence of  statements >
     End  loop
```

Reverse = causes the iteration to proceed downwards from the higher bound to the lower bound.

```
If < condition> then
   <sequence of statements>
Else   < sequence of statement>
End if;
```

The following PL/SQL block performs the following modification. All employees having "KING" as their manager get a 5% salary increase.

```
Declare
        Manager   EMP.MGR  % TYPE;
        Cursor emp.cur (mgr-no  number ) is
            Select SAL
             From   EMP
            Where   MGR = mgr-no
For  update  of  SAL;
```

```
Begin
        Select EMPNO  into   manager
        From    EMP
       Where    ENAME =  “king”;
 For  emp-rec in emp-cur (manager)  loop
          Update  EMP
          Set    SAL =emp-rec.sal   # 1.05
         Where  current of  emp.cur;
 End loop;
Commit;
End;
```

Procedure syntax

```
Create [ or replace ] procedure < procedure name >
  [ ( list of parameters >)]
            Is
   <declaration>
Begin
    < sequence of statement >
   [exception
        < exception  handling routine >]
End   [ < procedure  name >]
```

Function syntax

```
Create [or replace] function  < function  name >
    [ (< list of parameters >) ]
```

Return < date type > is

Deleted procedure or function by using:

Drop procedure < procedure name >

Drop function < function name >

This procedure is used to increase the salary of all employees who work in the department given by the procedure's parameter. The percentage of the salary increase is given by a parameter too.

```
Create procedure raise-salary (Dno number, percentage number DEFAULT o.s)
                                        Is
Cursor emp_cur (dept-no number) is
        Select SAL
        From EMP
        Where DEPTNO = dept-no
          For update of SAL;
Empsal number(8);

Begin
                Open emp-cur (dno); ……here dno is assigned to dept-no
                 Loop
                 Fetch emp-cur into empsal;
                 Exit
                When emp-cur %NOTFOUND;
                Update EMP
                Set SAL = empsal * ((100 + percentage) /100)
                Where current of emp-cur;
```

```
        End  loop;
        close emp-cur;
        commit;
end   raise-salary
```

this procedure can be called from the SQL plus shell using the command

```
execute   raise-salary (10,3);
```

if a procedure is called from a PL/SQL block the keyword execute is omitted.

Functions have the same structure as procedures. The only difference is that a function returns a value whose data type (un constrained) must be specified.

```
Create function get-dept-salary (dno  number)
   Return  number      is
       All-sal   number;
Begin
        All-sal  i=0;
       For  emp-sal    in
            (select   SAL
             From     EMP
             WHERE   DEPTNO  =  DNO
             AND    SAL   is   not  NULL)     loop
      All-sal  :=  all-sal   +   emp-sal.sal;
     Emp  loop;
    Return  all-sal;
End get-dept-salary;
```

In order to call a function from SQL plus shell, it is necessary to first define a variable to which the return value can be assigned.

Variable <variable name > < date type >;

E.g.

Variable salary number

Execute : salary i= get-dept-salary (20);

Note that the colon : must be put in front of the variable.

We use an after trigger because the inserted and updated row is not changed within the PL/SQL block (eg.in case of a constraint violation, it would be possible to restore the old attribute values).

Note that also modification on the table SALGRADE can cause a constraint violation. In order to maintain the complete condition we define the following trigger on the table SALGRADE. In case of a violation by an update modification however, we do not raise an exception, but restore the old attribute values.

Create or replace trigger check-salary-SALGRADE before update or delete on SALGRADE for each row.

When (new.MINSAL > old.MINSAL or new. MAXSAL < old.MAXSAL)

Nly restricting a salary range can cause a constraint violation

Declare

Job-emps number (3) i=0;

Begin

If deleting then :…….does there exist an employee having the deleted job?

Select count(*)

Into job-emps

From EMP

Where job = iodd.job;

If job.emps !=0 then…….. (Raise-application-error (-20240," there still exist employee with the job" II :old.job));

End if

End if

If updating then ……. (are there employees whose salary does not lie within the modified salary range?)

Salary count(*)

Into job-emps

From EMP

Where JOB = : new.JOB

AND SAL not between : new. MINSAL and

:new .MAXSAL;

If job-emps ! =0 then …… (restore old salary ranges)

: new .MINSAL i = iold. MINSAL;

:new .MAXSAL i= iold.MAXSAL;

End if;

End if (in this case a "before" trigger must be used

to restore the old attribute values of an

updated row.)

end.

Suppose we furthermore have a column BUDGET in one table DEPT that is used to store the budget available for each department.

Assume the integrity constraint requires that the total of all salaries in a department must not exceed the department budget.

Critical operations on the relation EMP are insertions into EMP and updates on the attributes SAL or DEPTNO

```
Create or replace trigger check-budget-EMP after insert or update of SAL DEPTNO on EMP
 Declare
      Cursor  DEPT-CURR  is
       Select  DEPTNO  BUDGET
       From  DEPT,
       DNO     DEPT.DEPTNO % TYPE;
       ALL SAL      DEPT.BUDGET % TYPE;
        DEPT-SAL      number;
Begin
         Open   DEPT-CURR;
         Loop
             Fetch  DEPT-CURR  into  DNO,  ALL SAL
             Exit  when  DEPT-CURR % NOT FOUND,
                    Select   sum (SAL)
                    INTO   DEPT SAL
                    FROM   EMP
                    WHERE  DEPTNO =DNO;
If    DEPT-SAL  >  ALL SAL  then
                       (  raise-application- error     (-20325, "total of salaries in the
                            department  II to char (DNO)   II "  exceeds budget) )
                  end if
            end  loop
        close  DEPT-CURR
end.
```